Faster Pseudorandom Correlation Generators via Walsh-Hadamard Transform

Zhe Li[1], Hongqing Liu[2(✉)], Chaoping Xing[2(✉)], Yizhou Yao[2(✉)], Chen Yuan[2(✉)]

[1] Institute of Mathematics and Interdisciplinary Sciences,
Xidian University, Xi'an, China
lizh0048@e.ntu.edu.sg
[2] School of Computer Science,
Shanghai Jiao Tong University, Shanghai, China
{liu.hong.qing,xingcp,yaoyizhou0620,chen_yuan}@sjtu.edu.cn

Abstract. The past few years have witnessed the growing importance of pseudorandom correlation generators (PCGs) for generating correlated randomness with sublinear communication. To date, quasi-linear time PCGs for oblivious linear evaluation (OLE) over arbitrary finite fields have been constructed under either Ring-LPN or Quasi-Abelian syndrome decoding (QA-SD) assumptions, with a throughput of millions of OLEs per second demonstrated, in particular, for the binary field. However, many modern MPC protocols deal with large prime fields, in which existing PCGs suffer from a significant efficiency gap due to a quasi-linear number of *multiplications* involved in FFT (Fast Fourier Transform) algorithms. Moreover, FFT typically relies on FFT-friendly fields that contain large smooth multiplicative subgroups, and therefore are not well-suited to popular fields, such as Mersenne prime fields.

In this work, we close the gap by leveraging the well-known Walsh-Hadamard transform (WHT) in the context of QA-SD based PCGs. Although WHT is still a quasi-linear time algorithm like normal FFTs, no multiplication is needed — addition and subtraction suffice. Since multiplications over a prime field $\mathbb{F}_p$ typically incur an $O(\log p)$ overhead over additions, our scheme that avoids a large number of multiplications perfectly fits the large prime field setting. Moreover, we show that with overwhelming probability, the underlying random QA code has a high relative distance, e.g., 0.268 for rate 1/4. Experimental results show that WHT is at least one magnitude faster than FFT over a 64-bit smooth prime field. Consequently, our PCG achieves $27,000$ OLE per second over a 64-bit prime field. This is the first full implementation of PCG for OLE over an arbitrary large prime field that we are aware of.

We then build PCG for vector-OLE over arbitrary large prime fields from QA-SD assumptions, and fully implement it using the libOTe library. We achieve a throughput of over $5,000,000$ vector-OLEs per second over a 64-bit prime field, roughly four times faster than state-of-the-art PCGs from either expand-accumulate (EA) codes (Boyle et al., CRYPTO 2022), or expand-convolute (EC) codes (Raghuraman et al., CRYPTO 2023).

Table of Contents

1 Introduction . . . 2
1.1 Our Contribution . . . 4
1.2 Related Works . . . 8
1.3 Technical Overview . . . 10
2 Preliminaries . . . 12
2.1 Basic Notations . . . 12
2.2 Pseudorandom Correlation Generators . . . 12
2.3 Function Secret Sharing and Puncturable PRF . . . 13
2.4 Syndrome Decoding of Quasi-Abelian Codes . . . 14
2.5 Ring Isomorphism . . . 15
3 Walsh-Hadamard Transform . . . 16
4 Minimum Distance of Quasi-Abelian Codes . . . 18
5 PCG for VOLE over Large Prime Fields . . . 20
6 PCG for OLE over Large Prime Fields . . . 23
7 Efficiency Analysis . . . 25
7.1 Parameter Selection . . . 25
7.2 Implementation and Evaluation . . . 26
A More Preliminaries . . . 35
A.1 PCG . . . 35
A.2 PPRF . . . 36
A.3 Properties of Group Rings . . . 37
B Missing Proofs . . . 38
B.1 Proof for VOLE . . . 38
B.2 Proof for OLE . . . 39
C PCG Setup Protocols . . . 40
D Concrete Attacks to QA-SD . . . 41
E Stationary Syndrome Decoding of Quasi-Abelian . . . 44
E.1 Definition of QA-SSD . . . 44
E.2 PCG Based on QA-SSD Assumption . . . 45
E.3 Distributed Setup from QA-SSD . . . 47
E.4 Security Analysis for QA-SSD . . . 49
E.5 Parameter Selection for QA-SSD . . . 50

1 Introduction

Correlated randomness stands as a fundamental and crucial part of secure multi-party computation (MPC) protocols with pre-processing. Modern MPC protocols, e.g., SPDZ-line of works [DPSZ12, KOS16, CDE+18, RS22, EXY22], typically follow the paradigm of Beaver [Bea91] to prepare (authenticated) multiplication triples in the pre-processing phase. However, for the implementation of

pre-processing phase, classic approaches based on either homomorphic encryption [DPSZ12, CDE+18] or oblivious transfer [KOS16, EXY22], incur $\Omega(|\mathcal{C}|)$ communication cost. Concretely, gigabytes and terabytes of communications are required for typical circuit sizes ranging from 10^6 to 10^9.

Generating multiplication triples has been a major bottleneck for improving the performance of MPC protocols. Recently, a new approach initiated in [BCG+17, BCGI18, BCG+19b] suggests generating pseudorandom correlations that are computationally indistinguishable from the ideal target correlations. This approach, abstracted by the notion of a pseudorandom correlation generator (PCG) [BCG+19b], substantially reduces the communication complexity to $O(\mathsf{poly}(\lambda) \cdot \log |\mathcal{C}|)$, where λ is the security parameter. Since then, there have been numerous subsequent works [BCG+19a, YWL+20, BCG+20b, BCG+22, BCG+23, RRT23, BCCD23, ABG+24, BBC+24, LXYY25a, LXYY25b, KPRR25] that aim to not only generalize PCGs for more types of correlations, but also improve the concrete efficiency. Eventually, based on PCGs for OLE (oblivious linear evaluation) and PCGs for VOLE (vector oblivious linear evaluation), [RS22] constructs LeMans, the state-of-the-art SPDZ-line protocol.

To construct efficient PCGs, Boyle et al. [BCGI18] put forward a general template that basically consists of two steps: (i) using distributed multi-point functions (DMPF) [GI14, BGI16, BGH+24] to succinctly distribute "sparse" correlated randomness, and (ii) expanding into long pseudorandom correlations via syndrome decoding of random linear codes, a long-standing hard problem from coding theory. Notably, both steps are dominant in the overall computational complexity. To improve step (ii), a line of work [CRR21, BCG+22, RRT23, BCH+25, KPR+25] suggests replacing by a code with fast encoding (more precisely, syndrome computing) algorithms, such as LDPC codes [CRR21], expand-accumulate (EA) codes [BCG+22], expand-convolute (EC) codes [RRT23], repeat-accumulate-accumulate (RAA) codes [BCF+25] and more recently block-accumulate (BA) codes [KPR+25]. However, due to insufficient understanding of the minimum distance of LPDC codes, [CRR21] was later broken in [RRT23]. Although formal analyses on the minimum distance of the underlying codes are given in detail [BCG+22, RRT23, BCF+25], their analyses are restricted to the binary field case due to a combinatorial nature. And it seems extremely difficult to extend to large prime fields, which are of great significance due to wide usage in SPDZ-line protocols and VOLE-based highly efficient proof systems [BDSW23]. Towards solving this issue, these works follow a heuristic that moving to large fields should yield better minimum distance, and thus it is plausibly secure to use parameters derived from the binary field.

Although the encoding algorithms of those codes [BCG+22, RRT23, BCF+25, KPR+25] are concretely close to linear time[3] and do not rely on a specific field, we note that they mostly suit constant-sized fields or even binary fields. In this

[3] Asymptotically, both EA code and EC code are quasi-linear time for achieving a constant relative minimum distance. However, in practical regimes, they are concretely close to linear time, in particular when instantiated with reasonable but aggressive parameters. Thus we treat them as linear-time encodable codes as well.

setting, both multiplication and addition can be done in almost the same time. However, when migrated to large prime fields as in [BCG+22, RRT23, KPR+25], the multiplication appears to consume much more computational resources than the addition. To see this, for a field $\mathbb{F}_p$ of size p, one multiplication takes $\Omega(\log^2 p)$ bit operations while one addition only takes $\Omega(\log p)$ bit operations. Hence, if we adapt these codes to some widely used prime field $\mathbb{F}_p$, e.g., a Mersenne prime $p = 2^{61} - 1$ [WYX+21, LXY24], the multiplication operations immediately become the bottleneck for the encoding algorithm. Despite being somewhat counterintuitive, it might be the case that linear-time encodable codes are slower than some quasi-linear ($O(N \log N)$) time algorithms that involve fewer multiplications, where N is the code length, and typically $N \approx 2^{20}$. Even if we pretend that multiplication and addition take almost the same time, an $O(N \log N)$ algorithm with a small hidden constant can still be faster than a linear-time algorithm. This is because the constant factor overhead in a linear time algorithm may be bigger than $\log N \approx 20$. This is exactly the case for our new PCG construction.

Last but not least, a general linear code with fast encoding only admits efficient PCGs for VOLE. Unfortunately, for OLE it requires at least quadratic computation following the blueprint of [BCG+19b, BCG+20b]. To obtain quasi-linear time PCGs for OLE, follow-up works [BCG+20b, BCCD23] suggest relying on either Ring-LPN assumptions[4], or syndrome decoding of quasi-abelian (QA) codes, which enjoy a polynomial ring structure that allows performing Fast Fourier Transform (FFT). However, FFT-based encoding algorithms typically rely on a specific FFT-friendly field, making the scheme non field-agnostic in terms of generating VOLE and OLE.

From the above discussions, we are motivated to find a linear code over arbitrary large prime fields, such that (i) has good provable minimum distance, (ii) encoding can be done in almost $N \log N$ time and the number of multiplications could be $O(N)$; (iii) involves a polynomial ring structure that enables quasi-linear time PCGs for OLE.

1.1 Our Contribution

In this work, we show a family of *quasi-abelian* (QA) codes satisfying the above requirements by giving an efficient syndrome computing algorithm based on the Walsh-Hadamard transform. Following the PCG literature, we use a parity-check matrix $H \in (\mathbb{F}_p[\mathbb{G}])^{k \times c}$ to specify a QA code $\mathcal{C}$ of rate $(c-k)/c$, where $\mathbb{F}_p[\mathbb{G}]$ is a group ring defined by a field $\mathbb{F}_p$ and an abelian group $\mathbb{G}$ (we refer to Section 2.4 for more details). For simplicity, we focus on a systematic matrix H with $k = 1$, and denote $N = |\mathbb{G}|$. In this case, $H \cdot \mathbf{e} \in \mathbb{F}_p[\mathbb{G}]$ is called a syndrome, where $\mathbf{e} \in (\mathbb{F}_p[\mathbb{G}])^c$, and multiplication is defined over $\mathbb{F}_p[\mathbb{G}]$. We denote c as the *compression factor*.

[4] Although the problem is historically called Ring-LPN, we remark that the concrete instantiation in [BCG+20b] can be viewed as syndrome decoding of quasi-cyclic codes.

Theorem 1.1 (Informal). *There exists a family of quasi-abelian codes with code rate $(c-1)/c$ and code length cN such that a syndrome can be computed in $cN \log N + (c-1)N$ additions/subtractions in $\mathbb{F}_p$, and $(c-1)N$ multiplications in $\mathbb{F}_p$.*

We note that for rate $1/2$, and $N = 2^{20}$, we only need $41N$ additions/subtractions and N multiplications in $\mathbb{F}_p$. In comparison, a widely used instantiation of EA code [BCG+22] (expander weight 21 and rate $1/5$) needs $21N$ multiplications and $26N$ additions in $\mathbb{F}_p$. This implies that even if we merely compare the number of total operations, our QA codes could still be faster than some linear-time encodable codes. Theorem 1.1 actually applies to arbitrary QA codes over $\mathbb{F}_p[\mathbb{G}]$ with $\mathbb{G} = \mathbb{Z}_2^n$ and any prime $p > 2$. In this case, one can naturally identify $\mathbb{F}_p[\mathbb{G}]$ with the multivariate polynomial ring $\mathbb{F}_p[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^2 - 1, \ldots, \mathtt{X}_n^2 - 1)$, which is isomorphic to $\mathbb{F}_p^N$.

Concrete Minimum Distance. It is well-known that a random QA code asymptotically approaches the Gilbert-Varshamov (GV) bound [FL15, FL22], when $c \to \infty$ with $k/c, |\mathbb{G}|$ being fixed. However, we are interested in QA codes with a small constant c and the group size $|\mathbb{G}|$ could be as large as 2^{20}. This mismatches results in a large gap on the concrete minimum distance. For instance, from [FL22, Lemma 6.2] one can only guarantee distance 0.01 with probability at least $1 - 2^{-22.5}$, for a random QA code $\subset (\mathbb{F}_p[\mathbb{Z}_2^n])^2$ with rate $1/2$, where p is a 64-bit prime and $n = 20$.

Inspired by the approach of [FL22], we give a better bound on the minimum distance of QA codes. The results are stated as follows:

Theorem 1.2. *Let $H_p(\cdot)$ be the p-ary entropy function and $k, c \in \mathbb{N}$. Let $\mathbb{G} = \mathbb{Z}_2^n$ with $N := 2^n$ and $\gcd(p, N) = 1$. Assume that $H_p(\delta) + \frac{k + \log_p N}{c} < 1$. Then, a random QA code $\mathcal{C} \subseteq (\mathbb{F}_p[\mathbb{G}])^c$ of rate k/c has relative minimum distance $\leq \delta$ with probability*

$$\lessapprox N \cdot p^{-c(1-H_p(\delta)-k/c)} = p^{k+\log_p(N)-c(1-H_p(\delta))}.$$

We derive concrete minimum distances of QA codes from Theorem 1.2, and compare them to fast codes proposed in the literature. Details are given in Table 1. Note that our analyses are over large $\mathbb{F}_p$, while the provable minimum distance of other codes[5] is only known for $\mathbb{F}_2$. We remark that the condition in Theorem 1.2 contains an additive gap $\frac{\log_p N}{c}$ relative to the classic GV bound. The proof of Theorem 1.2 is in Section 4.

[5] Juxtaposed EA is a variant of EA, and has been analyzed over arbitrarily large fields [BFK+24]. However, [BFK+24] requires expander weight $\geq 18 \log N$, which is much larger than that of standard EA code [BCG+22] used in the PCG literature, making it less efficient.

Field	Code	Rate r	Distance δ	Failure Error
$\mathbb{F}_2$	EA [BCG+22]	1/5	0.05	0.00721
	EC [RRT23]	1/2	0.05	$2^{-28.6}$
	BA [KPR+25]	1/2	0.1	2^{-20}
	RAA [BCF+25]	1/4	0.19	$2^{-25.4}$
64-bit $\mathbb{F}_p$	Juxtaposed EA [BFK+24]	1/4	0.1	2^{-100}
	QA	1/2	0.1	$2^{-30.2}$
		1/4	0.268	2^{-100}
		1/4	0.3	$2^{-71.6}$
		1/5	0.4	$2^{-73.1}$

Table 1: Concrete minimum distance of fast codes proposed in the literature. Here the code dimension is set as 2^{20}.

Faster PCGs from QA codes. We leverage Walsh-Hadamard Transform to build faster PCGs over $\mathbb{F}_p$ based on the syndrome decoding problem for quasi-abelian codes (QA-SD). We note that QA-SD assumptions have recently been extensively introduced and studied in the PCG scenarios [BCCD23, BBC+24, LXYY25a, LXYY25b, BDHV25]. However, their focus is on the construction of PCGs for OLE correlation over very small fields, such as $\mathbb{F}_2$. Instead, we focus on PCGs over large prime fields for both OLE and VOLE.

QA-SD Assumption over large fields. Notably, a recent work [BDHV25] shows a concrete attack that breaks QA-SD assumptions over constant-sized fields for many proposed parameters [BBC+24, LXYY25a]. Specifically, suppose $c = 2$ and a systematic $H := (1, a) \in (\mathbb{F}_p[\mathbb{G}])^2$. The QA-SD assumption states that the syndrome $b = as + e$ is computationally indistinguishable from uniform over $\mathbb{F}_p[\mathbb{G}]$, where $s, e \in \mathbb{F}_p[\mathbb{G}]$ are of small Hamming weight. In the following, we instantiate $\mathbb{F}_p[\mathbb{G}]$ as the polynomial ring $\mathcal{R} := \mathbb{F}_p[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^2 - 1, \ldots, \mathtt{X}_n^2 - 1) \simeq \mathbb{F}_p^N$, where $p > 2$ is a prime. The attack of [BDHV25] basically follows from the intuition that a random element $a \in \mathcal{R}$ has roughly $\frac{1}{p}$ fraction of points that evaluates to 0. For each such point α, we have $b(\alpha) = e(\alpha)$. This means one can obtain roughly $\frac{1}{p}$ fraction of evaluations for e, which is sufficient to recover e as e (of small Hamming weight) is a sparse polynomial. However, such an attack fails if the field $\mathbb{F}_p$ is sufficiently large. Thus, our instantiation over a large prime field $\mathbb{F}_p$ is immune to this attack. Moreover, given the concretely high minimum distances of QA codes from Table 1, one can readily show that QA-SD over large fields is resilient to linear attacks [BCG+20a, BCG+22].

PCG for VOLE. We employ Theorem 1.1 to efficiently compute the syndrome $b = as + e$, using only $N \log N$ additions and N multiplications over $\mathbb{F}_p$. Although it is a quasi-linear algorithm, it significantly outperforms previous linear-time encoding algorithms when p is large, i.e., p is a 32-bit or 64-bit prime. Moreover, our PCG for VOLE also benefits substantially from the good minimum distance of QA codes. We fully implement the encoding of the QA code and the corresponding

VOLE protocol. Experimental results are summarized in Table 2, and further details are given in Section 7. The source code is available at https://anonymous.4open.science/r/WHTPCG-D4AB/.

	N	$\log_2 p \approx 32$		$\log_2 p \approx 64$	
		Runtime of encoding	Runtime of VOLE	Runtime of encoding	Runtime of VOLE
EA	2^{20}	157 ms	545 ms	190 ms	714 ms
EC	2^{20}	168 ms	542 ms	171 ms	553 ms
QA	2^{20}	15 ms	123 ms	40 ms	148 ms
EA	2^{25}	7.61 s	24.44 s	7.82 s	28.39 s
EC	2^{25}	6.82 s	22.03 s	7.29 s	23.34 s
QA	2^{25}	0.65 s	4.43 s	1.73 s	5.94 s

Table 2: Comparison of state-of-the-art VOLE protocols to generate a VOLE instance of length N over $\mathbb{F}_p$ with 128-bit security. All protocols were executed on a single core of AMD EPYC 9754 processor, with run time averaged across ten trials. "EA" and "EC" refer to expand-accumulate and expand-convolute code, respectively. The code rates of EA, EC and our QA code are set to 0.2, 0.5 and 0.5, respectively. For EA code, we use expander weight $w = 21$. For EC code, we use expander weight $w = 7$ and convolute state size $m = 24$.

PCG for OLE. As shown in [BCG+19b, BCG+20b], dual-LPN assumptions from linear time encodable codes can be used to build PCGs for OLE correlations. However, these suffer from $O(N^2)$ computational complexity, which is impractical for typical parameters such as $N \approx 2^{20}$. Quasi-linear time PCGs are first constructed from Ring-LPN assumptions in [BCG+20b], where to apply Fast Fourier Transform (FFT), the field size p must satisfy $2N \mid p-1$ for some $N = 2^n$, ruling out Mersenne primes. Subsequently, [BCCD23] gives quasi-linear time PCGs from QA-SD assumptions focusing on small fields, and follow-up works [BBC+24, LXYY25a] give concrete evaluations for OLE over $\mathbb{F}_2$. Although generally the approach of [BCCD23] works for any field of size ≥ 3, to our best knowledge, PCGs tailored for generic large prime fields have not been investigated and not yet implemented.

By applying Theorem 1.1 to the framework of [BCG+20b, BCCD23], we obtain a *field-agnostic* PCG for OLE, where the seed expansion procedure takes only $6N \log N + 4N$ additions and $3N$ multiplications over the underlying prime field $\mathbb{F}_p$, plus $O(\lambda N)$ calls to a secure PRG, where N is the number of OLE correlations. We fully implement it with a *distributed setup* in the libOTe library. It is worth mentioning that the distributed DPF key generation protocols employed in our PCG are defined over the group $\mathbb{G} = \mathbb{Z}_2^n$, which is the native domain for existing distributed DPF setup protocols [BGI16, DS17, GYW+23].

In contrast, the OLE protocol based on Ring-LPN assumptions requires that the distributed DPF key generation should be defined over $\mathbb{Z}_N$ [BCG+20b]. This requires expensive BitAdd calls to convert additions over $\mathbb{Z}_N$ into shares over $\mathbb{Z}_2^m$ for some $m \in \mathbb{N}$, which in turn rely on garbled circuit based protocols [KSS09]. This makes the implementation of such a distributed DPF impractical and thus the OLE protocols based on Ring-LPN have not been implemented yet. Moreover, the standard binary DPFs are more efficient than the ternary DPFs used in [BBC+24, LXYY25a, LXYY25b] because of the optimizations from [BGI16]. From Table 7, we conclude that our OLE protocol generates roughly 27000 OLEs every second.

In addition, both of our PCGs for VOLE and OLE are *programmable*, and moreover, they can share the same QA-SD instance. This enables the efficient construction of multi-party authenticated multiplication triples [DPSZ12], following a similar approach of [RS22]. Notably, prior to our work, state-of-the-art PCGs for VOLE and OLE are based on *different* assumptions, mismatching the conditions of [RS22].

Security Parameter. To prove the pseudorandomness of LPN assumption, we need to analyze the weight distribution of linear codes. Such analysis usually provides very conservative and inefficient parameters. Moreover, the theoretical analysis is quite involved. In contrast, our new QA-SD assumption follows the approach of [BCCD23, LXYY25a, LXYY25b] for random quasi-abelian codes. Since a random quasi-abelian code is asymptotically good and we show a concretely high relative minimum distance, it is not surprising that the weight parameter t can be chosen much smaller than the LPN assumption relying on linear-time encodable codes.

Extending to Stationary QA-SD Assumption and Beyond. Recently, Kolesnikov et al. [KPRR25] proposed a stationary syndrome decoding assumption that allows the reuse of the error locations. Roughly speaking, their new assumption says that the syndromes $H_1\mathbf{e}_1, \ldots, H_m\mathbf{e}_m$ are pseudorandom vectors if $\mathbf{e}_i$ are random t-sparse vectors with the same support set and H_i are public random parity-check matrices. This leads to a more efficient implementation of the DMPF scheme. In this work, we also generalize to stationary QA-SD assumptions, which leads to a similar improvement as in [KPRR25]. We refer to Section E in the Supplementary Material for details. Notably, a concurrent work [ARR25] further improves the distributed setup for DMPF via *reverse cuckoo hash*, which allows them to build efficient PCG for OLE from stationary Ring-LPN. We believe that stationary QA-SD is more suitable in this context, and leave incorporating their approach as future work.

1.2 Related Works

PCG based on LPN assumptions. A VOLE is a degree-1 correlation: $(\mathbf{x}_0, \Delta) \in \mathbb{F}_p^N \times \mathbb{F}_p$ and $(\mathbf{x}_1, \mathbf{v}) \in \mathbb{F}_p^N \times \mathbb{F}_p^N$ such that $\mathbf{x}_0 + \mathbf{x}_1 = \Delta \cdot \mathbf{v}$. Boyle et al. [BCGI18, BCG+19a] proposed the PCG construction for VOLE based on LPN or syndrome

decoding assumption. Basically speaking, PCG based on the syndrome decoding assumption samples a public random matrix H such that $H\mathbf{e}$ is computationally indistinguishable from uniform, where $\mathbf{e}$ is a secret t-sparse vector. Once $\mathbf{e}$ is chosen, a function secret sharing (FSS) scheme (for multi-point functions) is invoked to distribute an additive sharing $(\mathbf{y}_0, \mathbf{y}_1)$ with $\mathbf{y}_0+\mathbf{y}_1 = \Delta\cdot\mathbf{e}$ in $O(t \log N)$ communications. The last step is to locally compute the pseudorandom vectors $\mathbf{x}_i = H\mathbf{y}_i$ and $\mathbf{v} = H\mathbf{e}$. The efficiency of the PCG depends on the implementation of FSS scheme and the computation of $H\mathbf{y}_i$ and $H\mathbf{e}$. If H is a matrix, then the computational complexity is $O(N^2)$. Thus, it is desirable to design a structured H that preserves the pseudorandomness of $H\mathbf{e}$. In [BCGI18, BCG+19a], they proposed sparse LPN or quasi-cyclic codes which provide a reasonable trade-off between security and efficiency. To further improve the efficiency, Boyle et al. [BCG+22] introduced the expand-accumulate (EA) code, a linear-time encodable code. A subsequent work by Raghuraman et al. [RRT23] proposed the expand-convolute (EC) code, which achieves roughly twice the speed of EA codes while maintaining the same security level.

PCG based on Ring-LPN assumptions [BCG+20b]. At a high level, the PCG construction of [BCG+20b] first generates an OLE instance over a polynomial ring $\mathcal{R} = \mathbb{F}_p[\mathtt{X}]/(F(\mathtt{X}))$, where $\mathbb{F}_p$ is a large prime field and $F(\mathtt{X})$ is a degree-N polynomial. To convert this OLE instance over $\mathcal{R}$ into N OLE instances over $\mathbb{F}_p$, an isomorphism between $\mathcal{R}$ and $\mathbb{F}_p^N$ is required. To enable an efficient computation of this isomorphism, $F(\mathtt{X})$ is chosen as a cyclotomic polynomial with $N = 2^n$, and the prime p satisfies $p \equiv 1 \pmod{2N}$. This allows the isomorphism to be computed in $O(N \log N)$ time using FFT algorithms.

The Ring-LPN assumption [HKL+12] is introduced to efficiently generate an OLE instance over $\mathcal{R}$. Given a public random element $a \xleftarrow{\$} \mathcal{R}$ and two random t-sparse elements $s, e \leftarrow \mathcal{R}$, the Ring-LPN assumption states that $a \cdot s + e$ is pseudorandom. For $\theta \in \{0, 1\}$, party P_θ is given two t-sparse polynomials $s_\theta, e_\theta \leftarrow \mathcal{R}$ and sets $x_\theta = a \cdot s_\theta + e_\theta$ as a pseudorandom element according to the Ring-LPN assumption. Then we have

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

Since a is public, the additive shares of $x_0 \cdot x_1$ can be derived from the additive shares of four cross-terms $s_0 s_1, s_1 e_0, s_0 e_1, e_0 e_1$. For $\theta \in \{0, 1\}$, s_θ, e_θ are t-sparse polynomials with degree$< N$, therefore these four cross-terms are t^2-sparse polynomials with degree$< 2N - 1$, which can be succinctly shared by FSS schemes.

PCG based on quasi-abelian syndrome decoding assumptions [BCCD23]. The PCG construction of [BCG+20b] inherently requires that field size p is larger than the number of OLE instances N, which does not work for small fields. To address this problem, Bombar et al. [BCCD23] propose the quasi-abelian syndrome decoding (QA-SD) assumption, which could be regarded as a multivariate version of the Ring-LPN assumption. The main difference is that QA-SD considers the

ring $\mathcal{R} = \mathbb{F}_q[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(F(\mathtt{X}_1), \ldots, F(\mathtt{X}_n))$ with $n > 1$, where $F(\mathtt{X}) = \mathtt{X}^d - 1$ and $d \mid q - 1$. In this case, an OLE instance over $\mathcal{R}$ can be converted into $N = d^n$ OLE instances over $\mathbb{F}_q$. However, this construction cannot directly produce OLE instances over $\mathbb{F}_2$ due to the field size limitation $d \mid q - 1$, which implies $q \geq 3$. Although Bombar et al. [BBC+24] provide a method to transform OLE instances over $\mathbb{F}_4$ to multi-party multiplication triples over $\mathbb{F}_2$, such a transform incurs additional linear communication overhead.

PCG based on trace functions [LXYY25a, LXYY25b]. To directly generate OLE instances over $\mathbb{F}_2$, Li et al. [LXYY25a] introduce a novel tool: the trace function over the polynomial ring. For a base field $\mathbb{F}_p$ and its extension field $\mathbb{F}_q \coloneqq \mathbb{F}_{p^k}$, the trace function $\mathsf{Tr}_{\mathbb{F}_q} : \mathbb{F}_q \to \mathbb{F}_p$ is defined as $\mathsf{Tr}_{\mathbb{F}_q}(x) \coloneqq \sum_{i=0}^{k-1} x^{p^i}$. By extending the domain from $\mathbb{F}_q$ to a polynomial ring $\mathcal{R}$ over $\mathbb{F}_q$, the authors of [LXYY25a] redefine a new trace function $\mathsf{Tr}_{\mathcal{R}}$ to locally map some correlation over $\mathcal{R}$ into multiple OLE instances over $\mathbb{F}_p$. As there is no restriction on k, their construction can be established on either Ring-LPN or QA-SD assumptions. Furthermore, a subsequent work [LXYY25b] proposes PCG constructions for OLE over an arbitrary integer ring $\mathbb{Z}_{p^k}$ by adapting the trace function to Galois rings.

1.3 Technical Overview

Our goal is to improve the concrete efficiency of PCGs for both VOLE and OLE over large prime fields. From the PCG literature, we observe that though two different flavors of LPN assumptions are primarily used (dual-LPN for VOLE vs. Ring-LPN/QA-SD for OLE), the core task is to efficiently compute the syndrome. This reduces to identifying a family of linear codes that has a fast encoding code as well as a good minimum distance for security concerns.

Observing that for large prime fields multiplication is significantly more expensive than addition, multiplication operations dominate the overall encoding time. In existing linear-time encodable codes, the encoding time is closely related to the expander degree, and unfortunately, decreasing the degree would substantially reduce the minimum distance and thus compromise the security. Consequently, such codes cannot achieve the best of both worlds.

We instead focus on optimizing PCGs based on Ring-LPN/QA-SD assumptions, where Fast Fourier Transform (FFT) is employed to compute the syndrome $as + e$ with a, s, e belonging to some polynomial ring $\mathcal{R}$.

Fast Fourier Transform. The Fast Fourier Transform proposed by Cooley and Tuckey [CT65] in 1965, provides an efficient algorithm for polynomial multiplication, which is widely used in modern cryptography. At its core, FFT is a divide-and-conquer method for fast polynomial evaluation and interpolation. FFT performs particularly well over some algebraic structures such as the cyclotomic ring $\mathbb{F}_p[X]/(X^N + 1)$ used in Ring-LWE [LPR13], where $p \equiv 1 \bmod 2N$ and $N = 2^n$ for $n \geq 1$. The efficiency comes from the symmetry of the evaluation points $\omega, \omega^3, \ldots, \omega^{2N-1}$, where ω is a primitive $2N$-th root of unity modulo p

such that $\omega^N \equiv -1 \bmod p$. FFT over cyclotomic rings has been extensively optimized in the past decades, making it the recommended approach in [BCG+20b]. Note that the polynomial ring $\mathbb{F}_p[X]/(X^{2N}-1)$ also permits a similarly efficient FFT by utilizing all $2N$ evaluation points (the complete set of $2N$-th roots of unity), since the identity $X^{2N}-1 = (X^N-1)(X^N+1)$ holds. This ring fits the framework of the QA-SD assumption by setting the number of variables n to 1, and thus serves as the primary structure for our implementation of Ring-LPN in [BCG+20b].

For the multiplication of multivariate polynomials, the theoretical FFT algorithm has been considered in [Obe07, BCCD23]. For a multivariate polynomial ring $\mathcal{R} = \mathbb{F}_p[\mathtt{X}_1,\ldots,\mathtt{X}_n]/(\mathtt{X}_1^d-1,\ldots,\mathtt{X}_n^d-1)$ where $d \mid p-1$, FFT algorithm requires $O(n \times d^n) = O(N \log N)$ multiplications, analogous to the univariate setting. For the special field $\mathbb{F}_4$, Bombar et al. [BBC+24] design and implement an efficient FFT algorithm. Instead of classically dividing polynomial evaluation and interpolation into 2 sub-problems, they divide it into 3 sub-problems. In addition, they pack multiple $\mathbb{F}_4$ elements within a single machine word in order to perform several FFTs in parallel.

Our approach. However, such FFT optimizations are limited to small fields. For large fields, we show a conceptually simple, yet highly efficient FFT algorithm.

Let $\mathbb{F}_p$ denote a large prime field. We will assume QA-SD assumption over a specific group ring $\mathbb{F}_p[\mathbb{Z}_2^n]$. In particular, one can simply view $\mathbb{F}_p[\mathbb{Z}_2^n]$ as the polynomial ring

$$\mathcal{R} \coloneqq \mathbb{F}_p[\mathtt{X}_1,\ldots,\mathtt{X}_n]/(\mathtt{X}_1^2-1,\ldots,\mathtt{X}_n^2-1).$$

For any prime p such that $2 \mid p-1$, it holds that $\mathtt{X}_i^2-1 \mid \mathtt{X}_i^{p-1}-1$ over $\mathbb{F}_p$ and $1,-1$ are two distinct roots of $\mathtt{X}_i^2-1$. By the Chinese Remainder Theorem (CRT), we conclude that the polynomial ring $\mathcal{R}$ is isomorphic to $\mathbb{F}_p^N$ with $N = 2^n$, for any prime $p > 2$.

Our key observation is that in this case, the evaluation points are $\{1,-1\}^n$, which forms a binary n-cube. To evaluate some $f(\mathtt{X}_1,\ldots,\mathtt{X}_n) \in \mathcal{R}$ at all points in $\{1,-1\}^n$, it is equivalent to compute

$$\widehat{\mathbf{f}} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^{\otimes n} \cdot \mathbf{f},$$

where $\mathbf{f} \in \mathbb{F}_p^N$ denotes the coefficients of f, and $\widehat{\mathbf{f}} \in \mathbb{F}_p^N$ denotes the evaluations. The above computation is precisely the well-known *Walsh-Hadamard transform*. In contrast to other transforms, Walsh-Hadamard transform can be computed with only $N \log N$ additions/subtractions without multiplications. This feature enables *extremely efficient* polynomial evaluation and interpolation compared to general FFT since multiplications are far more costly than additions/subtractions over large prime fields. Indeed, the Walsh-Hadamard transform can be viewed as a specific instance of a multidimensional FFT [Kun79].

Now, to compute a syndrome $a \cdot s + e$, for $a, s, e \in \mathcal{R}$, it suffices to first compute the Walsh-Hadamard transform $\widehat{\mathbf{a}}, \widehat{\mathbf{s}}, \widehat{\mathbf{e}}$ of a, s, e, and then compute

$$\widehat{\mathbf{a}} * \widehat{\mathbf{s}} + \widehat{\mathbf{e}},$$

where $*$ denotes component-wise multiplication. It is straightforward to see that this perfectly fits frameworks of PCG for VOLE and OLE, where polynomial interpolations are not necessarily required. In practice, for N VOLEs, each party only needs to perform N or $2N$ multiplications over $\mathbb{F}_p$ plus several $(2 \sim 4)$ Walsh-Hadamard transform calls. For N OLEs, it requires $2N$ multiplications over $\mathbb{F}_p$ plus 4 Walsh-Hadamard transform calls. Note that each Walsh-Hadamard transform requires only $N \log N$ additions/subtractions over $\mathbb{F}_p$.

Notably, to apply our technique, we only require that p is an odd prime. This means that our PCGs are *field-agnostic*. Particularly, this allows for efficient computation over a Mersenne prime field $\mathbb{F}_p = 2^k - 1$, which permits an efficient modular algorithm. In contrast, classical FFT does not directly work for such a field since $p - 1$ has only a single 2 factor.

2 Preliminaries

2.1 Basic Notations

We use $\mathbb{F}_p$ to denote a finite field with an odd size p and let $\mathbb{F}_p^* = \mathbb{F}_p \setminus \{0\}$. Given $\ell, r \in \mathbb{Z}$, we use $[\ell, r]$ to denite the index set $\{\ell, \ell+1, \cdots, r\}$, as well as $[\ell, r) = \{\ell, \ell+1, \cdots, r-1\}$ and $[r] = [1, r] = \{1, 2, \cdots, r\}$. Divisibility is denoted as $a \mid b$, representing that a divides b.We use $x \xleftarrow{\$} S$ to denote a uniformly random sample drawn from set S. Let λ denote the security parameter.

In this paper, we mainly consider a polynomial ring $\mathcal{R} = \mathbb{F}_p[\mathtt{X}_1, \cdots, \mathtt{X}_n]/(\mathtt{X}_1^2 - 1, \cdots, \mathtt{X}_n^2 - 1)$ with n variables $\{\mathtt{X}_1, \cdots, \mathtt{X}_n\}$. For a polynomial $f \in \mathcal{R}$, we use $\mathbf{f} \in \mathbb{F}_p^N$ to denote the coefficients of f and $\widehat{\mathbf{f}} \in \mathbb{F}_p^N$ to represent the evaluation of f over $\{-1, 1\}^n$, where $N = 2^n$. For a vector $\mathbf{e} \in \mathbb{F}_p^n$, we denote $\mathsf{wt}(\mathbf{e})$ as its Hamming weight. Furthermore, for a polynomial f with n variables, we use $\mathsf{wt}(f)$ to denote the Hamming weight of the corresponding coefficient vector $\mathsf{wt}(\mathbf{f})$. A polynomial $f \in \mathcal{R}$ with $\mathsf{wt}(f) = t$ is denoted as a t-sparse polynomial.

For two vectors $\mathbf{u} = (u_1, \cdots, u_n), \mathbf{v} = (v_1, \cdots, v_n) \in R^n$ for some ring R, we denote $\mathbf{u} \otimes \mathbf{v} \in R^{n^2}$ as their tensor product, i.e., $\mathbf{u} \otimes \mathbf{v} = (u_j \cdot v_j)_{i,j \in [n]} = (v_1 \cdot \mathbf{u}, \cdots, v_n \cdot \mathbf{u})$. Similarly, we write $\mathbf{u} \boxplus \mathbf{v} \in R^{n^2}$ to denote the outer sum of two vectors, i.e., $\mathbf{u} \boxplus \mathbf{v} = (u_i + v_j)_{i,j \in [n]} = (v_1 + \mathbf{u}, \cdots, v_n + \mathbf{u})$. We use $\mathbf{u} + \mathbf{v} \in R^n$ and $\mathbf{u} * \mathbf{v} \in R^n$ to denote the component-wise addition and multiplication of these two vectors, respectively. The symbol $\langle \mathbf{u}, \mathbf{v} \rangle \in R$ represents the inner product between them, which equals to $\sum_{i \in [n]} u_i \cdot v_i$.

2.2 Pseudorandom Correlation Generators

The notion of pseudorandom correlation generators PCG was first introduced in [BCG+19a, BCG+19b]. Informally, a PCG for some correlation allows for

distributing 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. Furthermore, *programmable* PCG enables the reuse of seed, and allows for extending 2-party correlations from PCG to multi-party correlations. We refer to Section A.1 in the Supplementary Material for the formal definition and properties of PCG.

2.3 Function Secret Sharing and Puncturable PRF

GI16bFunction 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 and I is an input domain, 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. We briefly introduce two special function secret sharing schemes, distributed point function and puncturable pseudorandom function, which are applied to PCG for OLE and VOLE, respectively.

Distributed Point Function. 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 2.1 (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 $\mathsf{SPFSS.FullEval}$ as evaluations of $\mathsf{SPFSS.Eval}$ 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. For regular multi-point distributions, the number of PRG calls is sheaved by a factor t. Notably, a recent work [BGH+24] presents improved FSS constructions for multi-point functions.

Puncturable Pseudorandom Function. A puncturable pseudorandom function (PPRF) is a pseudorandom function F (PRF) that, given an input x and a PRF key, can generate a punctured key $k\{x\}$ which can evaluate F at every point except x and does not conceal any information about x. PPRF was first introduced in [KPTZ13] and then serve as an improvement to replace DPF to design PCG for VOLE and OT [BCG+19a]. Formal definition of PPRF is referred to Section A.2 in the Supplementary Material.

2.4 Syndrome Decoding of Quasi-Abelian Codes

Group ring. Before describing quasi-abelian codes, we first introduce the notation of group rings. We defer the properties of group rings to Section A.3 in the Supplementary Material.

Definition 2.2 (Group Ring). *Let R be a ring, and $\mathbb{G}$ be a group. Define the set $R[\mathbb{G}] = \{\sum_{g\in\mathbb{G}} a_g g \mid a_g \in R,$ and there are finitely many $a_g \neq 0\}$. The two elements $\sum_{g\in\mathbb{G}} a_g g, \sum_{g\in\mathbb{G}} b_g g \in R[\mathbb{G}]$ are said to be equal if and only if $a_g = b_g$ for any $g \in \mathbb{G}$. Define addition and multiplication on $R[\mathbb{G}]$ as follows:*

$$(\sum_{g\in\mathbb{G}} a_g g) + (\sum_{g\in\mathbb{G}} b_g g) = \sum_{g\in\mathbb{G}} (a_g + b_g) g,$$
$$(\sum_{g\in\mathbb{G}} a_g g) \cdot (\sum_{g\in\mathbb{G}} b_g g) = \sum_{g\in\mathbb{G}} \big(\sum_{h\in\mathbb{G}} (a_h b_{h^{-1}g})\big) g.$$

$R[\mathbb{G}]$ with addition and multiplication defined as above is called a group ring.

Quasi-Abelian codes. Let $\mathbb{F}_p$ be a prime field and $\mathbb{G}$ a finite abelian group. Let ℓ be a positive integer, and consider the free $\mathbb{F}_p[\mathbb{G}]$-module of rank ℓ:

$$(\mathbb{F}_p[\mathbb{G}])^\ell := \mathbb{F}_p[\mathbb{G}] \oplus \cdots \oplus \mathbb{F}_p[\mathbb{G}] = \{(a_1, \ldots, a_\ell \mid a_i \in \mathbb{F}_p[\mathbb{G}]\}.$$

Any $\mathbb{F}_p[\mathbb{G}]$-submodule of $(\mathbb{F}_p[\mathbb{G}])^\ell$ is called a quasi-$\mathbb{G}$ code of index ℓ. If $\mathbb{G}$ is abelian, then a quasi-$\mathbb{G}$ code is quasi-abelian. Let $G \in (\mathbb{F}_p[\mathbb{G}])^{k\times\ell}$ be a generating matrix of rank $k \leq \ell$. Then the quasi-$\mathbb{G}$ code given by G is defined as follows:

$$\mathfrak{C} := \{\mathbf{m} \cdot G \mid \mathbf{m} = (m_1, \ldots, m_k) \in (\mathbb{F}_p[\mathbb{G}])^k\}$$

The quasi-$\mathbb{G}$ code $\mathfrak{C}$ of index ℓ can be seen as a linear code of length $|\mathbb{G}| \times \ell$ over $\mathbb{F}_p$, where the generating matrix is formed out of $k \times \ell$ square blocks of size $|\mathbb{G}|$ (induced by G). Assuming $|\mathbb{G}| = N$, fix an ordering $g_1, \ldots, g_N$ of elements in $\mathbb{G}$ and define the isomorphism $\varphi : \mathbb{F}_p[\mathbb{G}] \to \mathbb{F}_p^N$. Let $\mathsf{HW}_t(\mathbb{F}_p[\mathbb{G}])$ denote the set $\{a \in \mathbb{F}_p[\mathbb{G}] \mid \mathsf{wt}(\varphi(a)) = t\}$, from which the noise will be uniformly sampled. A random quasi-$\mathbb{G}$ code of index 2, in systematic form, has a parity-check matrix $H \in (\mathbb{F}_p[\mathbb{G}])^2$ of the form $H = (1 \mid a)$, where a is uniformly distributed in $\mathbb{F}_p[\mathbb{G}]$.

Syndrome decoding of quasi-abelian codes. As in [BCCD23, LXYY25a], we consider QA-SD problem over the group ring $\mathbb{F}_q[\mathbb{Z}_d^n]$, which is isomorphic to $\mathbb{F}_q[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \ldots, \mathtt{X}_n^d - 1) \simeq \mathbb{F}_q^N$, where $d \mid q-1$, and $N = d^n$.

Definition 2.3 (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 $a_i \xleftarrow{\$} \mathcal{R}$, $i \in [c-1]$. 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 2.4 (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:*

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

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

The above problems are referred to as search QA-SD$(c, \mathcal{R}, t)$ and decisional QA-SD$(c, \mathcal{R}, t)$, respectively. We say that the search (decisional) QA-SD$(c, \mathcal{R}, t)$ assumption holds when there exists no PPT algorithm that solves the problem with a non-negligible advantage. For simplicity, we set the compression factor $c = 2$ in our protocol, and we denote QA-SD$(\mathcal{R}, t)$ as QA-SD$(2, \mathcal{R}, t)$. In this work, we mainly focus on the QA-SD problem over a large prime field $\mathbb{F}_p$ and set $q = p$ and $d = 2$.

2.5 Ring Isomorphism

In this section, we briefly recall some properties of multivariate rings. The most important feature we used is a ring isomorphism between $\mathcal{R} = \mathbb{F}_q[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \ldots, \mathtt{X}_n^d - 1)$ and $\mathbb{F}_q^{d^n}$.

Lemma 2.5 (The Multivariate Ring Isomorphism). *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 a ring isomorphism*

$$\phi : \mathcal{R} \to \mathbb{F}_q^N \text{ with } N = d^n.$$

Proof. Since $d \mid q-1$, the polynomial $\mathtt{X}^d - 1$ splits completely into distinct linear factors over $\mathbb{F}_q$:

$$\mathtt{X}^d - 1 = \prod_{i=1}^{d} (\mathtt{X} - \alpha_i)$$

where $\{\alpha_i\}_{i\in[d]}$ are d distinct elements of $\mathbb{F}_q$. Define the evaluation set $S := \{(x_1, \ldots, x_n) : x_i \in \{\alpha_1, \ldots, \alpha_n\}\}$. For $f \in \mathcal{R}$, define

$$\phi(f) := \left(f(\mathbf{a})\right)_{\mathbf{a}\in S} \in \mathbb{F}_q^N.$$

It is straightforward to verify that ϕ is a well-defined ring homomorphism, bijection, and thus an isomorphism. □

3 Walsh-Hadamard Transform

The Walsh-Hadamard transform (WHT) is a linear transformation that maps a vector of $N = 2^n$ elements to another vector of N elements, typically over the reals or complex numbers [AR75, MM12, CS17]. It has applications in signal processing, coding theory, and cryptography. Conceptually, the WHT can be viewed as an n-dimensional discrete Fourier transform (DFT) of size 2^n. It is defined recursively via the Hadamard matrices $H_n \in \{1, -1\}^{2^n \times 2^n}$:

$$H_n := \begin{pmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{pmatrix}, H_1 := \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \text{ and } H_0 := 1.$$

Compared with the DFT, the Walsh-Hadamard transform has the advantage that it requires only additions and subtractions, whereas the DFT involves multiplications by complex roots of unity. This makes the WHT especially efficient for large input sizes.

In this section, we generalize the Walsh-Hadamard transform to the multivariate polynomial ring $\mathcal{R} := \mathbb{F}_p[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^2 - 1, \ldots, \mathtt{X}_n^2 - 1)$ and present an efficient algorithm to compute it.

Traditionally, the WHT is defined over $\mathbb{Z}_2^n$ with values in $\mathbb{C}$. For a function $f : \mathbb{Z}_2^n \to \mathbb{C}$, its Walsh-Hadamard transform $\hat{f} : \mathbb{Z}_2^n \to \mathbb{C}$ is defined as

$$\hat{f}(\mathbf{x}) := \sum_{\mathbf{y}\in\mathbb{Z}_2^n} f(\mathbf{y})(-1)^{\langle \mathbf{x},\mathbf{y}\rangle}$$

where $\langle \mathbf{x}, \mathbf{y}\rangle = \sum_{i=1}^{n} x_i y_i$ is the inner product of $\mathbf{x}$ and $\mathbf{y}$.

Definition 3.1 (WHT over Multivariate Polynomial Ring). *For a polynomial $f \in \mathcal{R}$, the Walsh-Hadamard transform of f is defined as its evaluation over all points in $\{\pm 1\}^n$, i.e.,*

$$\{f(a_1, \ldots, a_n) : a_i \in \{-1, 1\}\}.$$

For input length $N = 2^n$, a direct computation of the WHT requires $O(N^2)$ operations. However, there exists a fast algorithm to compute the WHT with exactly $N \log N$ additions or subtractions. The traditional WHT algorithm is a divide-and-conquer algorithm that recursively breaks down a WHT of size N into two smaller WHTs of size $N/2$. The WHT algorithm follows the recursive definition of the Hadamard matrix H_n. Below, we present an in-place algorithm adapted to the polynomial ring $\mathcal{R}$.

Procedure 1: In-place Walsh-Hadamard Transform

WHT(f): On input a polynomial $f \in \mathcal{R} = \mathbb{F}_p[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^2 - 1, \ldots, \mathtt{X}_n^2 - 1)$, let $\mathbf{f} = (f_0, \ldots, f_{N-1}) \in \mathbb{F}_p^N$ denote the coefficients of f where $N = 2^n$.

For $\mathsf{step} = 1, 2, 4, 8 \ldots N/2$:
  For $i = 0, 2 \cdot \mathsf{step}, 4 \cdot \mathsf{step}, 6 \cdot \mathsf{step}, \ldots, N - 2 \cdot \mathsf{step}$:
    For $j = 0, 1, 2, \ldots, \mathsf{step} - 1$:
      1. Let $u := f_{i+j}$ and $v := f_{i+j+\mathsf{step}}$.
      2. Set $f_{i+j} := u + v$ and $f_{i+j+\mathsf{step}} := u - v$.

Output: The Walsh-Hadamard transform of f as $(f_0 \ldots f_{N-1}) \in \mathbb{F}_p^N$.

Theorem 3.2. *Procedure WHT performs exactly $N \log N$ additions or subtractions over $\mathbb{F}_p$.*

Proof. The outer loop runs $\log N$ iterations. In each outer iteration, the middle loop runs for $N/(2 \cdot \mathsf{step})$ iterations and the inner loop runs for step iterations. In each iteration of the inner loop, there are two additions or subtractions. Thus the total number of additions or subtractions is

$$\sum_{\mathsf{step}=1}^{N/2} \frac{N}{2 \cdot \mathsf{step}} \cdot \mathsf{step} \cdot 2 = N \sum_{\mathsf{step}=1}^{N/2} 1 = N \cdot \log N.$$

□

Note that the Walsh-Hadamard transform can be viewed as the FFT algorithm for the evaluation set $\{(\pm 1, \ldots, \pm 1)\}$. The computation complexity of FFT is dominated by the number of multiplications in $\mathbb{F}_p$. For Walsh-Hadamard transform, only additions and subtractions are used and thus the computation complexity is completely determined by the number of additions or subtractions in $\mathbb{F}_p$. This advantage will be amplified if the field size is very large. In this sense, our new QA-SD assumption leads to an improvement of PCG constructions.

Remark 3.3. In the standard WHT, computing the inverse WHT requires $O(N)$ multiplications over $\mathbb{F}_p$. However, in our PCG applications for VOLE and OLE, the inverse transform is unnecessary since the goal of PCG is to output multiplication triples over $\mathbb{F}_p$ rather than over $\mathcal{R}$. Thus the $O(N)$ multiplications over $\mathbb{F}_p$ are saved.

Remark 3.4. Recently, Alman and Rao [AR23] showed that the WHT can be computed with only $\frac{23}{24} N \log N + O(N)$ additions or subtractions, improving the classical $N \log N$ bound by a constant factor. For $p = O(1)$, [Alm23] further proposed an algorithm achieving $O(N \log N / \log\log N)$ bit operations via lookup tables. While asymptotically faster, these improvements have a limited impact in practice.

4 Minimum Distance of Quasi-Abelian Codes

Let $\mathbb{G}$ be a finite abelian group of order N and $\mathbb{F}_p$ be a finite field of size p. We consider quasi-abelian codes as defined in Section 2.4 over $\mathbb{F}_p$.

Definition 4.1 (Group Algebra). *Let $\mathbb{F}_p[\mathbb{G}]$ be a $\mathbb{F}_p$-vector space with basis $\mathbb{G}$, i.e., $\mathbb{F}_p[\mathbb{G}] = \{\sum_{x\in\mathbb{G}} a_x x : a_x \in \mathbb{F}_p\}$ and equipped with a multiplication induced by the group multiplication of $\mathbb{G}$, i.e., the multiplication of $\mathbb{F}_p[\mathbb{G}]$ is defined as*

$$\left(\sum_{x\in\mathbb{G}} a_x x\right) \cdot \left(\sum_{y\in\mathbb{G}} b_y y\right) \coloneqq \sum_{z\in\mathbb{G}} \left(\sum_{xy=z} a_x b_y\right) z.$$

Then $\mathbb{F}_p[\mathbb{G}]$ is called a group algebra *of $\mathbb{G}$ over $\mathbb{F}_p$.*

We introduce two main theorems from representation theory of finite groups [Wei03].

Lemma 4.2 (Maschke's theorem [Wei03, Theorem 1.14]). *If $\gcd(N, p) = 1$, then $\mathbb{F}_p[\mathbb{G}]$ is* semisimple.

From Maschke's theorem, the group algebra $\mathbb{F}_p[\mathbb{G}]$ is semisimple.

Lemma 4.3 (Wedderburn's theorem [Wei03, Proposition 3.18]). *$\mathbb{F}_p[\mathbb{G}]$ decomposes as a direct sum of simple ideals as follows*

$$\mathbb{F}_p[\mathbb{G}] = \mathbb{F}_p[\mathbb{G}]e_0 \oplus \mathbb{F}_p[\mathbb{G}]e_1 \oplus \cdots \oplus \mathbb{F}_p[\mathbb{G}]e_m,$$

where $e_0 = \frac{1}{n}\sum_{x\in\mathbb{G}} x$ and $e_1, \ldots, e_m$ are the primitive central idempotents.

Each $\mathbb{F}_p[\mathbb{G}]e_i$ is therefore a simple ideal of $\mathbb{F}_p[\mathbb{G}]$ and hence a group code. From Lemma 2.5, the ring $\mathbb{F}_p[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \ldots, \mathtt{X}_n^d - 1)$ is isomorphic to $\mathbb{F}_p^N$ where $d \mid (p-1)$ and $N = d^n$. Then for $\mathbb{G} \coloneqq \mathbb{Z}_d^n$, each $\mathbb{F}_p[\mathbb{G}]e_i$ has dimension exactly 1. Moreover, Wedderburn's theorem implies that every ideal I of $\mathbb{F}_p[\mathbb{G}]$ is a direct sum of some of the simple ideals $\mathbb{F}_p[\mathbb{G}]e_i$. Therefore, the ideals of dimension ℓ are precisely the direct sums of ℓ distinct simple ideals, and the number of dimension ℓ ideals is $\binom{N}{\ell}$.

We introduce some basic facts about group codes of $\mathbb{F}_p[\mathbb{G}]$.

Definition 4.4 (Group Code). *Any left ideal I of $\mathbb{F}_p[\mathbb{G}]$ denoted by $I \leq \mathbb{F}_p[\mathbb{G}]$ is called a* group code *as well as a $\mathbb{F}_p[\mathbb{G}]$ code. If $\mathbb{G}$ is abelian, then left ideal, right ideal and two-sided ideal are identical and thus I is called an abelian code.*

Given an element $a \in \mathbb{F}_p[\mathbb{G}]$, the group code generated by a can be defined as $\mathbb{F}_p[\mathbb{G}]a \coloneqq \{b \cdot a : b \in \mathbb{F}_p[\mathbb{G}]\}$. The weight of $a = \sum_{x \in \mathbb{G}} a_x x \in \mathbb{F}_p[\mathbb{G}]$ is the number of non-zero coefficients of a, i.e., $\mathsf{wt}(a) \coloneqq \sum_{a_x \neq 0} 1$. Thus, the minimum distance of $I \leq \mathbb{F}_p[\mathbb{G}]$ is defined as the minimum weight of nonzero elements in I.

Given a matrix $\mathbf{A} \in \mathbb{F}_p[\mathbb{G}]^{k \times c}$, the quasi-abelian code of index c is defined as

$$\mathcal{C}_{\mathbf{A}} \coloneqq \left\{ \mathbf{b} \cdot \mathbf{A} : \mathbf{b} \in \mathbb{F}_p[\mathbb{G}]^k \right\}.$$

Note that when $|\mathbb{G}| = 1$, a random QA code $\mathcal{C}_{\mathbf{A}}$ is nothing else than a random linear code over $\mathbb{F}_p$. Though we prove the general case, we are particularly interested in the case that $k = 1$, e.g., rank-1 QA code of index c.

To bound the minimum distance of $\mathcal{C}_{\mathbf{A}}$, for $\mathbf{b} \in \mathbb{F}_p[\mathbb{G}]^k$, define the indicator random variable $X_{\mathbf{b}}$ as

$$X_{\mathbf{b}} \coloneqq \begin{cases} 1, & \text{if } 0 < \mathsf{wt}(\mathbf{b} \cdot \mathbf{A}) \leq cN\delta \\ 0, & \text{otherwise.} \end{cases}$$

Let $X \coloneqq \sum_{\mathbf{b} \in \mathbb{F}_p[\mathbb{G}]^k \setminus \{0\}} X_{\mathbf{b}}$. To prove that $\Pr\left(\Delta(\mathcal{C}_{\mathbf{A}}) \leq \delta\right) = \Pr(X \geq 1)$ is bounded, it suffices to prove that $E[X]$ is bounded from Markov inequality.

Let $T_\ell \coloneqq \{I \leq \mathbb{F}_p[\mathbb{G}] : \dim(I) = \ell\}$ be the set of dimension ℓ ideals of $\mathbb{F}_p[\mathbb{G}]$ and $I^* \coloneqq \{b \in I : \mathbb{F}_p[\mathbb{G}]b = I\} \subseteq I$ be the set of generators of I. Then $\mathbb{F}_p[\mathbb{G}]$ can be viewed as a union of ideals as:

$$\mathbb{F}_p[\mathbb{G}] = \{0\} \bigcup \Big(\bigcup_{\ell \in [1,N]} \big(\bigcup_{I \in T_\ell} I^* \big) \Big).$$

For $\mathbf{b} = (\mathbf{b}_1 \dots \mathbf{b}_t) \in \mathbb{F}_p[\mathbb{G}]^k$, let $I_{\mathbf{b}} \coloneqq \mathbb{F}_p[\mathbb{G}]\mathbf{b}_1 + \dots \mathbb{F}_p[\mathbb{G}]\mathbf{b}_t$ be the ideal generated by $\mathbf{b}_1 \dots \mathbf{b}_t$ and $d_{\mathbf{b}} \coloneqq \dim(I_{\mathbf{b}})$ the dimension of $I_{\mathbf{b}}$. Given an ideal I of $\mathbb{F}_p[\mathbb{G}]$, let $d_I \coloneqq \dim(I)$. For the k-fold direct product I^k of I, it holds that $\dim(I^k) = k \cdot d_I$. Define $I^{k*} \coloneqq \{\mathbf{b} \in I^k : I_{\mathbf{b}} = I\}$. Then I^{k*} consists of k-tuples that exactly generate I. Similarly, by the definition of I^{k*}, $\mathbb{F}_p[\mathbb{G}]^k$ can be viewed as a union of ideals:

$$\mathbb{F}_p[\mathbb{G}]^k = \{0\} \bigcup \Big(\bigcup_{0 \neq I \leq \mathbb{F}_p[\mathbb{G}]} I^{k*} \Big) \tag{1}$$

Given $\mathbf{b} \in \mathbb{F}_p[\mathbb{G}]^k$, Fan and Lin [FL22, Lemma 5.5(1)] showed that

$$E\left[X_{\mathbf{b}}\right] \leq p^{-c \cdot d_{\mathbf{b}}(1 - H_p(\delta))}. \tag{2}$$

We now present a finer-grained analysis of $E[X]$ via Wedderburn's theorem.

Theorem 4.5. *For QA code with generator matrix $\mathbf{A} \xleftarrow{\$} \mathbb{F}_p[\mathbb{G}]^{k \times c}$, if $H_p(\delta) + \frac{k + \log_p N}{c} < 1$, then it holds that*

$$E\left[X\right] \lessapprox N \cdot p^{(1 - c(1 - H_p(\delta)))} = p^{1 + \log_p N - c(1 - H_p(\delta))}.$$

Proof. By the linearity of expectations, we have

$$E[X] = \sum_{\mathbf{b}\in\mathbb{F}_p[\mathbb{G}]^k\setminus\{\mathbf{0}\}} E\left[X_{\mathbf{b}}\right].$$

From Eq.(2), each $E[X_{\mathbf{b}}]$ is closely related to $d_{\mathbf{b}}$. From Eq.(1), the summation can be changed over the ideals of $\mathbb{F}_p[\mathbb{G}]^k$ instead of over $\mathbb{F}_p[\mathbb{G}]^k$. Then,

$$\begin{aligned} E[X] &= \sum_{\mathbf{b}\in\mathbb{F}_p[\mathbb{G}]^k\setminus\{0\}} E\left[X_{\mathbf{b}}\right] = \sum_{0\neq I\leq\mathbb{F}_p[\mathbb{G}]} \sum_{\mathbf{b}\in I^{k*}} E[X_{\mathbf{b}}] \\ &= \sum_{\ell\in[1,N]} \sum_{I\in T_\ell} \sum_{\mathbf{b}\in I^{k*}} E[X_{\mathbf{b}}] \leq \sum_{\ell\in[1,N]} \sum_{I\in T_\ell} \left|I^{k*}\right| \cdot p^{-c\ell(1-H_p(\delta))} \\ &\leq \sum_{\ell\in[1,N]} \sum_{I\in T_\ell} p^{k\ell-c\ell(1-H_p(\delta))}. \end{aligned}$$

From Wedderburn's theorem, the number of dimension ℓ ideals is $\binom{N}{\ell}$. Then

$$E[X] \leq \sum_{\ell\in[1,N]} \binom{N}{\ell} \cdot p^{k\ell-c\ell(1-H_p(\delta))} = \left(1+p^{k-c(1-H_p(\delta))}\right)^N - 1.$$

For $N\cdot p^{(k-c(1-H_p(\delta)))} \ll 1$, from Taylor expansion, it holds that $\left(1+p^{k-c(1-H_p(\delta))}\right)^N \lessapprox 1+N\cdot p^{(k-c(1-H_p(\delta)))}$. Thus,

$$E[X] \leq \left(1+p^{k-c(1-H_p(\delta))}\right)^N - 1 \lessapprox N\cdot p^{(k-c(1-H_p(\delta)))} = p^{\log_p N+k-c(1-H_p(\delta))}.$$

□

From Markov inequality, we have:

Corollary 4.6. *For $H_p(\delta)+(\log_p N+k)/c \leq 1$, it holds that*

$$\Pr_{\mathbf{A}\xleftarrow{\$}\mathbb{F}_p[\mathbb{G}]^{k\times c}}[\Delta(\mathcal{C}_{\mathbf{A}})\leq\delta] \leq p^{\log_p N+k-c(1-H_p(\delta))}.$$

In particular, for $k=1$, it becomes rank-1 QA code.

Corollary 4.7. *For $H_p(\delta)+(\log_p N+1)/c < 1$, it holds that*

$$\Pr_{\mathbf{a}\xleftarrow{\$}\mathbb{F}_p[\mathbb{G}]^{c}}[\Delta(\mathcal{C}_{\mathbf{a}})\leq\delta] \leq p^{\log_p N+1-c(1-H_p(\delta))}.$$

5 PCG for VOLE over Large Prime Fields

In this section, we present an efficient PCG construction for VOLE over large prime fields. A VOLE is a degree-1 PCG correlation to output correlations $(\mathbf{x}_1,\Delta)\in\mathbb{F}_p^N\times\mathbb{F}_p$ and $(\mathbf{x}_2,\mathbf{v})\in\mathbb{F}_p^N\times\mathbb{F}_p^N$ such that $\mathbf{x}_1+\mathbf{x}_2=\Delta\mathbf{v}$. Boyle et

al. [BCG+19a] proposed the PCG construction for VOLE based on LPN or syndrome decoding assumption. Generating VOLE from the syndrome decoding assumption needs to compute $H\mathbf{v}$ and $H\mathbf{e}$, where H is a public random matrix, $\mathbf{v}$ is a secret pseudorandom vector, and $\mathbf{e}$ is a secret t-sparse vector. If H does not have any structure, such computation takes time $\Omega(N^2)$, which seems less efficient. Thus, there are some efforts to design more structured H while still maintaining psudorandomness of $H\mathbf{e}$. The state-of-the-art construction takes $O(N)$ time to compute $H\mathbf{v}$ [BCG+22, RRT23] via a linear time encodable code. Although their security analysis is made for a binary field, it can be naturally generalized to a larger field. However, moving to a larger field increases the computation complexity. For example, the operations of multiplications and additions over large field are not equally efficient. In view of this, we propose an efficient PCG construction for VOLE based on the QA-SD assumption and Walsh-Hadamard transform. We also use the puncturable pseudorandom function (PPRF) to share the locations and values, which is more efficient than distributed point function.

Construction 2: $\mathsf{Cons}_{\mathsf{VOLE}}^{\mathbb{F}_p}$

PARAMETER: Security parameter λ, Hamming weight $t = t(\lambda)$, $N = 2^n$, $\mathcal{R} = \mathbb{F}_p[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^2 - 1, \ldots, \mathtt{X}_n^2 - 1)$. A PPRF scheme $(\mathsf{PPRF.Gen}, \mathsf{PPRF.Puncture}, \mathsf{PPRF.Eval})$ for sums of t point functions, with domain $[0, N)$ [a] and range $\mathbb{F}_p$.
PUBLIC INPUT: A uniformly random $a \in \mathcal{R}$ and its Walsh-Hadamard transform $\widehat{\mathbf{a}} = \mathbf{WHT}(a)$.
CORRELATION: After expansion, outputs $(\mathbf{x}, \mathbf{z}_0) \in \mathbb{F}_p^{2N}$ and $(\Delta, \mathbf{z}_1) \in \mathbb{F}_p^N \times \mathbb{F}_p$ such that $\Delta \cdot \mathbf{x} = \mathbf{z}_0 + \mathbf{z}_1$.
Gen: On input 1^λ:

1. For $\sigma = 0$, sample random vectors $\mathbf{A} = (a_1, \ldots, a_t), \mathbf{B} = (b_1, \ldots, b_t) \leftarrow [0, N)^t$ and $\mathbf{s} = (s_1, \ldots, s_t), \mathbf{e} = (e_1, \ldots, e_t) \leftarrow \mathbb{F}_p^t$
2. For $\sigma = 1$, sample $\Delta \in \mathbb{F}_p$.
3. Sample two PPRF keys $\mathbf{k}_s, \mathbf{k}_e \leftarrow \mathsf{PPRF.Gen}(1^\lambda)$, and compute two punctured keys $\mathbf{k}_s^{\mathbf{A}} \leftarrow \mathsf{PPRF.Puncture}(\mathbf{k}_s, \mathbf{A})$ and $\mathbf{k}_e^{\mathbf{B}} \leftarrow \mathsf{PPRF.Puncture}(\mathbf{k}_e, \mathbf{B})$.
4. For $i \in [t]$, let $s'_i = \Delta \cdot s_i - \mathsf{PPRF.Eval}(\mathbf{k}_s, a_i)$ and $e'_i = \Delta \cdot e_i - \mathsf{PPRF.Eval}(\mathbf{k}_e, b_i)$
5. Let $\mathbf{k}_0 = \Big((\mathbf{k}_s^{\mathbf{A}}, \mathbf{A}, (s'_i)_{i\in[t]}), (\mathbf{k}_e^{\mathbf{B}}, \mathbf{B}, (e'_i)_{i\in[t]})\Big)$ and $\mathbf{k}_1 = (\mathbf{k}_s, \mathbf{k}_e, \Delta)$.
6. Output $(\mathbf{k}_0, \mathbf{k}_1)$.

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

1. For $\sigma = 0$, parse $\mathbf{k}_0$ as $\Big((\mathbf{k}_s^{\mathbf{A}}, \mathbf{A}, (s'_i)_{i\in[t]}), (\mathbf{k}_e^{\mathbf{B}}, \mathbf{B}, (e'_i)_{i\in[t]})\Big)$.
   - For $j \in [N]$, define $\mathbf{v} \in \mathbb{F}_p^N$ as $\mathbf{v}[j] = s'_i$ if $j \in \mathbf{A}$ and $\mathbf{v}[j] = -\mathsf{PPRF.Eval}(\mathbf{k}_s^{\mathbf{A}}, j)$ otherwise, and define $\mathbf{u} \in \mathbb{F}_p^N$ as $\mathbf{u}[j] = e'_i$ if $j \in \mathbf{B}$ and $\mathbf{u}[j] = -\mathsf{PPRF.Eval}(\mathbf{k}_e^{\mathbf{B}}, j)$ otherwise.

– Define elements of $\mathcal{R}$

$$s = \sum_{\ell=1}^{t} s[\ell] \cdot A[\ell], \quad e = \sum_{\ell=1}^{t} e[\ell] \cdot B[\ell]$$

Interpret the coefficient vectors $\mathbf{u}$ and $\mathbf{v}$ as the correspond polynomials u and v, respectively.
– Compute $\widehat{\mathbf{v}} = \mathbf{WHT}(v)$, $\widehat{\mathbf{u}} = \mathbf{WHT}(u)$, and $\widehat{\mathbf{s}} = \mathbf{WHT}(s)$, $\widehat{\mathbf{e}} = \mathbf{WHT}(e)$.
– Let $\mathbf{x} = \widehat{\mathbf{a}} * \widehat{\mathbf{s}} + \widehat{\mathbf{e}}$ and $\mathbf{z}_0 = \widehat{\mathbf{a}} * \widehat{\mathbf{v}} + \widehat{\mathbf{u}}$
– Output $(\mathbf{x}, \mathbf{z}_0)$.

2. For $\sigma = 1$, parse $\mathbf{k}_1$ as $(\mathbf{k}_s, \mathbf{k}_e, \Delta)$.
– Compute $\mathbf{w}_0 \leftarrow \mathsf{PPRF.FullEval}(\mathbf{k}_s)$ and $\mathbf{w}_1 \leftarrow \mathsf{PPRF.FullEval}(\mathbf{k}_e)$ in $\mathbb{F}_p^N$, and interpret them as the corresponding polynomials w_0 and w_1, respectively.
– Compute $\widehat{\mathbf{w}}_0 = \mathbf{WHT}(w_0)$, $\widehat{\mathbf{w}}_1 = \mathbf{WHT}(w_1)$ and let $\mathbf{z}_1 = \widehat{\mathbf{a}} * \widehat{\mathbf{w}}_0 + \widehat{\mathbf{w}}_1$.
– Output $(\Delta, \mathbf{z}_1)$.

[a] Each j corresponds to a basis $\prod_{i\in[n]} \mathtt{x}_i^{j_i}$ of $\mathcal{R}$, where $j_i \in \{0,1\}$.

Theorem 5.1. *Assume a secure* PPRF *scheme for sums of point functions and QA-SD$(\mathcal{R}, t)$ is hard. Then there exists a PCG construction* $\mathsf{Cons}^{\mathbb{F}_p}_{\mathsf{VOLE}}$ *that generates VOLE correlations over $\mathbb{F}_p$. If the* PPRF *is based on a* $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ *via the* PRG*-based construction from [BCG+19a], for generating $N := 2^n$ VOLE correlations over $\mathbb{F}_p$, we have that:*

– *For party with $\sigma = 0$, the seed size is at most: $2t(\lambda \log \frac{N}{t} + \log N + \log q)$ bits.*
– *For party with $\sigma = 1$, the seed size is at most: $2\lambda + \log q$ bits.*
– *For party with $\sigma = 0$, the computation of* ***Expand*** *can be done with at most $(4 + \lfloor \frac{\log p}{\lambda} \rfloor)N$* PRG *operations and $2N$ multiplications and $2N + 4N \log N$ additions in $\mathbb{F}_p$*
– *For party with $\sigma = 1$, the computation of* ***Expand*** *can be done with at most $(4 + \lfloor \frac{\log p}{\lambda} \rfloor)N$* PRG *operations and N multiplications and $N + 2N \log N$ additions in $\mathbb{F}_p$.*

Proof. We defer the proof to Section B.1 in the Supplementary Material. □

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

Proof. To show that $\mathsf{Cons}^{\mathbb{F}_p}_{\mathsf{VOLE}}$ is programmable, we can re-define **Gen** by letting it take additional inputs (ρ_0, ρ_1), where $\rho_0 = (\mathbf{A}, \mathbf{B}, \mathbf{s}, \mathbf{e})$ and $\rho_1 = \Delta$. Party P_0 can locally compute $\mathbf{x} \in \mathbb{F}_p^N$ from ρ_0 and Party P_1 can locally compute Δ. 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 OLE over Large Prime Fields

In this section, we provide a new PCG construction for OLE over large prime fields. We note that the previous construction [BCG+20b] leverages the evaluation of univariate polynomial $f(\mathtt{X})$ at the elements in a subset $S \subseteq \mathbb{F}_p$ to generate $|S|$ OLE instances. To accelerate this evaluation, one needs to employ the FFT for polynomial over finite field. This means that the set S must be a multiplicative group of size $N := 2^r \mid p-1$ for some integer r. If this is the case, this evaluation requires $O(N \log N)$ number of $\mathbb{F}_p$ multiplications. Otherwise, the evaluation requires $O(N \log^2 N)$ number of $\mathbb{F}_p$ multiplications. We note that when p is a large prime, the multiplication in $\mathbb{F}_p$ is slower than the addition in $\mathbb{F}_p$. To make a comparison, we first present the construction in [BCG+20b], which is based on the Ring-LPN assumption.

Suppose the goal is to generate N random OLE instances over $\mathbb{F}_p$. Assume that $N \mid p-1$ and thus $\mathtt{X}^N - 1$ splits completely over $\mathbb{F}_p$, i.e., $\mathtt{X}^N - 1 = \prod_{i=1}^{N}(\mathtt{X} - \alpha^i)$ for some $\alpha \in \mathbb{F}_p$. Let $\mathcal{R} = \mathbb{F}_q[\mathtt{X}]/(\mathtt{X}^N - 1)$. Then, the framework relies on the ring isomorphism $\mathcal{R} \simeq \mathbb{F}_p^N$, which is implied by the Chinese Remainder Theorem. 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 assumption implies that $x_\sigma := a \cdot s_\sigma + e_\sigma$ is pseudorandom, for each $\sigma \in \{0,1\}$. The key observation is the following:

$$x_0 \cdot x_1 = (a \cdot s_0 + e_0) \cdot (a \cdot s_1 + e_1) = a^2 \cdot (s_0 s_1) + a \cdot (s_0 e_1 + s_1 e_0) + e_0 e_1. \quad (3)$$

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 $\mathcal{R} \simeq \mathbb{F}_p^N$, additive sharings of $x_0 \cdot x_1$ over $\mathcal{R}$ are equivalent to N OLE correlations over $\mathbb{F}_p$, 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 by using SPFSS. To convert the polynomials $x_\sigma \in \mathcal{R}$ into a vector $\in \mathbb{F}_p^N$ efficiently, one needs to apply the FFT over finite field. Such a transform requires at least $N \log N$ multiplications and $N \log N$ additions in $\mathbb{F}_p$. Since p is a large prime, the computational cost is a dominant part in this construction.

To mitigate this computational cost, we propose a new construction with a much faster evaluation algorithm. Suppose the goal is to generate $N = 2^n$ random OLE instances over $\mathbb{F}_p$ where p is an odd prime. Let $\mathcal{R} = \mathbb{F}_q[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^2 - 1, \ldots, \mathtt{X}_n^2 - 1)$. It is clear that $\mathtt{X}_i^2 - 1 = (\mathtt{X}_i - 1)(\mathtt{X}_i + 1)$ as p is an odd prime. Let $a \xleftarrow{\$} \mathcal{R}$ be the public input, and s_σ, e_σ be t-sparse elements of $\mathcal{R}$, where $\sigma \in \{0,1\}$. We will prove it later that $x_\sigma := a \cdot s_\sigma + e_\sigma$ is pseudorandom, for each $\sigma \in \{0,1\}$. We now distribute the 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 by using SPFSS. This can be done similarly to [BCG+20b]. However, when we want to evaluate x_σ, the evaluation can be done very efficiently as $\mathtt{X}_i$ only takes 1 and -1. Instead of using FFT over prime fields, we can mimic the Walsh-Hadamard transform for Boolean functions to accelerate the evaluation. We present our PCG construction in $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$ and defer its setup to Section C in the Supplementary Material.

Construction 3: $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$

PARAMETER: Security parameter λ, Hamming weight $t = t(\lambda)$, $N = 2^n$, $\mathcal{R} = \mathbb{F}_p[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^2 - 1, \ldots, \mathtt{X}_n^2 - 1)$. An FSS scheme $(\mathsf{SPFSS.Gen}, \mathsf{SPFSS.FullEval})$ for sums of t^2 point functions, with domain $[0, N)$[a] and range $\mathbb{F}_p$.
PUBLIC INPUT: A uniformly random $a \in \mathcal{R}$ and its Walsh-Hadamard transform $\widehat{\mathbf{a}} = \mathbf{WHT}(a)$, $\widehat{\mathbf{a}} * \widehat{\mathbf{a}}$.
CORRELATION: After expansion, outputs $(\mathbf{x}_0, \mathbf{z}_0) \in \mathbb{F}_p^{2N}$ and $(\mathbf{x}_1, \mathbf{z}_1) \in \mathbb{F}_p^{2N}$ such that $\mathbf{x}_0 * \mathbf{x}_1 = \mathbf{z}_0 + \mathbf{z}_1$.
Gen: On input 1^λ:

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

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

1. Parse $\mathbf{k}_\sigma$ as $\Big((K_\sigma^i)_{i\in[0,3]}, (\mathbf{A}_\sigma^0, \mathbf{s}_\sigma), (\mathbf{A}_\sigma^1, \mathbf{e}_\sigma)\Big)$.
2. Define elements of $\mathcal{R}$
$$s_\sigma = \sum_{\ell=1}^{t} \mathbf{s}_\sigma[\ell] \cdot \mathbf{A}_\sigma^0[\ell], \quad e_\sigma = \sum_{\ell=1}^{t} \mathbf{e}_\sigma[\ell] \cdot \mathbf{A}_\sigma^1[\ell].$$
3. For $i \in [0,3]$, compute $u_{\sigma,i} \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^i)$,viewed as $\mathcal{R}$ elements.
4. Compute $\widehat{\mathbf{u}}_{\sigma,i} = \mathbf{WHT}(u_{\sigma,i})$ for $i \in [0,3]$, $\widehat{\mathbf{s}}_\sigma = \mathbf{WHT}(s_\sigma)$ and $\widehat{\mathbf{e}}_\sigma = \mathbf{WHT}(e_\sigma)$.
5. Compute[b]
$$\mathbf{z}_\sigma = \widehat{\mathbf{a}} * \widehat{\mathbf{a}} * \widehat{\mathbf{u}}_{\sigma,0} + \widehat{\mathbf{a}} * (\widehat{\mathbf{u}}_{\sigma,1} + \widehat{\mathbf{u}}_{\sigma,2}) + \widehat{\mathbf{u}}_{\sigma,3}, \quad \mathbf{x}_\sigma = \widehat{\mathbf{a}} * \widehat{\mathbf{s}}_\sigma + \widehat{\mathbf{e}}_\sigma$$
6. Output $(\mathbf{x}_\sigma, \mathbf{z}_\sigma)$.

[a] Each j corresponds to a basis $\prod_{i\in[n]} \mathtt{X}_i^{j_i}$ of $\mathcal{R}$, where $j_i \in \{0,1\}$. The addition on $[0,N)$ is actually the addition on $\mathbb{G} = \mathbb{Z}_2^n$.
[b] The Walsh-Hadamard transform does not incur any multiplications in $\mathbb{F}_p$. The multiplication is only required in the component-wise product of two vectors. Thus, we only need $3N$ multiplications and $4N + 6N \log N$ additions in $\mathbb{F}_p$ to compute $(\mathbf{x}_\sigma, \mathbf{z}_\sigma)$.

Theorem 6.1. *Assume a secure* FSS *scheme* SPFSS *for sums of point functions and QA-SD($\mathcal{R}, t$) 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* $N := 2^n$ *OLE correlations over* $\mathbb{F}_p$*, we have that:*

- *Each party's seed has a maximum size around:* $4t^2\big(\log \frac{\lambda}{t}(\lambda+2)+\lambda+\log p\big)+2t(\log N+\log p)$ *bits.*
- *The computation of* ***Expand*** *can be done with at most* $(2+\lfloor\frac{\log p}{\lambda}\rfloor)4Nt$[6] PRG *operations and* $3N$ *multiplications and* $4N+6N\log N$ *additions in* $\mathbb{F}_p$*.*

Proof. We defer the proof to Section B.2 in the Supplementary Material. □

Theorem 6.2. *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 $\rho_\sigma = (\mathbf{A}_\sigma^0, \mathbf{A}_\sigma^1, \mathbf{s}_\sigma, \mathbf{e}_\sigma)$, for $\sigma \in \{0,1\}$. Note that ρ_σ corresponds to two t-sparse elements s_σ, e_σ of $\mathcal{R}$ as well as an element $x_\sigma := a \cdot s_\sigma + e_\sigma \in \mathcal{R}$. Party P_σ can locally compute $x_\sigma \in \mathbb{F}_p^N$ from ρ_σ. This essentially proves the programmability. Correctness follows that of Theorem 6.1. The programmable security can be proved in a very similar way to that of Theorem 6.1.

7 Efficiency Analysis

7.1 Parameter Selection

For QA-SD assumption, we provide both aggressive and conservative parameters, and list concrete parameters in Table 3. For completeness, we extend to the QA-SSD assumption and display the corresponding parameters in Table 8 in the Supplementary Material. We briefly introduce how to obtain aggressive and conservative parameters as follows.

Aggressive parameters. In [BBC+24], Bombar et al. provide a SageMath script [Bom24] to estimate the security level of the QA-SD problem for $\mathcal{R} = \mathbb{F}_p[\mathtt{X}_1, \cdots, \mathtt{X}_n]/(\mathtt{X}_1^{p-1}-1, \cdots, \mathtt{X}_n^{p-1}-1)$. They mainly consider ISD attacks for syndrome decoding and the folding attack for QA-SD. We refer to Section D in the Supplementary Material for a detailed discussion of known attacks to QA-SD assumptions. The parameters chosen by [Bom24] seem aggressive since QA-SD is less studied compared to syndrome decoding and efficient attacks may appear in the future. We modify [Bom24] to choose the aggressive parameters of the QA-SD problem for $\mathcal{R} = \mathbb{F}_p[\mathtt{X}_1, \cdots, \mathtt{X}_n]/(\mathtt{X}_1^2-1, \cdots, \mathtt{X}_n^2-1)$, where $\mathbb{F}_p$ is a field with an odd size.

[6] If we use optimization in [GYW+23], the number of calls can be further reduced to $6Nt + \lfloor\frac{\log p}{\lambda}\rfloor 4Nt$.

Conservative parameters. To prevent possible attacks, we give some conservative parameters by following the linear test framework[BCG+22], which covers most known attacks for LPN assumption, including ISD [Pra62, Ste88, MMT11, BJMM12, MO15, BM18, DEEK24] and SD attack [Jab01, Ove06, DT17, CDMT22, MT23, CDMT24]. Instead of enumerating the concrete complexity of each attack, this framework associates the security with the pseudo distance of code, which represents the minimum distance of the codeword that is efficiently computable. In [FL15], it is shown that a random quasi-abelian code meets GV bound, which achieves a good relative distance for a fixed rate. Note that GV bound is almost equivalent to Singleton bound when the field size is large enough.

7.2 Implementation and Evaluation

We implement the multivariate polynomial evaluation based on Walsh-Hadamard transform in C, which is an essential part of our PCG construction. We perform the experiment on a server with AMD EPYC 9754 Processor (2.25GHz CPU) and 512GB of RAM. All experiments are averaged across ten trials and evaluated on a single core.

Efficiency of Polynomial Evaluation. The efficiency of PCG for OLE based on Ring-LPN is only estimated in [BCG+20b] and there exists no complete instantiation yet. Therefore we implement FFT for the ring $\mathcal{R} = \mathbb{F}_p[\mathtt{X}]/(\mathtt{X}^N - 1)$ to make a fair comparison with [BCG+20b] , where N is a power of 2 and $N \mid p-1$. For both 32-bit and 64-bit primes, we benchmark the runtime of FFT and Walsh-Hadamard transform and display the results in Table 4, which shows that Walsh-Hadamard transform is 4.5x-12.7x faster than FFT.

Efficiency of Encoding. Based on our implementation of multivariate polynomial evaluation, we implement the encoding of QA code with the compression factor 2. The encoding of QA code consists of two Walsh-Hadamard transforms and a single component-wise multiplication. To compare with state-of-the-art VOLE protocols, we use the libOTe library [PR] to benchmark the encoding time of expand-accumulate (EA) code [BCG+22] and expand-convolute (EC) code [RRT23].[7] While the security of our QA-SD assumption is mainly determined by the Hamming weight t and compression ratio c, the security of syndrome decoding of EA code and EC code depends on more parameters. To make a fair comparison, we choose the parameters of EA and EC code with the same security level. For EA code, we set the expander weight and rate to 21 and 0.2, respectively.[8] For EC code, we set the expander weight and convolution state

[7] To make a fair comparison, we add the support of prime field operations for libOTe by modifying the CoeffCtx file.

[8] In the security analysis of EA code, they suggest that the rate of their code is at most $\frac{1}{\ln p}$. Thus, it should be decreased significantly with the size of field. We choose a relatively large rate anyway.

λ	N	c	Conservative			Aggressive		
			t	seed size	stretch	t	seed size	stretch
80	2^{20}	2	41	$2^{17.2}$	13	33	$2^{16.6}$	22
80	2^{20}	3	26	$2^{17.1}$	15	19	$2^{16.2}$	27
80	2^{20}	4	20	$2^{17.2}$	13	9	$2^{15.0}$	65
128	2^{20}	2	67	$2^{19.2}$	3	57	$2^{18.8}$	5
128	2^{20}	3	42	$2^{19.1}$	4	24	$2^{17.6}$	11
128	2^{20}	4	33	$2^{19.3}$	3	15	$2^{17.1}$	15
80	2^{25}	2	41	$2^{17.6}$	350	33	$2^{16.9}$	540
80	2^{25}	3	26	$2^{17.5}$	370	19	$2^{16.6}$	693
80	2^{25}	4	20	$2^{17.5}$	352	9	$2^{15.3}$	1667
128	2^{25}	2	67	$2^{19.6}$	87	57	$2^{19.2}$	115
128	2^{25}	3	42	$2^{19.4}$	94	24	$2^{17.9}$	275
128	2^{25}	4	33	$2^{19.6}$	86	15	$2^{17.4}$	380

Table 3: Comparison of concrete parameters and seed size of PCG for OLE from QA-SD assumption over $\mathbb{F}_p$, where p is a large prime such that $\log_2 p \approx 64$, λ is the security parameter, N is the number of OLE instances and c is the compression factor. The seed size is counted as the equivalent number of field elements and stretch is the ratio between the size of N random OLE instances and the seed size of PCG.

N	$\log_2 p \approx 32$		$\log_2 p \approx 64$	
	Ring-LPN	QA-SD	Ring-LPN	QA-SD
2^{20}	75 ms	6 ms	111 ms	13 ms
2^{21}	157 ms	12 ms	239 ms	29 ms
2^{22}	329 ms	27 ms	489 ms	69 ms
2^{23}	0.69 s	0.07 s	1.03 s	0.14 s
2^{24}	1.44 s	0.13 s	2.17 s	0.29 s
2^{25}	3.03 s	0.26 s	4.50 s	0.62 s
2^{26}	6.30 s	0.57 s	9.62 s	1.73 s

Table 4: The runtime of FFT for Ring-LPN and Walsh-Hadamard transform for QA-SD to obtain N evaluations over $\mathbb{F}_p$. To permit FFT, the prime p is chosen to satisfy that $N \mid p-1$.

size to 7 and 24, respectively. The rate of EC code is set to 0.5, the same as our QA code. We list the elaborate encoding time of these codes in Table 5. The experimental result shows that encoding of QA code is 3.5x-12.6x and 3.2x-11.2x faster than EA and EC code, respectively.

N	$\log_2 p \approx 32$			$\log_2 p \approx 64$		
	EA	EC	QA	EA	EC	QA
2^{20}	157 ms	168 ms	15 ms	190ms	171 ms	40 ms
2^{21}	379 ms	340 ms	34 ms	443 ms	393 ms	87 ms
2^{22}	859 ms	762 ms	68 ms	906 ms	846 ms	193 ms
2^{23}	1.77 s	1.65 s	0.15 s	1.85 s	1.74 s	0.41 s
2^{24}	3.64s	3.42 s	0.32 s	3.95 s	3.52 s	0.85 s
2^{25}	7.61 s	6.82 s	0.65 s	7.82 s	7.29 s	1.73 s
2^{26}	15.78 s	13.88 s	1.37 s	16.34 s	14.72 s	4.59 s

Table 5: The encoding time of expand-accumulate (EA) code, expand-convolute (EC) code and quasi-abelian (QA) code to generate a syndrome of length N over $\mathbb{F}_p$. For QA code, we set the compression factor $c = 2$. For EA code, we set expander weight $w = 21$ and rate $R = 0.2$. For EC code, we set expander weight $w = 7$, convolution state size $m = 24$ and rate $R = 0.5$.

Efficiency of VOLE. We use libOTe library to fully implement our new VOLE protocol $\mathsf{Cons}^{\mathbb{F}_p}_{\mathsf{VOLE}}$ in Section 5. Our implementation consists of the implementation of PPRF and the encoding of QA code. As a comparison, we also use libOTe library to implement the VOLE based on EA and EC codes. Following Table 3, we choose $t = 64$ and $c = 2$ for our $\mathsf{Cons}^{\mathbb{F}_p}_{\mathsf{VOLE}}$ to achieve 128-bit security. We directly follow libOTe to set the Hamming weight t of EA and EC code to 248, which is due to the low minimum distance of these codes. The experimental results in Table 6 show that our VOLE protocol is 4.3x-5.7x and 3.5x-5.3x faster than VOLE based on EA and EC code, respectively.

Efficiency of OLE. We use libOTe library to fully implement our new OLE protocol $\mathsf{Cons}^{\mathbb{F}_p}_{\mathsf{OLE}}$ over $\mathbb{F}_p$ for $\log_2 p \approx 64$ with a distributed setup in Section 6. Note that it is basically impractical to implement OLE protocol based on Ring-LPN assumption. The reason is that to utilize FFT algorithm of univariate polynomial, the domain of FSS in $\mathsf{Cons}^{\mathbb{F}_p}_{\mathsf{OLE}}$ must be a cyclic group of size $N \mid p-1$. This implies that the distributed point function realizing FSS must include expensive BitAdd that are used to compute bit-wise shares over $\mathbb{Z}_N$. The concrete efficiency of such FSS is not known yet. On the other hand, the OLE protocol based on our QA-SD assumption resorts to FSS whose domain is defined as a

N	$\log_2 p \approx 32$			$\log_2 p \approx 64$		
	EA	EC	QA	EA	EC	QA
2^{20}	545 ms	542 ms	122 ms	714 ms	552 ms	148 ms
2^{21}	1.27 s	1.09 s	0.23 s	1.56 s	1.26 s	0.31 s
2^{22}	2.73 s	2.49 s	0.48 s	3.26 s	2.74 s	0.68 s
2^{23}	5.71 s	5.32 s	1.06 s	6.81 s	5.64 s	1.33 s
2^{24}	11.95 s	10.86 s	2.14 s	13.85 s	11.55 s	2.95 s
2^{25}	24.44 s	22.03 s	4.43 s	28.39 s	23.34 s	5.94 s
2^{26}	49.61 s	44.78 s	8.73 s	58.36 s	47.13 s	13.35 s

Table 6: The runtime of VOLE protocols based on expand-accumulate (EA) code, expand-convolute (EC) code and quasi-abelian (QA) code to generate a VOLE instance of length N with 128-bit security. For QA code, we set compression factor $c = 2$ and Hamming weight $t = 64$. For EA code, we set expander weight $w = 21$ and rate $R = 0.2$. For EC code, we set expander weight $w = 7$, convolution state size $m = 24$ and rate $R = 0.5$. The Hamming weight t of EA and EC code is 248, which is computed by the libOTe library.

vector space $\mathbb{F}_2^n$. This is implemented with a distributed setup as class RegularDPF in libOTe library. We modify the source code of RegularDPF to support the range of prime field $\mathbb{F}_p$. From Table 7, we conclude that our OLE protocol generates roughly 27000 OLEs per second.

N	$(c,t) = (2,64)$		$(c,t) = (3,32)$	
	Runtime of OLE (s)	Communication Cost (MB)	Runtime of OLE (s)	Communication Cost (MB)
2^{18}	9.49	12.84	10.55	7.79
2^{19}	18.65	13.85	21.41	8.36
2^{20}	37.68	14.86	43.37	8.93
2^{21}	81.78	15.86	85.01	9.497
2^{22}	159.38	16.87	173.19	10.06
2^{23}	318.57	17.88	350.84	10.63
2^{24}	761.04	18.89	782.54	11.19

Table 7: There are two settings of our OLE implementations over $\mathbb{F}_p$ for $\log_2 p \approx 64$, i.e., $(c,t) = (2,64)$ and $(c,t) = (3,32)$. The performance indicates that the runtime of $(c,t) = (2,64)$ is shorter while the communication cost of $(c,t) = (3,32)$ is smaller.

Acknowledgment. The authors would like to thank anonymous reviewers at CRYPTO 2026 for their helpful suggestions. The work was supported in part by the National Key Research and Development Program of China under Grants 2022YFA1004900, 2023YFE0123900 and 2025YFA1017202, and the National Natural Science Foundation of China under Grants 12361141818, 12426302, 12271084 and 12571577.

References

ABG+24. Amit Agarwal, Elette Boyle, Niv Gilboa, Yuval Ishai, Mahimna Kelkar, and Yiping Ma. Compressing unit-vector correlations via sparse pseudorandom generators. In *CRYPTO 2024*, volume 14927 of *LNCS*, pages 346–383. Springer, 2024. 3

Alm23. Josh Alman. Faster walsh-hadamard transform and matrix multiplication over finite fields using lookup tables. In *SOSA 2023*, pages 137–144. SIAM, 2023. 18

AR75. Nasir Ahmed and Kamisetty Ramamohan Rao. *Walsh-Hadamard Transform*, pages 99–152. Springer Berlin Heidelberg, Berlin, Heidelberg, 1975. 16

AR23. Josh Alman and Kevin Rao. Faster walsh-hadamard and discrete fourier transforms from matrix non-rigidity. In *STOC 2023*, pages 455–462. ACM, 2023. 18

ARR25. Amit Agarwal, Srinivasan Raghuraman, and Peter Rindal. Fully distributed multi-point functions for pcgs and beyond. *IACR Cryptol. ePrint Arch.*, 2025:2294, 2025. 8

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. 3, 6, 7, 8, 10, 11, 25, 42, 43, 49

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. 3, 4, 6, 7, 8, 9, 11, 15, 39, 42, 43, 49

BCF+25. Martijn Brehm, Binyi Chen, Ben Fisch, Nicolas Resch, Ron D. Rothblum, and Hadas Zeilberger. Blaze: Fast snarks from interleaved RAA codes. In *EUROCRYPT 2025*, volume 15604 of *LNCS*, pages 123–152. Springer, 2025. 3, 6

BCG+17. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In *ACM CCS 2017*, pages 2105–2122. ACM, 2017. 3

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. 3, 8, 9, 12, 14, 21, 22, 38

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. 3, 4, 7, 12

BCG+20a. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In Sandy Irani, editor, *FOCS 2020*, pages 1069–1080. IEEE, 2020. 6

BCG+20b. 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. 3, 4, 7, 8, 9, 11, 23, 26, 39

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. 3, 4, 5, 6, 9, 21, 26

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. 3

BCGI18. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *CCS 2018*, pages 896–912. ACM, 2018. 3, 8, 9

BCH+25. Fabrice Benhamouda, Caicai Chen, Shai Halevi, Yuval Ishai, Hugo Krawczyk, Tamer Mour, Tal Rabin, and Alon Rosen. Encrypted matrix-vector products from secret dual codes. In *CCS 2025*, pages 394–408. ACM, 2025. 3

BDHV25. Charles Bouillaguet, Claire Delaplace, Mickaël Hamdad, and Damien Vergnaud. Practical cryptanalysis of pseudorandom correlation generators based on quasi-abelian syndrome decoding. In *ASIACRYPT 2025*, volume 16248 of *LNCS*, pages 436–465. Springer, 2025. 6, 42, 43, 44, 49, 50

BDSW23. Carsten Baum, Samuel Dittmer, Peter Scholl, and Xiao Wang. Sok: vector ole-based zero-knowledge protocols. *Des. Codes Cryptogr.*, 91(11):3527–3561, 2023. 3

Bea91. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO 1991*, volume 576 of *LNCS*, pages 420–432. Springer, 1991. 2

BFK+24. Alexander R. Block, Zhiyong Fang, Jonathan Katz, Justin Thaler, Hendrik Waldner, and Yupeng Zhang. Field-agnostic snarks from expand-accumulate codes. In *CRYPTO 2024*, volume 14929 of *LNCS*, pages 276–307. Springer, 2024. 5, 6

BGH+24. Elette Boyle, Niv Gilboa, Matan Hamilis, Yuval Ishai, and Yaxin Tu. Improved constructions for distributed multi-point functions. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 44–44. IEEE Computer Society, 2024. 3, 14

BGI15. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *EUROCRYPT 2015*, volume 9057 of *LNCS*, pages 337–367. Springer, 2015. 13

BGI16. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *ACM CCS 2016*, pages 1292–1303. ACM, 2016. 3, 7, 8, 13, 14, 25, 39, 47

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. 26, 42

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. 26, 42

BØ23. Pierre Briaud and Morten Øygarden. A new algebraic approach to the regular syndrome decoding problem and implications for PCG constructions.

In *EUROCRYPT 2023*, volume 14008 of *LNCS*, pages 391–422. Springer, 2023. 49

Bom24. Maxime Bombar. Estimator complexity for the $\mathbb{F}_4$OLEAGE PCG. https://github.com/mbombar/estimator_folding, 2024. 25, 50

CDE+18. Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. Spd$\mathbb{Z}_{2^k}$: Efficient MPC mod $\mathbb{Z}_{2^k}$ for dishonest majority. In *CRYPTO 2018*, volume 10992 of *LNCS*, pages 769–798. Springer, 2018. 2, 3

CDMT22. Kévin Carrier, Thomas Debris-Alazard, Charles Meyer-Hilfiger, and Jean-Pierre Tillich. Statistical decoding 2.0: Reducing decoding to LPN. In *ASIACRYPT 2022*, volume 13794 of *LNCS*, pages 477–507. Springer, 2022. 26, 42

CDMT24. Kévin Carrier, Thomas Debris-Alazard, Charles Meyer-Hilfiger, and Jean-Pierre Tillich. Reduction from sparse LPN to LPN, dual attack 3.0. In *EUROCRYPT 2024*, volume 14656 of *LNCS*, pages 286–315. Springer, 2024. 26, 42

CRR21. Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In *CRYPTO 2021*, volume 12827 of *LNCS*, pages 502–534. Springer, 2021. 3

CS17. Thomas W. Cusick and Pantelimon Stanica. Chapter 2 - fourier analysis of boolean functions. In Thomas W. Cusick and Pantelimon Stanica, editors, *Cryptographic Boolean Functions and Applications (Second Edition)*, pages 7–29. Academic Press, second edition edition, 2017. 16

CT65. James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965. 10

DEEK24. Léo Ducas, Andre Esser, Simona Etinski, and Elena Kirshanova. Asymptotics and improvements of sieving for codes. In *EUROCRYPT 2024*, volume 14656 of *LNCS*, pages 151–180. Springer, 2024. 26, 42

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. 2, 3, 8

DS17. Jack Doerner and Abhi Shelat. Scaling ORAM for secure computation. In *ACM CCS 2017*, pages 523–535. ACM, 2017. 7

DT17. Thomas Debris-Alazard and Jean-Pierre Tillich. Statistical decoding. In *ISIT*, pages 1798–1802. IEEE, 2017. 26, 42

EXY22. Daniel Escudero, Chaoping Xing, and Chen Yuan. More efficient dishonest majority secure computation over $\mathbb{Z}_{2^k}$ via galois rings. In *CRYPTO 2022*, volume 13507 of *LNCS*, pages 383–412. Springer, 2022. 2, 3

FL15. Yun Fan and Liren Lin. Thresholds of random quasi-abelian codes. *IEEE Trans. Inf. Theory*, 61(1):82–90, 2015. 5, 26

FL22. Yun Fan and Liren Lin. Asymptotic properties of quasi-group codes. *CoRR*, abs/2203.00958, 2022. 5, 19

GGM86. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986. 36

GI14. Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658. Springer, 2014. 3, 13

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 2023*, volume 14004 of *LNCS*, pages 330–362. Springer, 2023. 7, 25, 39

HKL+12. Stefan Heyse, Eike Kiltz, Vadim Lyubashevsky, Christof Paar, and Krzysztof Pietrzak. Lapin: An efficient authentication protocol based on Ring-LPN. In *FSE 2012*, volume 7549 of *LNCS*, pages 346–365. Springer, 2012. 9

Jab01. A. Kh. Al Jabri. A statistical decoding algorithm for general linear block codes. In *IMACC*, volume 2260 of *LNCS*, pages 1–8. Springer, 2001. 26, 42

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. 2, 3

KPR+25. Vladimir Kolesnikov, Stanislav Peceny, Rahul Rachuri, Srinivasan Raghuraman, Peter Rindal, and Harshal Shah. Block-accumulate codes: Accelerated linear codes for pcgs and ZK. *IACR Cryptol. ePrint Arch.*, page 1828, 2025. 3, 4, 6

KPRR25. Vladimir Kolesnikov, Stanislav Peceny, Srinivasan Raghuraman, and Peter Rindal. Stationary syndrome decoding for improved pcgs. In *CRYPTO 2025*, volume 16000 of *LNCS*, pages 284–317. Springer, 2025. 3, 8, 44, 49, 50

KPTZ13. Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM CCS 2013*, pages 669–684. ACM, 2013. 14

KSS09. Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *CANS 2009*, volume 5888 of *LNCS*, pages 1–20. Springer, 2009. 8

Kun79. Henry O. Kunz. On the equivalence between one-dimensional discrete walsh-hadamard and multidimensional discrete fourier transforms. *IEEE Trans. Computers*, 28(3):267–268, 1979. 11

LPR13. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-lwe cryptography. In *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, 2013. 10

LXY24. Fengrun Liu, Xiang Xie, and Yu Yu. Scalable multi-party computation protocols for machine learning in the honest-majority setting. In *USENIX Security 2024*. USENIX Association, 2024. 4

LXYY25a. Zhe Li, Chaoping Xing, Yizhou Yao, and Chen Yuan. Efficient pseudorandom correlation generators for any finite field. In *EUROCRYPT 2025*, volume 15605 of *LNCS*, pages 145–175. Springer, 2025. 3, 6, 7, 8, 10, 15, 43

LXYY25b. Zhe Li, Chaoping Xing, Yizhou Yao, and Chen Yuan. Efficient pseudorandom correlation generators over $\mathbb{Z}/p^k\mathbb{Z}$. In *CRYPTO 2025*, volume 16003 of *LNCS*, pages 207–240. Springer, 2025. 3, 6, 8, 10

MM12. Dan C. Marinescu and Gabriela M. Marinescu. Chapter 1 - preliminaries. In Dan C. Marinescu and Gabriela M. Marinescu, editors, *Classical and Quantum Information*, pages 1–131. Academic Press, Boston, 2012. 16

MMT11. Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 107–124. Springer, 2011. 26, 42

MO15. Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 203–228. Springer, 2015. 26, 42

MT23. Charles Meyer-Hilfiger and Jean-Pierre Tillich. Rigorous foundations for dual attacks in coding theory. In *TCC 2023*, volume 14372 of *LNCS*, pages 3–32. Springer, 2023. 26, 42

Obe07. Ulrich Oberst. The fast fourier transform. *SIAM J. Control. Optim.*, 46(2):496–540, 2007. 11

Ove06. Raphael Overbeck. Statistical decoding revisited. In *ACISP*, volume 4058 of *LNCS*, pages 283–294. Springer, 2006. 26, 42

PR. Lance Roy Peter Rindal. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe. 26

Pra62. Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Inf. Theory*, 8(5):5–9, 1962. 26, 42

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. 3, 4, 6, 9, 21, 26

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. 2, 3, 8

Sen11. Nicolas Sendrier. Decoding one out of many. In *PQCrypto*, volume 7071 of *LNCS*, pages 51–67. Springer, 2011. 42

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. 26, 42

Wei03. Steven H. Weintraub. *Representation theory of finite groups: algebra and arithmetic*, volume 59 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2003. 18

WYX+21. Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In *USENIX Security 2021*, pages 501–518. USENIX Association, 2021. 4

YWL+20. Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In *CCS 2020*, pages 1607–1626. ACM, 2020. 3

Supplementary Material

A More Preliminaries

A.1 PCG

To give a formal definition, we start with correlation generators and reverse-sampleable correlation generators.

Definition A.1 (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 A.2 (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 $\mathcal{C}(1^\lambda)$.

Definition A.3 (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 RSample is the reverse sampling algorithm for correlation $\mathcal{C}$.

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.4 (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

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

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

A.2 PPRF

Definition A.5 (t-Puncturable Pseudorandom Function). *A puncturable pseudorandom function (PPRF) with key space* $\mathcal{K}$*, domain* $\mathcal{X}$ *and range* $\mathcal{Y}$*, is a pseudorandom function* F *with an additional punctured key space* $\mathcal{K}_p$ *with three probabilistic polynomial-time algorithms* $(\mathsf{F.Gen}, \mathsf{F.Puncture}, \mathsf{F.Eval})$ *such that*

1. $\mathsf{F.Gen}(1^\lambda)$ *outputs a random key* $K \in \mathcal{K}$*.*
2. $\mathsf{F.Puncture}(K, S)$ *receives the input* $S \subset \mathcal{X}$ *of size* t *and* K *and outputs a punctured key* $K\{S\} \in \mathcal{K}_p$*.*
3. $\mathsf{F.Eval}(K\{S\}, x)$ *receives the punctured key* $K\{S\}$ *punctured at all points in* S *and a point* $x \in \mathcal{X}$ *output* $F(K, x)$ *if* $x \notin S$ *and* $\perp$ *otherwise.*

A PPRF can be constructed from any length-doubling pseudorandom generator and the GGM tree [GGM86]. Recall that in VOLE, the sender holds the vector $\mathbf{v} = (v_1, \ldots, v_n)$ and the receiver holds a value Δ. If $\mathbf{v}$ is a t-sparse vector, then the support set S of $\mathbf{v}$ can be used as the input of PPRF. The output of the receiver is a pseudorandom vector $\mathbf{u}$ evaluating F at each index and the output of the sender is a pseudorandom vector $\mathbf{u}$ puncturing out the indices in S. For index i in S, the sender will receive $F(k, i) + v_i$. In this case, the seed size of PPRF is $t\lambda \log n$ bits, which is smaller than that of distributed point function. Therefore, we can design a more efficient PCG for VOLE by replacing distributed point function with PPRF.

A.3 Properties of Group Rings

Lemma A.6. *Let R be a ring, $\mathbb{G}$ be a multiplicative group, and $R[\mathbb{G}]$ be a group ring in Definition 2.2. The followings hold:*

1. *$R[\mathbb{G}]$ is commutative if and only if R is commutative and $\mathbb{G}$ is Abelian.*
2. *If R has identity 1, and $\mathbb{G}$ has identity e, then $1e$ is the identity of $R[\mathbb{G}]$.*

In this work, we particularly instantiate $R[\mathbb{G}]$ with R a large prime field $\mathbb{F}_p$ and $\mathbb{G}$ an Abelian group. By fixing an ordering $g_1, \ldots, g_n$ of the elements in $\mathbb{G}$, $\mathbb{F}_p[\mathbb{G}]$ is isomorphic to $\mathbb{F}_p^n$ with respect to $\varphi : \mathbb{F}_p[\mathbb{G}] \to \mathbb{F}_p^n, \sum_{i=1}^n a_i g_i \mapsto (a_1, \ldots, a_n)$, where the multiplication over $\mathbb{F}_p^n$ is defined as follows:

$$(a_1, \ldots, a_n) \cdot (b_1, \ldots, b_n) = \Big(\sum_{g_i g_j = g_1} a_i b_j, \ldots, \sum_{g_i g_j = g_n} a_i b_j \Big).$$

In fact, the multiplication over $\mathbb{F}_p^n$ can be characterized by standard matrix multiplications. For $a := \sum_{i=1}^n a_i g_i \in \mathbb{F}_p[\mathbb{G}]$, define

$$\mathbf{A} = \begin{pmatrix} \varphi(a \cdot g_1) \\ \vdots \\ \varphi(a \cdot g_n) \end{pmatrix} \in \mathbb{F}_p^{n \times n}.$$

Essentially, the matrix $\mathbf{A}$ defines the multiply-by-a map $m \mapsto ma$ in $\mathbb{F}_p[G]$ with respect to the basis $(g_1, \ldots, g_n)$.

$$\varphi(ma) = \varphi(m)\mathbf{A} = (m_1, \ldots, m_n) \begin{pmatrix} \varphi(a \cdot g_1) \\ \vdots \\ \varphi(a \cdot g_n) \end{pmatrix}.$$

Lemma A.7. *Let $\mathbb{G}_1, \mathbb{G}_2$ be two finite Abelian groups. Then*

$$\mathbb{F}_p[\mathbb{G}_1 \times \mathbb{G}_2] \simeq \mathbb{F}_p[\mathbb{G}_1] \otimes_{\mathbb{F}_p} \mathbb{F}_p[\mathbb{G}_2],$$

where $\times$ denotes the direct product, and $\otimes_{\mathbb{F}_p}$ denotes the tensor product over $\mathbb{F}_p$.

Examples. We give concrete examples of group rings.

- Let $\mathbb{G} = \{1\}$ be the trivial group consisting of only an identity. Then $\mathbb{F}_p[G]$ is isomorphic to $\mathbb{F}_p$.
- Let $\mathbb{G} = \mathbb{Z}/n\mathbb{Z}$ be the cyclic group with n elements. Assuming that n is coprime to p, then one can see $\mathbb{F}_p[\mathbb{G}]$ is essentially the polynomial ring $\mathbb{F}_p[\mathtt{X}]/(\mathtt{X}^n - 1)$, with respect to the ordering $\overline{0}, \overline{1}, \ldots, \overline{n-1}$ and the isomorphism $\varphi : \mathbb{F}_p[\mathbb{G}] \to \mathbb{F}_p[\mathtt{X}]/(\mathtt{X}^n - 1), \sum_{i=0}^{n-1} a_i \overline{i} \mapsto \sum_{i=0}^{n-1} a_i \mathtt{X}^i$.
- Let $\mathbb{G} = \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$ with n_1, n_2 coprime to p. Then by Lemma A.7,

$$\begin{aligned} \mathbb{F}_p[\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}] &\simeq \mathbb{F}_p[\mathbb{Z}_{n_1}] \otimes_{\mathbb{F}_p} \mathbb{F}_p[\mathbb{Z}_{n_2}] \simeq \mathbb{F}_p[\mathtt{X}]/(\mathtt{X}^{n_1} - 1) \otimes_{\mathbb{F}_p} \mathbb{F}_p[\mathtt{Y}]/(\mathtt{Y}^{n_2} - 1) \\ &\simeq \mathbb{F}_p[\mathtt{X}, \mathtt{Y}]/(\mathtt{X}^{n_1} - 1, \mathtt{Y}^{n_2} - 1). \end{aligned}$$

Moreover, assuming n divides $p-1$, and $\alpha_0, \ldots, \alpha_{n-1}$ are n roots of $\mathtt{X}^n - 1$ over $\mathbb{F}_p$. Then, $\phi : f(\mathtt{X}) \mapsto (f(\alpha_0), \ldots, f(\alpha_{n-1}))$ is a ring isomorphism of $\mathbb{F}_p[\mathtt{X}]/(\mathtt{X}^n - 1)$ and the vector space $\mathbb{F}_p^n$. Hence, $\phi \circ \varphi : \mathbb{F}_p[\mathbb{G}] \to \mathbb{F}_p^n$ is a ring isomorphism. Finally let $n \mid p-1$ and $\mathbb{G} = \mathbb{Z}_n^\ell$, then by Lemma A.7, $\mathbb{F}_p[\mathbb{G}] \simeq \mathbb{F}_p[\mathtt{X}_1, \ldots, \mathtt{X}_\ell]/(\mathtt{X}_1^n - 1, \ldots, \mathtt{X}_\ell^n - 1) \simeq \mathbb{F}_p^N$, where $N = n^\ell$.

B Missing Proofs

B.1 Proof for VOLE

Theorem B.1 (Theorem 5.1). *Assume a secure* PPRF *scheme for sums of point functions and QA-SD$(\mathcal{R}, t)$ is hard. Then there exists a PCG construction* $\mathsf{Cons}_{\mathsf{VOLE}}^{\mathbb{F}_p}$ *that generates VOLE correlations over $\mathbb{F}_p$. If the* PPRF *is based on a* PRG $: \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ *via the* PRG*-based construction from [BCG+19a], for generating $N := 2^n$ VOLE correlations over $\mathbb{F}_p$, we have that:*

- *For party with $\sigma = 0$, the seed size is at most: $2t(\lambda \log \frac{N}{t} + \log N + \log q)$ bits.*
- *For party with $\sigma = 1$, the seed size is at most: $2\lambda + \log q$ bits.*
- *For party with $\sigma = 0$, the computation of* ***Expand*** *can be done with at most $(4 + \lfloor \frac{\log p}{\lambda} \rfloor)N$* PRG *operations and $2N$ multiplications and $2N + 4N \log N$ additions in $\mathbb{F}_p$*
- *For party with $\sigma = 1$, the computation of* ***Expand*** *can be done with at most $(4 + \lfloor \frac{\log p}{\lambda} \rfloor)N$* PRG *operations and N multiplications and $N + 2N \log N$ additions in $\mathbb{F}_p$.*

Proof. We assume the QA-SD assumption holding for regular noise distribution to optimize our parameters. We follow the argument in [BCG+19a] to estimate the seed size and PRG calls. The correctness is straightforward and the security analysis is based on a QA-SD assumption and the underlying PPRF scheme. Here we consider the view of P_1, i.e., $\sigma = 1$. The case of $\sigma = 0$ works in a similar way. Let $\mathbf{k}_0, \mathbf{k}_1 \xleftarrow{\$} \mathsf{PCG.Gen}$ with associated expanded outputs $(\mathbf{x}, \mathbf{z}_0)$ and $(\Delta, \mathbf{z}_1)$. We need to show that

$$\{(\mathbf{k}_1, (\mathbf{x}, \mathbf{z}_0))\} \equiv \left\{ (\mathbf{k}_1, (\tilde{\mathbf{x}}, \tilde{\mathbf{z}}_0)) \mid \begin{array}{l} \tilde{\mathbf{x}} \xleftarrow{\$} \mathbb{F}_p^N, \\ \tilde{\mathbf{z}}_1 = \Delta * \tilde{\mathbf{x}} - \mathbf{z}_0 \end{array} \right\}$$

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

- Replace $\mathbf{z}_0$ by $\Delta * \mathbf{x} - \mathbf{z}_1$.
- Replace punctured PPRF keys $\mathbf{k}_s^{\mathbf{A}}, \mathbf{k}_e^{\mathbf{B}}$ 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 PPRF scheme, this distribution is indistinguishable from the original distribution.
- Since the simulated key is independent of the output, we can then replace $\mathbf{x}$ with a fresh $\widetilde{\mathbf{x}}$ by the QA-SD assumption.
- Reverse step 2 by using PPRF security property once again.

□

B.2 Proof for OLE

Theorem B.2 (Theorem 6.1). *Assume a secure* FSS *scheme* SPFSS *for sums of point functions and QA-SD($\mathcal{R}, t$) 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* $N := 2^n$ *OLE correlations over* $\mathbb{F}_p$*, we have that:*

- *Each party's seed has a maximum size around:* $4t^2\big(\log \frac{\lambda}{t}(\lambda+2)+\lambda+\log p\big)+2t(\log N+\log p)$ *bits.*
- *The computation of* ***Expand*** *can be done with at most* $(2+\lfloor\frac{\log p}{\lambda}\rfloor)4Nt$[9] PRG *operations and* $3N$ *multiplications and* $4N+6N\log N$ *additions in* $\mathbb{F}_p$*.*

Proof. Regarding the size of the different parameters, we also follow optimizations of [BCG+20b, BCCD23], e.g., assuming the QA-SD assumption holds also for regular noise distributions. As for SPFSS scheme, since our domain size is $N = 2^n$, we follow the argument in [BGI16] to estimate the seed size and PRG calls.

Correctness is straightforward given by the Construction $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$. Security is based on a QA-SD assumption and the underlying SPFSS scheme, and we prove in a similar way to those of [BCG+20b, 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 $(\mathbf{x}_0, \mathbf{z}_0)$ and $(\mathbf{x}_1, \mathbf{z}_1)$.

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

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

- Replace $\mathbf{z}_1$ by $\mathbf{x}_0 * \mathbf{x}_1 - \mathbf{z}_0$.
- Step by step replace each of 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 $\mathbf{x}_1$ by a fresh $\tilde{\mathbf{x}}_1$. It is also impossible to distinguish this distribution from the previous one, since the K_0^i are now completely independent of $\mathbf{x}_1$, and we can rely on the QA-SD assumption.
- Reverse step 2 by using the SPFSS security property once again.

We remark that QA-SD assumption implies $x_1 \in \mathcal{R}$ can not be distinguished from uniform, and Lemma 2.5 implies that $\mathbf{x}_1$ computed from a random $x_1 \in \mathcal{R}$ are uniformly distributed in $\mathbb{F}_p^N$. □

[9] If we use optimization in [GYW+23], the number of calls can be further reduced to $6Nt + \lfloor\frac{\log p}{\lambda}\rfloor 4Nt$.

C PCG Setup Protocols

In this section, we provide distributed two-party protocols for PCG for OLE correlations assuming the pseudorandomness of QA-SD assumptions. We only present the distributed setup protocol to realize the functionality $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{OLE-Setup}}$. The expand protocol is exactly the same as in $\mathsf{Cons}^{\mathbb{F}_p}_{\mathsf{OLE}}$as there is no interaction between two parties. We also note that our PCG protocol can be used in a bootstrapping flavor, a smaller number of PCG outputs will be reserved for the next round of PCG seed generations.

Functionality 4: $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{OLE-Setup}}$

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

We first introduce two functionalities $\mathcal{F}_{\mathsf{2PC}}$ and $\mathcal{F}_{\mathsf{DPF}}$, which support general two-party computation and distributed point function, respectively. Given these functionalities, we show a protocol that realizes the setup functionality $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{OLE-Setup}}$against semi-honest adversaries. The malicious secure protocol can be constructed similarly by replacing $\mathcal{F}_{\mathsf{DPF}}$ with a maliciously secure distributed point function and using the authenticated secret sharing scheme.

Functionality 5: $\mathcal{F}_{\mathsf{2PC}}$

This functionality operates on elements of a finite ring R, generally an integer ring $\mathbb{Z}_D$ or a finite field $\mathbb{F}_p$, where $D, p \in \mathbb{N}$. Each value stored by the functionality is associated with a unique identifier that is given to all parties. Let $[x]^R$ denote the identifier for a value $x \in R$ that is stored in the functionality. **Functionality**: This functionality supports the following operations:

- $\mathsf{Input}(P_\sigma, x)$: Receive a value $x \in R$ from party P_σ and store $[x]^R$.
- $\mathsf{Add}([x]^R, [y]^R)$: Compute $z = x + y$ and store $[z]^R$.
- $\mathsf{Mul}([x]^R, [y]^R)$: Compute $z = x \cdot y$ and store $[z]^R$.

Functionality 6: $\mathcal{F}_{\mathsf{DPF}}$

Parameters: Security parameter λ, distributed point function $\mathsf{DPF} = (\mathsf{DPF}.\mathsf{Gen}, \mathsf{DPF}.\mathsf{Eval})$ with domain $[0, D)$ and range $\mathbb{F}_p$, where $D, p \in \mathbb{N}$.
Functionality: On input $[\alpha]^{\mathbb{Z}_{D-1}}$ and $[\beta]^{\mathbb{F}_p}$:

1. Sample keys $(K_0, K_1) \leftarrow \mathsf{DPF}.\mathsf{Gen}(1^\lambda, \alpha, \beta)$.
2. For $\sigma \in \{0,1\}$, output K_σ to P_σ.

Protocol 7: $\Pi^{\mathbb{F}_p}_{\mathsf{OLE-Setup}}$

Parameters: Security parameter 1^λ, length $N = 2^n$, $\mathbb{F}_p$ and $\mathcal{R} = \mathbb{F}_p[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^2 - 1, \ldots, \mathtt{X}_n^2 - 1)$. Let DPF be a distributed point function $\mathsf{DPF} := (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ with domain $[0, N)$ and range $\mathbb{F}_p$. 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 $\mathbf{A}^0_\sigma, \mathbf{A}^1_\sigma \xleftarrow{\$} [0, N)^t$ and $\mathbf{b}^0_\sigma, \mathbf{b}^1_\sigma \xleftarrow{\$} \mathbb{F}_p^t$. We remark that the pair $(\mathbf{A}^i_\sigma, \mathbf{b}^i_\sigma)$ defines a t-sparse element in $\mathcal{R}$.
2. P_σ inputs the positions and the values. For $i \in \{0, 1\}, j \in [t]$,
$$\left[\mathbf{A}^i_\sigma[j]\right]^{\mathbb{Z}_{N-1}} \leftarrow \mathsf{Input}(P_\sigma, \mathbf{A}^i_\sigma[j]), \left[\mathbf{b}^i_\sigma[j]\right]^{\mathbb{F}_p} \leftarrow \mathsf{Input}(P_0, \mathbf{b}^i_\sigma[j]).$$
3. Compute the cross sums of positions and products of values. For each $i, j \in [t]$,
$$\left[\alpha^{ij}_0\right]^{\mathbb{Z}_{N-1}} \leftarrow \mathsf{Add}(\left[\mathbf{A}^0_0[i]\right], \left[\mathbf{A}^0_1[j]\right]), \left[\beta^{ij}_0\right]^{\mathbb{F}_p} \leftarrow \mathsf{Mul}(\left[\mathbf{b}^0_0[i]\right], \left[\mathbf{b}^0_1[j]\right])$$
$$\left[\alpha^{ij}_1\right]^{\mathbb{Z}_{N-1}} \leftarrow \mathsf{Add}(\left[\mathbf{A}^1_0[i]\right], \left[\mathbf{A}^0_1[j]\right]), \left[\beta^{ij}_1\right]^{\mathbb{F}_p} \leftarrow \mathsf{Mul}(\left[\mathbf{b}^1_0[i]\right], \left[\mathbf{b}^0_1[j]\right])$$
$$\left[\alpha^{ij}_2\right]^{\mathbb{Z}_{N-1}} \leftarrow \mathsf{Add}(\left[\mathbf{A}^0_0[i]\right], \left[\mathbf{A}^1_1[j]\right]), \left[\beta^{ij}_2\right]^{\mathbb{F}_p} \leftarrow \mathsf{Mul}(\left[\mathbf{b}^0_0[i]\right], \left[\mathbf{b}^1_1[j]\right])$$
$$\left[\alpha^{ij}_3\right]^{\mathbb{Z}_{N-1}} \leftarrow \mathsf{Add}(\left[\mathbf{A}^1_0[i]\right], \left[\mathbf{A}^1_1[j]\right]), \left[\beta^{ij}_3\right]^{\mathbb{F}_p} \leftarrow \mathsf{Mul}(\left[\mathbf{b}^1_0[i]\right], \left[\mathbf{b}^1_1[j]\right])$$
4. Sample the FSS key shares via calling $\mathcal{F}_{\mathsf{DPF}}$ with domain $[0, N)$ and range $[0, N)$. For each $\kappa \in [0, 3], i, j \in [t]$,
$$(K^{ij\kappa}_0, K^{ij\kappa}_1) \leftarrow \mathcal{F}_{\mathsf{DPF}}(\left[\alpha^{ij}_\kappa\right]^{\mathbb{Z}_{N-1}}, \left[\beta^{ij}_\kappa\right]^{\mathbb{F}_p}).$$
5. P_σ outputs $\mathsf{k}_\sigma := \left(\left\{K^{ij\kappa}_\sigma\right\}_{i,j\in[t],\kappa\in[0,3]}, \left\{\mathbf{A}^i_\sigma, \mathbf{b}^i_\sigma\right\}_{i\in\{0,1\}}\right)$.

Theorem C.1. *The protocol $\Pi^{\mathbb{F}_p}_{\mathsf{OLE-Setup}}$ securely realizes the PCG seed generation functionality $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{OLE-Setup}}$ with security against semi-honest adversaries in the $(\mathcal{F}_{\mathsf{2PC}}, \mathcal{F}_{\mathsf{DPF}})$-hybrid model.*

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

D Concrete Attacks to QA-SD

In this section, we discuss the known attacks against our QA-SD assumption to analyze its security. We concentrate on a general QA-SD instance for $\mathcal{R} = \mathbb{F}_p[\mathtt{X}_1, \cdots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \cdots, \mathtt{X}_n^d - 1)$ where $\mathbb{F}_p$ is a finite field and $d \mid p - 1$. Given a QA-SD instance $(\mathbf{a}, b = \langle \mathbf{a}, \mathbf{e}\rangle + e_0)$ where $\mathbf{a} = (a_1, \cdots, a_{c-1}) \xleftarrow{\$} \mathcal{R}^{c-1}$, $\mathbf{e} = (e_1, \cdots, e_{c-1})$ and $\{e_i\}_{i\in[0,c-1]}$ are t-sparse elements of $\mathcal{R}$, the goal of an

adversary is either to distinguish this tuple with uniform distribution or recover the vectors $\{e_i\}_{i\in[0,c-1]}$.

Recall that a QA-SD instance can be regarded as a linear code over $\mathbb{F}_p$ with code length cN and dimension $(c-1)N$, which achieves a code rate $1-1/c$. In this case, a QA-SD instance could be regarded as an LPN instance for a structured quasi-abelian code, and general attacks for LPN are directly applicable to QA-SD problems. We first discuss general attacks for LPN, and then introduce two special attacks for QA-SD: decoding one out of many (DOOM) and the folding attack. Furthermore, we present an additional attack targeted at small fields, which was recently proposed in [BDHV25].

General Attacks for LPN. Here we consider the syndrome decoding problem, which is the dual form of LPN problem: Given $H \in \mathbb{F}_p^{k\times n}$ and $\mathbf{b} \in \mathbb{F}_p^k$, the goal is to find $\mathbf{e} \in \mathbb{F}_p^n$ such that $H \cdot \mathbf{e} = \mathbf{b}$ and $\mathtt{wt}(\mathbf{e}) = w$. General attacks for syndrome decoding can be divided into the following two categories.

Information Set Decoders (ISD). Till now, most of the best algorithms for solving the syndrome decoding problem have fallen in the family of ISD algorithms, which date back to Prange's seminal work [Pra62] and have been extended to several variants [Ste88, MMT11, BJMM12, MO15, BM18, DEEK24]. The ISD attacks follow the same principle, making the bet that in a certain subset of k positions (the "information set") there exist very few errors and applying this to speed up the decoding to recover $\mathbf{s}$ and $\mathbf{e}$. We refer to [BBC+24, Appendix C] for an elaborate analysis of the concrete computation complexity of known ISD attacks for the syndrome decoding problem.

Statistical Decoding (SD). The other category of attacks, called statistical decoding, has been introduced in [Jab01, Ove06] and was improved in the subsequent works [DT17, CDMT22, MT23, CDMT24]. Instead of recovering $\mathbf{s}$ and $\mathbf{e}$, SD attacks aim to distinguish the pseudorandom codeword $\mathbf{b}$ with a random vector. Although SD attacks outperform ISD attacks in some settings, this happens only for codes with a small rate, which is generally below 1/2 as shown in [CDMT24, Figure 1]. Due to the high code rate in our construction, we will not consider SD attacks.

Decoding One Out of Many. For QA-SD problems, the quasi-abelian structure permits an additional decoding one out of many (DOOM) attack [Sen11], which brings a $d^{n/2}$ speedup over any attacks. In detail, given a QA-SD instance $(\mathbf{a}, b = \langle \mathbf{a}, \mathbf{e}\rangle + e_0)$, the adversary can obtain d^n associated instances $(\mathbf{a}, b \cdot \mathtt{X}^i)$, where $\{\mathtt{X}^i\}_{i\in[0,N-1]}$ is a basis of the ring $\mathcal{R}$.

Folding Attack. Following [BCCD23, BBC+24], we describe the folding attack for QA-SD problem in the language of group ring, which represents the multivariate polynomial ring $\mathbb{F}_p[\mathtt{X}_1, \cdots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \cdots, \mathtt{X}_n^d - 1)$ as $\mathbb{F}_q[\mathbb{G}]$, where

$\mathbb{G} = \mathbb{Z}_d^n$. Let $\mathbb{H}$ denote a subgroup of $\mathbb{G}$, which can always be represented as $\mathbb{Z}_d^{n'}$ for $1 \leq n' \leq n$. The group homomorphism $\mathbb{G} \to \mathbb{G}/\mathbb{H}$ induces a morphism:

$$\pi_{\mathbb{H}} : \mathbb{F}_p[\mathbb{G}] \to \mathbb{F}_p[\mathbb{G}/\mathbb{H}], \quad \sum_{g \in \mathbb{G}} a_g g \mapsto \sum_{\bar{g} \in \mathbb{G}/\mathbb{H}} \left(\sum_{\bar{h} \in \mathbb{H}} a_{g+h} \right) \bar{g}$$

This maps a vector of length $|\mathbb{G}|$ to another one of length $|\mathbb{G}/\mathbb{H}|$ by summing the $|\mathbb{H}|$ entries of each coset of $\mathbb{G}/\mathbb{H}$.

Given a QA-SD instance $(\mathbf{a}, b = \langle \mathbf{a}, \mathbf{e} \rangle + e_0)$, the mapping $\pi_{\mathbb{H}}$ is extended to obtain the following:

$$\pi_{\mathbb{H}}(\mathbf{a}, b) = (\pi_{\mathbb{H}}(\mathbf{a}), \pi_{\mathbb{H}}(b)) = (\pi_{\mathbb{H}}(\mathbf{a}), \langle \pi_{\mathbb{H}}(\mathbf{a}), \pi_{\mathbb{H}}(\mathbf{e}) \rangle + \pi_{\mathbb{H}}(e_0)) \in \mathbb{F}_p[\mathbb{G}/\mathbb{H}]^c$$

This output corresponds to a new linear code with a smaller length $|\mathbb{G}/\mathbb{H}|$ and the same code rate $1 - 1/c$. To analyze the security of the folded QA-SD assumption, it is essential to estimate the Hamming weight of $\{\pi_{\mathbb{H}}(e_i)\}_{i \in [0, c-1]}$, which is quantitatively given in [BBC+24, Proposition 18].

To launch a folding attack, the adversary may try the subgroups $\mathbb{H}$ of $\mathbb{G}$ to find a folded instance with a smaller Hamming weight u and run the best decoding algorithm for syndrome decoding problem. On average, the runtime for this attack is

$$\frac{C_{\mathsf{LPN}}(u) + C_{\mathsf{Folding}}}{\Pr[\mathsf{wt}(\pi_{\mathbb{H}}(e)) = u]} \tag{4}$$

where $C_{\mathsf{LPN}}(u)$ denotes the complexity of attacks for syndrome decoding problem with Hamming weight $\leq u$ (as previously discussed, the complexity of various ISD attacks), and C_{Folding} denotes the complexity of folding operations, which is the same as the size of the original instance. Note that a DOOM attack still provides a $\sqrt{|\mathbb{G}/\mathbb{H}|}$ speedup for the folded instance.

Attacks for Small Fields. In [BDHV25], Bouillaguet et al. propose a practical attack that breaks some parameters provided by [BCCD23, BBC+24, LXYY25a]. For a QA-SD instance $(\mathbf{a}, b = \langle \mathbf{a}, \mathbf{e} \rangle + e_0)$, let S denote the set of evaluation points for a variable with $|S| = d$. Since $\{a_i\}_{i \in [c-1]}$ are random public polynomials, the adversary knows d^n evaluations of these polynomials over S^n and $1/p^{c-1}$ fraction of them satisfy that $a_1(\mathbf{x}) = \cdots = a_{c-1}(\mathbf{x}) = 0$ for some $\mathbf{x} \in S^n$. In this case, we have that $b(\mathbf{x}) = e_0(\mathbf{x})$ and we obtain around d^n/p^{c-1} evaluations of a t-sparse polynomial e_0. Rewrite e_0 as

$$e_0(\mathtt{X}_1, \cdots, \mathtt{X}_n) = \sum_{j=0}^{d-1} \mathtt{X}_i^j P_j(\mathtt{X}_1, \cdots, \mathtt{X}_{i-1}, \mathtt{X}_{i+1}, \cdots, \mathtt{X}_n)$$

for some $i \in [n]$. According to the analysis of [BDHV25], we expect to obtain $d^{n-1} p^{-d(c-1)}$ evaluations of $\{P_j\}_{j \in [0, d-1]}$. For $j \in [0, d-1]$, when $\mathtt{X}_i^j$ never occurs in any of the t monomials of e_0, P_j will always be zero, as well as its evaluations. This happens with a probability $(1 - 1/d)^t$. To distinguish a QA-SD

instance from a random instance, the adversary checks whether there exists $j \in [0, d-1]$ such that all evaluations of P_j are equal to zero. In [BDHV25], a series of improvements is proposed to accelerate the practical cryptanalysis.

To defend against the attack of [BDHV25], we need to either decrease the number of evaluations of P_j (increase the compression factor c) or reduce the probability that P_j is zero (increase the Hamming weight t). For a field $\mathbb{F}_p$ such that $\log_d p \geq \frac{n-1}{d(c-1)}$, this attack does not work anymore, since the adversary has no evaluation of P_j. In our instantiation, we generally set $d = 2, n = 20, 25, 30$ and p as 32-bit, 64-bit primes; thus our construction is resistant to the attack of [BDHV25].

E Stationary Syndrome Decoding of Quasi-Abelian

In this section, we introduce a new assumption, stationary syndrome decoding of quasi-abelian (QA-SSD), and migrate the PCG construction and security analysis of QA-SD to QA-SSD.

E.1 Definition of QA-SSD

Recall that in syndrome decoding assumption, if $\mathbf{e} \in \mathbb{F}_p^n$ is a secret sparse vector and G is a $k \times n$ public matrix selected uniformly at random, then $G\mathbf{e}$ can not be distinguishable from a uniformly random vector in $\mathbb{F}_p^n$ given G. Recently, Kolesnikov et. al. [KPRR25] proposed a new assumption called Stationary Syndrome Decoding (SSD) so as to improve the performance of PCG. In this new assumption, we are given r pairs $(\mathbf{e}_r, G_1), \ldots, (\mathbf{e}_r, G_r)$ where $\mathbf{e}_1, \ldots, \mathbf{e}_r$ are correlated secret sparse vectors such that the support set of them are all the same. Then, this new assumption claims that $G_i\mathbf{e}_i$ is indistinguishable from a uniformly random vector given $G_1, \ldots, G_r$.

Definition E.1 (Stationary Syndrome Decoding [KPRR25]). *Stationary syndrome decoding is parameterized by an implicit computational security parameter κ, a field $\mathbb{F}$, dimensions $n, k, r \in \mathbb{N}$ with $k < n$ and distributions $\mathcal{L} \in Dist[\{0,1\}^n], \mathcal{G} \in Dist[\mathbb{F}^{k \times n}]$. For sample $L \leftarrow \mathcal{L}$ and for $i \in [r]$, sample $\mathbf{e}_i \in L \odot \mathbb{F}^n, G_i \xleftarrow{\$} \mathcal{G}$, The $(n, k, r, \mathbb{F}, \mathcal{L}, \mathcal{G})$-stationary syndrome decoding assumption states:*

$$\{(G_i, \mathbf{b}_i) : \mathbf{b}_i := G_i\mathbf{e}_i\}_{i \in [r]} \approx \{(G_i, \mathbf{b}_i) : \mathbf{b}_i \xleftarrow{\$} \mathbb{F}^k\}_{i \in [r]}.$$

This new assumption makes the PCG construction more efficient because we can invoke a function secret sharing scheme once to share noise vectors $\mathbf{e}_1, \ldots, \mathbf{e}_r$. In their security analysis, all known attacks against syndrome decoding do not gain advantages from the additional information of correlated noise vectors. In view of this, we propose a new assumption called Stationary Syndrome Decoding of Quasi-Abelian codes (QA-SSD).

Definition E.2 (Search QA-SSD). *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* $\mathbf{s}_1, \ldots, \mathbf{s}_r$ *be random* ct*-sparse elements of* $\mathcal{R}^c$ *with the same support set and* $e_1, \ldots, e_r$ *be random* t*-sparse elements of* $\mathcal{R}$ *with the same support set. Given access to* r *pairs of the form* $(\mathbf{a}_1, \langle \mathbf{a}_1, \mathbf{s}_1 \rangle + e_1), \ldots, (\mathbf{a}_r, \langle \mathbf{a}_r, \mathbf{s}_r \rangle + e_r)$, *the goal is to recover* $\mathbf{s}_1, \ldots, \mathbf{s}_r$ *and* $e_1, \ldots, e_r$.

Definition E.3 (Decisional QA-SSD). *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:*
$\mathcal{D}_0 : \{(\mathbf{a}_1, u_1), \ldots, (\mathbf{a}_r, u_r)\}$, *where* $\mathbf{a}_i \xleftarrow{\$} \mathcal{R}^c, u_i \xleftarrow{\$} \mathcal{R}$.
$\mathcal{D}_1 : \{(\mathbf{a}_1, \langle \mathbf{a}_1, \mathbf{s}_1 \rangle + e_1), \ldots, (\mathbf{a}_r, \langle \mathbf{a}_r, \mathbf{s}_r \rangle + e_r)\}$, *where* $\mathbf{a}_i \xleftarrow{\$} \mathcal{R}^c, \mathbf{s}_1, \ldots, \mathbf{s}_r$ *are random* ct*-sparse elements of* $\mathcal{R}^c$ *with the same support set and* $e_1, \ldots, e_r$ *are random* t*-sparse elements of* $\mathcal{R}$ *with the same support set.*

Similar to QA-SD, the above problems are referred as search QA-SSD$(c, \mathcal{R}, t, r)$ and decisional QA-SSD$(c, \mathcal{R}, t, r)$ problems, respectively. We say that the search (decisional) QA-SSD$(c, \mathcal{R}, t, r)$ assumption holds when there exists no PPT algorithm that solves the problem with a non-negligible advantage. We mainly consider the case $c = 2$ in our construction, and denote QA-SSD$(\mathcal{R}, t)$ as a simplified version of QA-SSD$(2, \mathcal{R}, t)$.

E.2 PCG Based on QA-SSD Assumption

In this subsection, we present the PCG of OLE based on QA-SSD. PCG for VOLE based on this assumption can be constructed in a similar way.

The performance of PCG for OLE depends on two key ingredients: the function secret sharing scheme and the evaluation algorithm. Our Walsh-Hadamard transform speeds up the evaluation algorithm. To improve the performance of SPFSS, we propose a new PCG construction based on QA-SSD assumption. The QA-SSD assumption can batch multiple instances of PCG together and run the SPFSS once. Let us briefly explain the construction. We use the same notation as in Section 6. Assume that we want to generate rN random OLE instances over $\mathbb{F}_p$ with $N = 2^n$. Let $a_1, \ldots, a_r$ be random polynomials in $\mathcal{R}$. For $\sigma \in \{0,1\}$ and $i \in [r]$, let $s_\sigma^{(1)}, \ldots, s_\sigma^{(r)}$ be t-sparse elements of $\mathcal{R}$ such that the index set of nonzero elements in $\{s_\sigma^{(i)}\}_{i \in [r]}$ is the same. We do the same thing for $e_\sigma^{(1)}, \ldots, e_\sigma^{(r)}$ as well. According to the QA-SSD assumption, we can claim that

$$x_\sigma^{(i)} = a_i \cdot s_\sigma^{(i)} + e_\delta^{(i)}$$

are pseudorandom for each $\sigma \in \{0,1\}$ and $i \in [r]$. Our goal is to compute the product

$$x_0^{(i)} \cdot x_1^{(i)} = a_i^2 s_0^{(i)} s_1^{(i)} + a_i (s_0^{(i)} e_1^{(i)} + s_1^{(i)} e_0^{(i)}) + e_0^{(i)} e_1^{(i)}.$$

for $i = 1, \ldots, r$. Instead of distributing the additive shares of cross-term $s_0^{(i)} s_1^{(i)}$ for $i = 1, \ldots, r$ separately, we can batch the cross term of $s_0^{(i)} s_1^{(i)}$ for $i = 1, \ldots, r$

together by utilizing the fact that the index set of nonzero elements in $s_\sigma^{(i)}$ for $i = 1, \ldots, r$ are the same. For example, let α_i be the first index of non-zero element in $s_0^{(i)} s_1^{(i)}$ and β_i be the value of this non-zero element. According to our construction, we have that $\{\alpha_i\}_{i\in[r]}$ are all the same, which we can denote as α. Then, we let $\beta = (\beta_1, \ldots, \beta_m) \in \mathbb{F}_{p^r}$ and send $(\alpha, \beta) \in [0, N) \times \mathbb{F}_{p^r}$ as the input to SPFSS. Thus, we can run SPFSS once instead of r times to save both the seed size and the number of PRG calls.

Construction 8: $\mathsf{Cons}_{\mathsf{BOLE}}^{\mathbb{F}_p}$

PARAMETER: Security parameter λ, noise weight $t = t(\lambda)$, $N = 2^n$, r, $\mathcal{R} = \mathbb{F}_p[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^2 - 1, \ldots, \mathtt{X}_n^2 - 1)$. An FSS scheme $(\mathsf{SPFSS.Gen}, \mathsf{SPFSS.FullEval})$ for sums of t^2 point functions, with domain $[0, N)$ and range $\mathbb{F}_{p^r}$.
PUBLIC INPUT: Uniformly random elements $a_1, \ldots, a_r \in \mathcal{R}$ and their Walsh-Hadamard transform $\widehat{\mathbf{a}}_i = \mathbf{WHT}(a_i)$, $\widehat{\mathbf{a}}_i * \widehat{\mathbf{a}}_i$.
CORRELATION: After expansion, outputs $(\mathbf{x}_0^{(i)}, \mathbf{z}_0^{(i)}) \in \mathbb{F}_p^{2N}$ and $(\mathbf{x}_1^{(i)}, \mathbf{z}_1^{(i)}) \in \mathbb{F}_p^{2N}$ such that $\mathbf{x}_0^{(i)} * \mathbf{x}_1^{(i)} = \mathbf{z}_0^{(i)} + \mathbf{z}_1^{(i)}$ for $i = 1, \ldots, r$.
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 $\boldsymbol{s}_\sigma, \boldsymbol{e}_\sigma \leftarrow \mathbb{F}_{p^r}^t$.
2. Sample FSS keys according to Eq.(3):
   $(K_0^0, K_1^0) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0 \boxplus \mathbf{A}_1^0, \boldsymbol{s}_0 \otimes \boldsymbol{s}_1)$,
   $(K_0^1, K_1^1) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1 \boxplus \mathbf{A}_1^0, \boldsymbol{e}_0 \otimes \boldsymbol{s}_1)$,
   $(K_0^2, K_1^2) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0 \boxplus \mathbf{A}_1^1, \boldsymbol{s}_0 \otimes \boldsymbol{e}_1)$,
   $(K_0^3, K_1^3) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1 \boxplus \mathbf{A}_1^1, \boldsymbol{e}_0 \otimes \boldsymbol{e}_1)$,
3. For $\sigma \in \{0, 1\}$, let $\mathbf{k}_\sigma = \Big((K_\sigma^i)_{i\in[0,3]}, (\mathbf{A}_\sigma^0, \boldsymbol{s}_\sigma), (\mathbf{A}_\sigma^1, \boldsymbol{e}_\sigma)\Big)$.
4. Output $(\mathbf{k}_0, \mathbf{k}_1)$.

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

1. Parse $\mathbf{k}_\sigma$ as $\Big((K_\sigma^i)_{i\in[0,3]}, (\mathbf{A}_\sigma^0, \boldsymbol{s}_\sigma), (\mathbf{A}_\sigma^1, \boldsymbol{e}_\sigma)\Big)$.
2. Parse $\boldsymbol{s}_\sigma \in \mathbb{F}_{p^r}^t$ as $(\boldsymbol{s}_\sigma^{(1)}, \ldots, \boldsymbol{s}_\sigma^{(r)})$ where $\boldsymbol{s}_\sigma^{(i)} \in \mathbb{F}_p^t$. [a] Do the same thing to $\boldsymbol{e}_\sigma$ to obtain $(\boldsymbol{e}_\sigma^{(1)}, \ldots, \boldsymbol{e}_\sigma^{(r)})$.
3. Define elements of $\mathcal{R}$
$$s_\sigma^{(i)} = \sum_{\ell=1}^{t} \boldsymbol{s}_\sigma^{(i)}[\ell] \cdot \mathbf{A}_\sigma^0[\ell], \quad e_\sigma^{(i)} = \sum_{\ell=1}^{t} \boldsymbol{e}_\sigma^{(i)}[\ell] \cdot \mathbf{A}_\sigma^1[\ell].$$
4. For $i \in [0, 3]$, compute $(u_{\sigma,i}^{(1)}, \ldots, u_{\sigma,i}^{(r)}) \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^i)$, viewed as $\mathcal{R}^m$ elements.
5. For $i \in [0, 3], j \in [r]$, compute $\widehat{\mathbf{u}}_{\sigma,i}^{(j)} = \mathbf{WHT}(u_{\sigma,i}^{(j)})$, $\widehat{\mathbf{s}}_\sigma^{(j)} = \mathbf{WHT}(s_\sigma^{(j)})$ and $\widehat{\mathbf{e}}_\sigma^{(j)} = \mathbf{WHT}(e_\sigma^{(j)})$.
6. Compute [b]
$$\mathbf{z}_\sigma^{(j)} = \widehat{\mathbf{a}}_j * \widehat{\mathbf{a}}_j * \widehat{\mathbf{u}}_{\sigma,0}^{(j)} + \widehat{\mathbf{a}}_j * (\widehat{\mathbf{u}}_{\sigma,1}^{(j)} + \widehat{\mathbf{u}}_{\sigma,2}^{(j)}) + \widehat{\mathbf{u}}_{\sigma,3}^{(j)}, \quad \mathbf{x}_\sigma = \widehat{\mathbf{a}}_j * \widehat{\mathbf{s}}_\sigma^{(j)} + \widehat{\mathbf{e}}_\sigma^{(j)}$$

7. Output $(\mathbf{x}_\sigma^{(j)}, \mathbf{z}_\sigma^{(j)})_{j\in[r]}$.

[a] Write the i-th component of $\boldsymbol{s}_\sigma$ as a vector $\mathbf{s}_i = (s_{i,1}, \ldots, s_{i,r}) \in \mathbb{F}_p^m$. Then, $\boldsymbol{s}_\sigma^{(j)} = (s_{1,j}, \ldots, s_{t,j})$.

[b] We only need $3rN$ multiplications and $4rN + 6rN \log N$ additions in $\mathbb{F}_p$ to compute $(\mathbf{x}_\sigma^{(j)}, \mathbf{z}_\sigma^{(j)})_{j\in[r]}$.

Theorem E.4. *Assume a secure* FSS *scheme* SPFSS *for sums of point functions and QA-SSD$(\mathcal{R}, t)$ is hard. Then there exists a PCG construction* $\mathsf{Cons}^{\mathbb{F}_p}_{\mathsf{BOLE}}$ *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 $N' := r2^n$ OLE correlations over $\mathbb{F}_p$, we have that:*

- *Each party's seed has a maximum size around:* $4t^2\big(\lambda \log \frac{2N'}{rt} + \lambda + 1 + r\log p\big) + 2t(\log \frac{N'}{r} + r \log p)$ *bits.*
- *The computation of* ***Expand*** *can be done with at most* $(2 + \lfloor \frac{r \log p}{\lambda} \rfloor) \frac{4N't}{r}$ PRG *operations and $3N'$ multiplications and $4N' + 6N' \log \frac{N'}{r}$ additions in $\mathbb{F}_p$.*

Proof. Compared to Theorem 6.1, the only change is the range of SPFSS which is reflected in the seed size. In comparison, the seed size is $4rt^2\big(\lambda \log \frac{2N'}{rt} + \lambda + 1 + \log p\big) + 2tr(\log \frac{N'}{r} + \log p)$ and the number of PRG calls is $(2 + \lfloor \frac{\log p}{\lambda} \rfloor)4N't$ if we run $\mathsf{Cons}^{\mathbb{F}_p}_{\mathsf{OLE}}$ m times. The security proof is almost the same as in Theorem 6.1 and thus we omit the details. □

Theorem E.5. *The PCG construction* $\mathsf{Cons}^{\mathbb{F}_p}_{\mathsf{BOLE}}$ *has programmability.*

Proof. The argument is very similar to Theorem 6.2. We omit the details. □

E.3 Distributed Setup from QA-SSD

In this subsection, we present a two-party protocol for OLE correlations based on QA-SSD assumptions. Recall that in $\mathsf{Cons}^{\mathbb{F}_p}_{\mathsf{BOLE}}$, we merge m instances over $\mathbb{F}_p$ into one instance over $\mathbb{F}_{p^m}$, which allows us to run the SPFSS once. In this distributed protocol, we want to apply the same trick to improve the performance. Because the t-sparse element in $\mathcal{R}$ is now shared via an additive secret sharing scheme, we need the following fact about additive secret sharing scheme to merge.

Lemma E.6. *Let $[x_1], \ldots, [x_r]$ be m additive secret sharings over $\mathbb{F}_p$. Assume that $[x_i] = (x_{i,1}, \ldots, x_{i,n})$. Then, $[x] = (x_{1,i}, \ldots, x_{r,i})_{i\in[n]}$ is an additive secret sharing over $\mathbb{F}_{p^r}$ with $x = (x_1, \ldots, x_r)$.*

Proof. Since $[x_i]$ is an additive secret sharing scheme, we have $x_i = \sum_{j=1}^n x_{i,j}$. This implies that $(x_1, \ldots, x_r) = (\sum_{j=1}^n x_{1,j}, \ldots, \sum_{j=1}^n x_{r,j})$ and thus $[x]$ is an additive secret sharing scheme. The privacy and reconstruction argument is

straightforward. Without loss of generality, we assume that the adversary obtains the first $n-1$ shares of $[x]$. This means the adversary obtains the first $n-1$ shares of each $[x_i]$. According to the privacy of additive secret sharing scheme, this adversary learns nothing about x_i for $i \in [m]$. The reconstruction is obvious and we omit it. □

From Lemma E.6, we can merge r additive secret sharing scheme over $\mathbb{F}_p$ into one additive secret sharing scheme over $\mathbb{F}_{p^r}$.

Functionality 9: $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{BOLE-Setup}}$

Parameters: Security parameter λ, a PCG for BOLE $\mathsf{PCG}_{\mathsf{BOLE}} := (\mathsf{PCG}_{\mathsf{BOLE}}.\mathsf{Gen}, \mathsf{PCG}_{\mathsf{BOLE}}.\mathsf{Expand})$ following Construction $\mathsf{Cons}^{\mathbb{F}_p}_{\mathsf{BOLE}}$.
Functionality:

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

Similar to the QA-SD assumption case, we can replace $\mathcal{F}_{\mathsf{DPF}}$ with a maliciously secure distributed point function and using the authenticated secret sharing scheme to construct the malicious secure two-party protocols for OLE correlations.

Protocol 10: $\Pi^{\mathbb{F}_p}_{\mathsf{BOLE-Setup}}$

Parameters: Security parameter λ, length $N = 2^n$, $\mathbb{F}_p$, r and $\mathcal{R} = \mathbb{F}_p[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^2 - 1, \ldots, \mathtt{X}_n^2 - 1)$. Let DPF be a distributed point function $\mathsf{DPF} := (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ with domain $[0, N)$ and range $\mathbb{F}_{p^r}$. Assume each variable is shared via additive sharing, for instance $[x] = (x_0, x_1)$ with $x_0 + x_1 = x$. Moreover, assume access to functionalities $\mathcal{F}_{\mathsf{2PC}}$, $\mathcal{F}_{\mathsf{DPF}}$.
Protocol:

1. P_σ samples random vectors $\mathbf{A}^0_\sigma, \mathbf{A}^1_\sigma \xleftarrow{\$} [0, N)^t$ and $\mathbf{b}^0_{\sigma,h}, \mathbf{b}^1_{\sigma,h} \xleftarrow{\$} \mathbb{F}_p^t$ for $h \in [r]$. We remark that the pair $(\mathbf{A}^i_\sigma, \mathbf{b}^i_{\sigma,h})$ defines a t-sparse element in $\mathcal{R}$ for $h \in [r]$.
2. P_σ inputs the positions and the values. For $i \in \{0,1\}, j \in [t], h \in [r]$,
$$\left[\mathbf{A}^i_\sigma[j]\right]^{\mathbb{Z}_{N-1}} \leftarrow \mathsf{Input}(P_\sigma, \mathbf{A}^i_\sigma[j]), \left[\mathbf{b}^i_{\sigma,h}[j]\right]^{\mathbb{F}_p} \leftarrow \mathsf{Input}(P_0, \mathbf{b}^i_{\sigma,h}[j]).$$
3. Compute the cross sums of positions and products of values. For each $i, j \in [t], h \in [r]$,
$$\left[\alpha_0^{ij}\right]^{\mathbb{Z}_{N-1}} \leftarrow \mathsf{Add}(\left[\mathbf{A}^0_0[i]\right], \left[\mathbf{A}^0_1[j]\right]), \left[\beta_0^{ijh}\right]^{\mathbb{F}_p} \leftarrow \mathsf{Mul}(\left[\mathbf{b}^0_{0,h}[i]\right], \left[\mathbf{b}^0_{1,h}[j]\right])$$
$$\left[\alpha_1^{ij}\right]^{\mathbb{Z}_{N-1}} \leftarrow \mathsf{Add}(\left[\mathbf{A}^1_0[i]\right], \left[\mathbf{A}^0_1[j]\right]), \left[\beta_1^{ijh}\right]^{\mathbb{F}_p} \leftarrow \mathsf{Mul}(\left[\mathbf{b}^1_{0,h}[i]\right], \left[\mathbf{b}^0_{1,h}[j]\right])$$
$$\left[\alpha_2^{ij}\right]^{\mathbb{Z}_{N-1}} \leftarrow \mathsf{Add}(\left[\mathbf{A}^0_0[i]\right], \left[\mathbf{A}^1_1[j]\right]), \left[\beta_2^{ijh}\right]^{\mathbb{F}_p} \leftarrow \mathsf{Mul}(\left[\mathbf{b}^0_{0,h}[i]\right], \left[\mathbf{b}^1_{1,h}[j]\right])$$
$$\left[\alpha_3^{ij}\right]^{\mathbb{Z}_{N-1}} \leftarrow \mathsf{Add}(\left[\mathbf{A}^1_0[i]\right], \left[\mathbf{A}^1_1[j]\right]), \left[\beta_3^{ijh}\right]^{\mathbb{F}_p} \leftarrow \mathsf{Mul}(\left[\mathbf{b}^1_{0,h}[i]\right], \left[\mathbf{b}^1_{1,h}[j]\right])$$

4. Merge $\left[\beta_\kappa^{ijh}\right]^{\mathbb{F}_p}$ for $h \in [r]$ into one secret sharing $\left[\beta_\kappa^{ij}\right]^{\mathbb{F}_{p^r}}$ according to Lemma E.6 for $\kappa \in [0,3]$.
5. Sample the FSS key shares via calling $\mathcal{F}_{\mathsf{DPF}}$ with domain $[0,N)$ and range $\mathbb{F}_{p^r}$. For each $\kappa \in [0,3], i,j \in [t]$,

$$(K_0^{ij\kappa}, K_1^{ij\kappa}) \leftarrow \mathcal{F}_{\mathsf{DPF}}(\left[\alpha_\kappa^{ij}\right]^{\mathbb{Z}_{N-1}}, \left[\beta_\kappa^{ij}\right]^{\mathbb{F}_{p^r}}).$$

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

Theorem E.7. *The protocol $\Pi^{\mathbb{F}_p}_{\mathsf{BOLE-Setup}}$ securely realizes the PCG seed generation functionality $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{BOLE-Setup}}$ with security against semi-honest adversaries in the $(\mathcal{F}_{\mathsf{2PC}}, \mathcal{F}_{\mathsf{DPF}})$-hybrid model.*

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

E.4 Security Analysis for QA-SSD

Kolesnikov et al. make an elaborate analysis to SSD assumption in [KPRR25]. For the linear attacks, they apply a unified linear test framework to argue the security. For the non-linear attacks, they mainly consider the algebraic attack in [BØ23] and show that the new assumption is resistant to this attack. To analyze the security of QA-SSD, we only need to focus on two attacks specialized for QA-SD (folding attack [BCCD23, BBC+24] and the attack for small fields [BDHV25]). However, unlike SD assumption, both attacks become more efficient in the stationary setting.

Folding attack. We briefly show how the number of instances r within QA-SSD makes it easier for the adversary to solve the problem. After receiving a QA-SSD instance $\{(\mathbf{a}_i, b_i = \langle \mathbf{a}_i, \mathbf{s}_i \rangle + e_i)\}_{i\in[r]}$, the adversary applies the folding attack to each QA-SD instance and selects the folded instance with the smallest Hamming weight. Then the adversary tries start-of-the-art decoding algorithms for this folded instance. Assume there exists at least one folded instance with Hamming weight u, then we rewrite Eq.(4) to estimate the complexity of folding attack for QA-SSD:

$$\frac{C_{\mathsf{LPN}}(u) + rC_{\mathsf{Folding}}}{1-(1-\Pr[\mathsf{wt}(\pi_{\mathbb{H}}(e))=u])^r} \geq \frac{C_{\mathsf{LPN}}(u) + rC_{\mathsf{Folding}}}{r\Pr[\mathsf{wt}(\pi_{\mathbb{H}}(e))=u]} \tag{5}$$

The inequality in Eq.(5) is taken from the union bound. From Eq.(5), we conclude that if the complexity of decoding is much larger than folding operations, the complexity to solve a QA-SSD problem is around t times smaller than QA-SD, which gives rise to a $O(\log r)$ bit loss of security.

Attack for small fields. For QA-SSD assumption, the feature "stationary" means that the sparse polynomials share the same patterns of monomials and only differ in the coefficients of these monomials. Since the attack for small fields [BDHV25] relies on the interpolation of sparse multivariate polynomials, resuing the positions of errors may result in a loss of security.

Here we give an improved attack of [BDHV25] to show the gain of the adversary from QA-SSD problem. Let us consider a QA-SSD instance $\{(\mathbf{a}_i, b_i = \langle \mathbf{a}_i, \mathbf{s}_i \rangle + e_i\}_{i\in[r]}$, where $\mathbf{a}_i = (a_{i,1}, \cdots, a_{i,c-1}) \xleftarrow{\$} \mathcal{R}^{c-1}$, $\mathbf{s}_i = (s_{i,1}, \cdots, s_{i,c-1})$, $\{s_{i,j}\}_{j\in[c-1]}$ and e_i are t-sparse polynomials. For $i \in [r]$, the adversary tries to find some n-dimensional points $\mathbf{x}_i$ such that each entry of $\mathbf{a}_i$ is zero at these points, then the adversary obtains some evaluations of e_i. Recall that e_i can be written as

$$e_i(\mathtt{X}_1, \cdots, \mathtt{X}_n) = \sum_{j=0}^{d-1} \mathtt{X}_k^j P_{i,j}(\mathtt{X}_1, \cdots, \mathtt{X}_{k-1}, \mathtt{X}_{k+1}, \cdots, \mathtt{X}_n)$$

for some $k \in [n]$. If there exists $i \in [r], j \in [d]$ such that $P_{i,j}$ is always equal to zero, we conclude that $\{P_{i',j}\}_{i'\neq i}$ are also zero polynomials. This is due to the same pattern of monomials within the QA-SSD problem. To distinguish a QA-SSD instance with a random distribution, the adversary collects the evaluations of $\{P_{i,j}\}_{i\in[r],j\in[d]}$ and judges whether there exists $j \in [d]$ such that all evaluations of $\{P_{i,j}\}_{i\in[r]}$ are zero. It is obvious that the adversary obtains r times more evaluations compared to the original attack in [BDHV25].

To defend against this attack, we require that $r < p^{d(c-1)}/d^{n-1}$ to guarantee the adversary has no evaluation of sub-polynomials. This gives an upper bound to the number of QA-SD instances r within a QA-SSD problem, which does not appear in the analysis of SSD assumption in [KPRR25].

E.5 Parameter Selection for QA-SSD

Similar to Section 7, we obtain both aggressive and conservative parameters for QA-SSD assumption. For aggressive parameters, we further modify the SageMath script in [Bom24] to support the analysis of folding attack in Section E.4. For conservative parameters, we combine the folding attack for QA-SSD and the analysis of SSD assumption in the linear test framework in [KPRR25] to obtain the Hamming weight t. The details are displayed in Table 8.

c	r	Conservative			Aggressive		
		t	seed size	stretch	t	seed size	stretch
2	2^4	138	$2^{21.2}$	14	59	$2^{18.9}$	69
2	2^8	143	$2^{21.3}$	212	60	$2^{18.9}$	1064
2	2^{12}	146	$2^{21.3}$	3427	62	$2^{19.0}$	15950
3	2^4	87	$2^{21.1}$	15	25	$2^{17.7}$	161
3	2^8	89	$2^{21.2}$	228	26	$2^{17.8}$	2382
3	2^{12}	92	$2^{21.3}$	3414	27	$2^{17.9}$	35534
4	2^4	69	$2^{21.3}$	13	15	$2^{17.2}$	239
4	2^8	71	$2^{21.3}$	202	16	$2^{17.2}$	3359
4	2^{12}	73	$2^{21.4}$	3051	16	$2^{17.2}$	56624

Table 8: Comparison of concrete parameters and seed size of PCG from QA-SSD assumption to generate $N = r \cdot 2^{20}$ OLEs over $\mathbb{F}_p$ under 128-bit security, where p is a large prime such that $\log_2 p \approx 64$, c is the compression factor and r is the number of instances within QA-SSD. The seed size is counted as the equivalent number of field elements and stretch is the ratio between the size of N random OLE instances and the seed size of PCG.