

Efficient Pseudorandom Correlation Generators over Binary Extension Fields and More

Zhe Li¹, Hongqing Liu²(✉), Chaoping Xing²(✉), Yizhou Yao²(✉), Chen Yuan²(✉)

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. In this work, we construct highly efficient pseudorandom correlation generators (PCGs) over binary extension fields. PCGs allow for the distribution of a great amount of correlated randomness by having each party locally expand a short seed, enabling efficient multi-party computation (MPC) protocols with silent and sublinear preprocessing. Our PCGs achieve *nearly linear time*, both asymptotically and concretely, where expanding N VOLEs (vector oblivious linear evaluations) over $\mathbb{F}_{2^k}$ consists of $O(N)$ PRG calls, $O(N)$ multiplications in $\mathbb{F}_{2^k}$ and $O(kN \log N)$ binary XORs. In fact, the $O(kN \log N)$ XORs are insignificant in practical parameter regimes where $N \approx 2^{20}$. Concretely, we achieve a throughput of $\approx 5,000,000$ VOLEs per second over $\mathbb{F}_{2^{128}}$, which is at least 4.7x, 1.9x and 1.6x faster than the state-of-the-art PCGs of Expand-Accumulate (EA), Block-Accumulate (BA) and Expand-Convolute (EC) codes, respectively. For oblivious linear evaluations (OLEs), expanding N instances requires $O(\lambda N)$ PRG calls, $O(N)$ multiplications in $\mathbb{F}_{2^k}$ and $O(kN \log N)$ binary XORs, where λ is the security parameter. Based on the libOTe library, we also give the first full implementation of PCGs for OLEs over extension fields, which shows a throughput of $\approx 42,000$ OLEs per second over $\mathbb{F}_{2^{128}}$.

At the heart of our technical contribution is an efficient multivariate FFT algorithm. It evaluates any n -variate polynomial $f(x_1, \dots, x_n)$ over $\mathbb{F}_{2^k}$ of individual degree ≤ 2 using exactly $\frac{5}{6}kN \log_3 N$ additions and $\frac{1}{6}kN \log_3 N$ multiplications in $\mathbb{F}_4$, where $N = 3^n$ and k is an arbitrary even integer. Note that this is considerably faster than normal FFT algorithms that require $O(N \log N)$ multiplications in the underlying field $\mathbb{F}_{2^k}$. Along the way, we generalize our approach to work over a degree- k Galois extension of $\mathbb{Z}_{2^\ell}$, denoted by $\mathbb{GR}(2^\ell, k)$. Similarly, our algorithm takes only $\frac{7}{6}kN \log_3 N$ additions and $\frac{1}{6}kN \log_3 N$ multiplications in $\mathbb{GR}(2^\ell, 2)$ to evaluate $f(x_1, \dots, x_n) \in \mathbb{GR}(2^\ell, k)[x_1, \dots, x_n]_{\leq 2}$. This implies practically efficient PCGs for (V)OLE over Galois rings.

As a side contribution of independent interest, we derive fast encoding algorithms for a family of Quasi-Abelian codes, which are even concretely 1.66x faster than Repeat-Accumulate-Accumulate (RAA) codes

over $\mathbb{F}_{2^{128}}$, whose encoding only involves a linear number of additions and two random permutations. As Quasi-Abelian codes have been proven to attain concretely large minimum distance in a recent work of Li et al. (CRYPTO 2026), this opens a door to building more efficient succinct arguments over extension fields.

Keywords: Pseudorandom correlation generators · Binary extension fields · Galois rings · Quasi-Abelian codes · Multivariate FFT

1 Introduction

MPC and correlated randomness. Secure multiparty computation (MPC) is a jewel of modern cryptography. Over the past few years, pseudorandom correlation generators (PCGs) [11, 12] have received attention due to their promising efficiency in MPC preprocessing, enabling practical implementations of modern MPC protocols. In PCG-based preprocessing, each party will be given a short seed, which then can be *efficiently* and *locally* expanded into a long pseudorandom string that satisfies a certain correlation. Typical and fundamental correlation types are oblivious linear evaluation (OLE) and vector OLE (VOLE), which suffice for most MPC protocols. Thanks to the rich literature [9, 11, 12, 14, 60, 25, 10, 52, 5, 45, 46, 42], so far we have already known how to construct quasi-linear time PCGs for (V)OLE over arbitrary finite fields [45], or more generally, arbitrary Galois rings [46], assuming the hardness of syndrome decoding problems. Moreover, many of them have been implemented with practical efficiency [10, 52, 4, 45, 46, 42], for instance, binary OLE can be generated with 2.2 million per second [45], while a recent work [43] reports a throughput of 27,000/s for OLE over large prime fields. However, there remains a huge efficiency gap for (V)OLE over *binary extension fields* as well as *Galois rings*, where existing works either incur a large computational overhead [14, 5] or do not support this regime [43]. Moreover, known constructions [46] also suffer from high costs of Galois ring operations, in particular, when the extension degree is large.

Why do binary extension fields and Galois Rings matter? Most maliciously secure MPC protocols work on a sufficiently large prime field, and focus on computing some arithmetic circuits that are defined over the same field. However, it is not always the case in practice. In many real-world applications, the task might be naturally expressed as a computer program, but converting it into an arithmetic circuit over fields incurs substantial overhead, as field operations do not align well with standard CPUs. Instead, computations over $\mathbb{F}_2$ and $\mathbb{Z}_{2^\ell}$ are often more native, as they naturally support fundamental instructions such as comparisons and bitwise manipulations. Therefore, protocols tailored for $\mathbb{F}_2$ or $\mathbb{Z}_{2^\ell}$ have received particular attention and moreover, recent works have proven their practical efficiency. However, to realize malicious security, these works often resort to their Galois extensions, i.e., $\mathbb{F}_{2^k} \supseteq \mathbb{F}_2$ and $\mathbb{GR}(2^\ell, k) \supseteq \mathbb{Z}_{2^\ell}$. That is, although the original task is over $\mathbb{F}_2$ ($\mathbb{Z}_{2^\ell}$), they essentially deal with $\mathbb{F}_{2^k}$ ($\mathbb{GR}(2^\ell, k)$), which

inevitably introduces an *embedding* overhead $\Omega(k)$. To mitigate this overhead, a line of works [21, 23, 27, 32] has developed a successful methodology, from the use of *reverse multiplication friendly embeddings* (RMFEs) [21, 27]. Unfortunately, RMFEs only amortize the online costs, while producing (V)OLEs in the preprocessing phase remains expensive.

Additionally, we have also seen the significance of binary extension fields and Galois Rings in building efficient proof systems [22, 33, 18, 47, 29, 19, 58, 39]. Remarkably, [29, 19] construct succinct non-interactive arguments over binary extension fields via some linear-time encodable codes, such as Spielman’s codes [54] and Repeat-Accumulate-Accumulate (RAA) codes [30]. Beyond that, [58, 39] further generalize to Galois rings by introducing Galois ring analogues of Reed-Solomon codes. In particular, an orthogonal work [47] also shows how to utilize PCGs for VOLE over Galois rings and RMFEs to build efficient interactive zero-knowledge protocols for $\mathbb{Z}_{2^\ell}$. Although these schemes over Galois rings are as efficient as over finite fields in terms of the number of arithmetic operations, they are in fact concretely much slower. Because, e.g., multiplication in $\mathbb{GR}(2^\ell, k)$ requires $O(\ell^2 k^2)$ bit operations, which is much more expensive than $\mathbb{F}_{2^k}$ for a typical $\ell = 64$. Moreover, unlike $\mathbb{F}_{2^k}$, the additive FFT (Fast Fourier Transform) algorithm [49] is not applicable to Galois ring $\mathbb{GR}(2^\ell, k)$. In contrast, multiplicative FFT requires $\mathbb{GR}(2^\ell, k)$ to have an $O(1)$ -smooth multiplicative subgroup G , and there are only a few choices of k , e.g., $k = 162$ implies a G of size 3^5 [39].

Efficient linear codes over $\mathbb{F}_{2^k}$ and $\mathbb{GR}(2^\ell, k)$. The key to improving concrete efficiency is to find a linear code with a good minimum distance such that expensive operations in $\mathbb{GR}(2^\ell, k)$ can be avoided as much as possible. In more detail, let $G \in (\mathbb{GR}(2^\ell, k))^{N \times cN}$ denote a generating matrix of a linear code $\mathcal{C}$ over a Galois ring. In PCG scenarios, the efficiency bottleneck lies at computing a syndrome $G \cdot \vec{e}$, while in succinct arguments, instead it is computing a codeword $\vec{m}^\top \cdot G$. Both applications require $\mathcal{C}$ to have concretely high minimum distance. However, known candidates of $\mathcal{C}$ all fall short in this case. For instance, to encode a message of length N with rate $1/2$, the Spielman-style code instantiated in Brakedown requires $34.57N$ multiplications in $\mathbb{GR}(2^\ell, k)$, which incurs a large computational overhead. RAA codes seem more suitable for Galois rings, since their encoding only consists of a linear number of additions and performing random permutations. However, permutations actually are not that cheap in implementation, and moreover, RAA codes suffer from a large failure probability when adapted to Galois rings [58, 19].

1.1 Our Contribution

As a stepping stone, we first propose efficient FFT algorithms for multivariate polynomials of individual degree 2 over $\mathbb{F}_{2^k}$ and $\mathbb{GR}(2^\ell, k)$, where $k \geq 2$ is an arbitrary even integer. In what follows, we view $\mathbb{F}_{2^k}$ as a special case of $\mathbb{GR}(2^\ell, k)$ where $\ell = 1$, and denote $\mathbb{GR}(2^\ell, k)$ by $\mathbb{GR}_k$ for simplicity. Unlike normal (univariate) FFTs, our algorithm completely avoids expensive multiplications in $\mathbb{GR}_k$, and does not rely on smooth multiplicative subgroups, making it suitable for

most k of interest, e.g., $k = 64$ or $k = 128$. Our results are summarized as follows.

Theorem 1.1 (Informal). *There exists an FFT algorithm over $\text{GR}_k = \mathbb{GR}(2^\ell, k)$ that evaluates any n -variate polynomial $f(x_1, \dots, x_n)$ with individual degree ≤ 2 at $N = 3^n$ points in time $O(kN \log N)$.*

We remark that the hidden constant of $O(kN \log N)$ is very small. Concretely, when $\ell = 1$, our FFT algorithm only performs $\frac{5}{6}kN \log_3 N$ additions and $\frac{1}{6}kN \log_3 N$ multiplications over $\mathbb{F}_4$. For $\ell \geq 2$, it requires $\frac{7}{6}kN \log_3 N$ additions and $\frac{1}{6}kN \log_3 N$ multiplications over $\text{GR}_2 := \mathbb{GR}(2^\ell, 2)$. Note that performing k multiplications over GR_2 is significantly cheaper than one multiplication over GR_k ³, in particular, for a common $k = 128$. This means that our FFT is much faster than typical quasi-linear time algorithms that perform $O(N \log N)$ operations in GR_k .

Fast linear codes over $\mathbb{F}_{2^k}$ and GR_k . Equipped with our new FFT algorithms, we put forward efficient encoding (as well as syndrome computing) algorithms for a family of *Quasi-Abelian* (QA) codes over GR_k . A (rank-1) QA code $\mathcal{C}_{a_1, \dots, a_c}$ of length cN and index c is defined as follows:

$$\mathcal{C}_{a_1, \dots, a_c} := \{b \cdot (a_1, \dots, a_c) \mid b \in \text{GR}_k[\mathbb{G}]\},$$

where $\text{GR}_k[\mathbb{G}]$ is a group ring defined by GR_k and an Abelian group $\mathbb{G}$ of order N , and $a_1, \dots, a_c \in \text{GR}_k[\mathbb{G}]$. In this paper, we particularly instantiate $\mathbb{G}$ with $\mathbb{Z}_3^n$, and thus $N = 3^n$. By our choice of $\mathbb{G}$, $\text{GR}_k[\mathbb{G}]$ is essentially the same as the polynomial ring $\mathcal{R} = \text{GR}_k[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$, which is isomorphic to GR_k^N . This is exactly the polynomial ring handled by our FFT algorithm: every element of $\mathcal{R}$ has a canonical representation with individual degree at most 2.

Now the encoding algorithm is conceptually simple: (i) first perform our FFT on $b, a_1, \dots, a_c$ and obtain the evaluations $\widehat{b}, \widehat{a}_1, \dots, \widehat{a}_c \in \text{GR}_k^N$; ⁴ (ii) second compute $\widehat{b} * \widehat{a}_1, \dots, \widehat{b} * \widehat{a}_c$ and then perform inverse FFT on them. The syndrome computing algorithm is similar, where the goal is to compute $a_1 e_1 + \dots + a_c e_c \in \text{GR}_k[\mathbb{G}]$ for $e_1, \dots, e_c \in \text{GR}_k[\mathbb{G}]$. We have the following results.

Theorem 1.2 (Informal). *There exists an algorithm that computes a code-word/syndrome of $\mathcal{C}_{a_1, \dots, a_c} \subseteq (\text{GR}_k[\mathbb{Z}_3^n])^c$ in performing $O(ckN \log N)$ operations over GR_2 and $(c-1)N$ multiplications in GR_k .*

Theorem 1.2 solves a common problem left in [43, 44], where they propose efficient encoding and syndrome computation algorithms for QA codes over arbitrary large fields except for $\mathbb{F}_{2^k}$.

We implement our algorithms over $\mathbb{F}_{2^k}$ and compare them to RAA codes in Table 1, which is the state-of-the-art code used in the literature. Concrete minimum distances of QA codes and RAA codes over $\mathbb{F}_{2^k}$ are derived from recent works [43, 44, 19]. Note that [19]’s analyses are only applicable to $r \leq 1/4$.

³ A school-book algorithm needs $O(k^2)$ multiplications over $\mathbb{Z}_{2^\ell}$, while a more involved FFT algorithm over general commutative rings has complexity $O(k \log k \log \log k)$ [56, Theorem 8.23].

⁴ $\widehat{a}_1, \dots, \widehat{a}_c$ can be pre-computed.

$N = 3^n$	k	Code	Rate r	Distance δ	Time (ms)	Failure Probability
3^{13}	128	QA	1/2	0.41	38	2^{-127}
			1/3	0.60	68	2^{-127}
			1/4	0.70	95	2^{-127}
			1/5	0.76	123	2^{-127}
		RAA	1/4	0.19	158	$2^{-25.4}$
			1/8	0.29	323	$2^{-86.2}$

Table 1: Encoding time of fast linear codes over $\mathbb{F}_{2^k}$.

PCGs over $\mathbb{F}_{2^k}$ and GR_k . Following the blueprint of [12, 11], we construct efficient PCGs for VOLE over $\mathbb{F}_{2^k}$ and GR_k from our QA code, under the hardness of Quasi-Abelian syndrome decoding (QA-SD) problems [5, 46].

Moreover, since our QA code exploits a polynomial ring structure as in previous works [14, 5], we can build quasi-linear time programmable PCGs for OLEs over $\mathbb{F}_{2^k}$ and GR_k . We emphasize that generally a fast linear code, e.g., RAA code, only admits *quadratic* time PCGs for OLE [12, 14], which is impractical for a typical $N \approx 2^{20}$. This is because a general linear code preserves only linear operations, whereas OLE requires multiplication; without additional multiplicative structure, all cross terms must be handled explicitly. Moreover, the univariate ring approach [14] is also incompatible with this scenario, because GR_k seems not to have $O(1)$ -smooth multiplicative subgroups that are as large as 2^{20} . The largest smooth group we are aware of only has size 3^5 for $k < 400$ [39].

We fully implement our PCGs for both VOLE and OLE in the libOTe library. Concretely, we achieve a throughput of $\approx 5,000,000$ VOLEs over $\mathbb{F}_{2^{128}}$ per second, which is at least 4.7x, 1.9x and 1.6x faster than PCGs from EA, BA and EC codes, respectively. For OLEs over $\mathbb{F}_{2^{128}}$, it is $\approx 42,000/s$, where the slowdown is mainly due to $O(\lambda N)$ PRG calls rather than $O(N)$ required in VOLE setting. We remark that recent advances in distributed multi-point functions (DMPF) [15, 1] can further reduce the number of PRG calls to $O(N)$, and we leave the integration to future work. Concrete numbers are referred to Table 6 and Table 7. The code implementation is available at <https://github.com/zhli271828/ExtensionFieldPCG>.

SNARKs and More Applications. Following Brakedown’s template [37], we can build an efficient polynomial commitment scheme (PCS) over $\mathbb{F}_{2^k}$ and GR_k via our QA code. By further incorporating with different polynomial IOPs (interactive oracle proofs) [53, 39, 58], we obtain transparent SNARKs over $\mathbb{F}_{2^k}$, $\mathbb{Z}_{2^\ell}$, and GR_k , respectively. Since our QA codes are concretely faster, and offer better minimum distance [44], our PCS not only has faster proving algorithms, but also has a smaller proof size and verification time. We note that our constructions over $\mathbb{F}_{2^k}$ perfectly complement [44] over $\mathbb{F}_{p^k}$ with $p > 2$, together implying efficient SNARKs over *arbitrary* large fields from QA codes. We leave concrete implementations as future work.

Additionally, our programmable PCGs over $\mathbb{F}_{2^k}$ and GR_k can be used for MPC preprocessing, especially for RMFE-based MPC protocols [23, 32]. These works, though designed for $\mathbb{F}_2$ or $\mathbb{Z}_{2^\ell}$, consume a large number of authenticated multiplication quintuples over $\mathbb{F}_{2^k}$ or GR_k . Before this work, they had to use OT (oblivious transfer) to generate those quintuples, which incurs a large communication overhead. By adapting [51]’ approach, we show how to efficiently generate those quintuples from our programmable PCGs, achieving $\approx 237(n-1)$ times improvements in communication for [32], where n is the number of parties. Finally, our PCGs for VOLE can be directly applied to [47] and improve their concrete efficiency, which builds efficient interactive zero-knowledge protocols over $\mathbb{Z}_{2^\ell}$ from VOLEs over GR_k .

1.2 Technical Overview

Limitations of Univariate Polynomials. A degree- N polynomial f can be represented either in the coefficient form $f = \sum_{i=0}^{N-1} f_i X^i$ or the point-value form $\{(\alpha_i, f(\alpha_i))\}_{i \in [0, N-1]}$. Fast Fourier Transform (FFT) [24] provides efficient transformation algorithms between these two representations, further supporting efficient polynomial multiplication, evaluation and interpolation. In the PCG framework [14], a single OLE correlation over a polynomial ring $R[X]/(g(X))$ is first established, and by Chinese Remainder Theorem (CRT) it is decomposed into multiple OLE instances over R . This “decomposition” essentially corresponds to evaluating a polynomial in $R[X]/(g(X))$ at some points in R , namely the roots of $g(X)$. An FFT algorithm is the key to accelerating this evaluation process from quadratic to quasi-linear time.

By choosing $g(X) = \prod_{i=0}^{N-1} (X - \alpha_i)$, where $\alpha_1, \dots, \alpha_{N-1}$ are distinct elements in R , the above framework [14] in fact works for any commutative ring R . However, naively evaluating at N points needs $\Omega(N^2)$ time, and performing classic FFT would introduce a stringent constraint on R . Unfortunately, for $R = \mathbb{F}_{2^k}$ or $R = \text{GR}_k = \mathbb{GR}(2^\ell, k)$ of this paper’s interest, existing FFT algorithms mostly fail. In more detail, they require R to have large $O(1)$ -smooth multiplicative subgroups. However, this is not the case for $\mathbb{F}_{2^k}$ and GR_k . To see this, we display the factorization of $2^k - 1$ for typical integers k in Table 2. In addition, we also show the largest smooth factor for $k \leq 400$ in Table 3, which indicates that $k = 162$ implies a practical FFT for polynomials of degree less than 3^5 . However, $3^5 = 243$ is too small to be used in the PCG context, where the number of (V)OLE instances N is usually as large as 2^{20} . Although there exists an FFT algorithm over general commutative rings [56, Theorem 8.23], it is impractical due to its $O(N \log N \log \log N)$ complexity and a large hidden constant ≈ 70 .

Remark 1.3. We note that for binary extension fields $\mathbb{F}_{2^k}$, there exist efficient *additive* FFTs [20, 34, 48, 49] in which the evaluation points consist of an affine subspace of $\mathbb{F}_{2^k}$ over $\mathbb{F}_2$ and $g(X)$ is a linearized polynomial. However, additive FFTs incur a large hidden constant and moreover, they are not compatible with GR_k , whose characteristic is 2^ℓ .

k	Factor of $2^k - 1$
16	$3 \times 5 \times 17 \times 257$
32	$3 \times 5 \times 17 \times 257 \times 65537$
64	$3 \times 5 \times 17 \times 257 \times 641 \times 65537 \times 6700417$
128	$3 \times 5 \times 17 \times 257 \times 641 \times 65537 \times 274177 \times 6700417 \times 67280421310721$
256	$3 \times 5 \times 17 \times 257 \times 641 \times 65537 \times 274177 \times 6700417$ $\times 67280421310721 \times 59649589127497217 \times 5704689200685129054721$

Table 2: The factoring of $2^k - 1$ for some positive integers k

p	Largest p -smooth factor	p	Largest p -smooth factor
3	$3^5 \mid 2^{162} - 1$	5	$5^3 \mid 2^{100} - 1$
7	$7^3 \mid 2^{147} - 1$	11	$11^3 \mid 2^{110} - 1$
13	$13^2 \mid 2^{156} - 1$	17	$17^2 \mid 2^{136} - 1$
19	$19^2 \mid 2^{342} - 1$	23	$23^2 \mid 2^{253} - 1$

Table 3: The largest p -smooth factor of $2^k - 1$ for $k \leq 400$.

Our Approach Based on Multivariate Polynomials. We overcome the above limitations by shifting to a multivariate polynomial ring $\mathcal{R} = \mathbb{F}_{2^k}[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^d - 1, \dots, \mathbf{x}_n^d - 1)$, where $d \mid 2^k - 1$ guarantees the isomorphism $\phi := \mathcal{R} \rightarrow \mathbb{F}_{2^k}^N$ for $N = d^n$. According to [5, Proposition 13], a standard multivariate FFT algorithm can evaluate this isomorphism with $O(N \log N)$ additions and multiplications over $\mathbb{F}_{2^k}$. However, when k is large, each multiplication in $\mathbb{F}_{2^k}$ requires at least $O(k \log k \log \log k)$ bit operations (generally with a large hidden constant). Despite its asymptotic quasi-linear complexity, this FFT algorithm remains concretely expensive for typical cryptographic parameters, e.g., $k = 128$.

To further reduce the computation complexity, we propose a non-trivial FFT algorithm to push almost all computation down to a small base field. For simplicity, let us consider $\mathcal{R} := \mathbb{F}_{2^k}[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$, where k is an even integer. Our core observation is that $\mathbf{x}^3 - 1$ splits completely over the finite field $\mathbb{F}_4$ with roots $1, \alpha, \alpha + 1$, where α is a generator of $\mathbb{F}_4$. Crucially, the extension field $\mathbb{F}_{2^k}$ contains $\mathbb{F}_4$ as a subfield. Therefore, the same roots $1, \alpha, \alpha + 1$ exist in $\mathbb{F}_{2^k}$, and evaluating any polynomial $f \in \mathcal{R}$ at all points in $\{1, \alpha, \alpha + 1\}^n$ induces a natural isomorphism $\phi : \mathcal{R} \rightarrow \mathbb{F}_{2^k}^N$ for $N = 3^n$. This isomorphism depends only on the existence of the subfield $\mathbb{F}_4$ and merely requires k to be even, allowing a wide choice of k , such as $k = 64, 128, 256$.

While the multivariate polynomials have coefficients in a large field $\mathbb{F}_{2^k}$, all evaluation points lie in the small subfield $\mathbb{F}_4$. We exploit this feature to decompose a single computation over $\mathbb{F}_{2^k}$ into $k/2$ computations over $\mathbb{F}_4$. Let $\{1 = \gamma^0, \gamma, \dots, \gamma^{k/2-1}\}$ denote the basis of $\mathbb{F}_{2^k}$ over $\mathbb{F}_4$. Then any element $a \in \mathbb{F}_{2^k}$ can be represented as $a = \sum_{j=0}^{k/2-1} a_j \gamma^j$, where each $a_j \in \mathbb{F}_4$. In this

case, any $f \in \mathcal{R}$ can be rewritten as $f = \sum_{j=0}^{k/2-1} f^{(j)} \cdot \gamma^j$, where every coefficient of $f^{(j)}$ lies in $\mathbb{F}_4$. The evaluation of f then reduces to evaluating each polynomial $f^{(j)} \in \mathcal{R}_2 := \mathbb{F}_4[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$ and merging these $k/2$ evaluations over $\mathbb{F}_4$ as the evaluation of f over $\mathbb{F}_{2^k}$.

Now we turn to the evaluation of a polynomial $f^{(j)} \in \mathcal{R}_2$. We apply a ternary transform, which refers to a quasi-linear time multidimensional FFT over $\mathcal{R}_2$ based on the three roots of $\mathbf{x}^3 - 1$ over $\mathbb{F}_4$, to enable an efficient multivariate polynomial evaluation. Here we briefly explain the main idea. The transformation from the coefficient vector $\mathbf{f}^{(j)}$ to the evaluation vector $\hat{\mathbf{f}}^{(j)}$ at all points $\{1, \alpha, \alpha + 1\}^n$ is given by the n -fold Kronecker power of a small 3×3 matrix A over $\mathbb{F}_4$:

$$\hat{\mathbf{f}}^{(j)} = A^{\otimes n} \cdot \mathbf{f}^{(j)}, \text{ where } A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha + 1 \\ 1 & \alpha + 1 & \alpha \end{pmatrix}$$

This tensor product admits a classic divide-and-conquer structure. For completeness, we describe the corresponding divide-and-conquer algorithm in [Section B](#). Similarly, the inverse FFT can be viewed as a matrix multiplication from $B^{\otimes n}$, where $B \times A = I_3$ and B is explicitly defined as

$$B := \begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha + 1 & \alpha \\ 1 & \alpha & \alpha + 1 \end{pmatrix}.$$

The above method can be generalized to ring $\mathcal{R}' := \mathbb{F}_{2^k}[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^d - 1, \dots, \mathbf{x}_n^d - 1)$ for any base field $\mathbb{F}_{2^s}$ satisfying $d \mid 2^s - 1$ and $s \mid k$.

This framework naturally extends to general Galois rings. Using Hensel's lemma ([Lemma 2.3](#)), we lift $\alpha \in \mathbb{F}_4$ to $\beta \in \mathbb{GR}(2^\ell, 2)$, yielding an analogous matrix A' for the ring $\mathbb{GR}_2 = \mathbb{GR}(2^\ell, 2)$. As Galois ring $\mathbb{GR}_k = \mathbb{GR}(2^\ell, k)$ can be viewed as an extension of $\mathbb{GR}_2$ for an even k , we can similarly apply the above decomposition technique. Consequently, we obtain an efficient multivariate FFT algorithm over $\mathbb{GR}_k$, which performs only $O(kN \log_3 N)$ operations within $\mathbb{GR}_2$.

2 Preliminaries

In this section, we focus on the notations, Galois rings and syndrome decoding of Quasi-Abelian code (QA-SD) assumption. The formal definitions of pseudo-random correlation generators (PCGs) and the two underlying cryptographic primitives, function secret sharing (FSS) and puncturable pseudorandom functions (PPRF), are deferred to [Section A.1](#) and [Section A.2](#) in the Supplementary Material.

2.1 Basic Notations

We use $\mathbb{F}_4$ to denote a finite field $\mathbb{F}_2[x]/(x^2 + x + 1) \in \mathbb{F}_2[x]$. Let α be the root of $x^2 + x + 1 \in \mathbb{F}_2[x]$, then we have that $\mathbb{F}_4 = \mathbb{F}_2(\alpha) = \{0, 1, \alpha, \alpha + 1\}$. In this work,

we mainly focus on the extension field $\mathbb{F}_{2^k}$, where the extension degree k is even. Given $\ell, r \in \mathbb{Z}$, we use $[\ell, r]$ to denote the index set $\{\ell, \ell+1, \dots, r\}$, as well as $[\ell, r) = \{\ell, \ell+1, \dots, r-1\}$ and $[r] = [1, r] = \{1, 2, \dots, r\}$. Divisibility is denoted as $a \mid b$, representing that a divides b . We use $x \stackrel{\$}{\leftarrow} S$ 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 $\mathcal{R} = \mathbb{F}_{2^k}[\mathbf{X}_1, \dots, \mathbf{X}_n]/(\mathbf{X}_1^3 - 1, \dots, \mathbf{X}_n^3 - 1)$ with n variables $\{\mathbf{X}_1, \dots, \mathbf{X}_n\}$. For a polynomial $f \in \mathcal{R}$, we use $\mathbf{f} \in \mathbb{F}_{2^k}^N$ to denote the coefficients of f in the lexicographic order of the monomials and $\hat{\mathbf{f}} \in \mathbb{F}_{2^k}^N$ to represent the evaluation of f over $\{1, \alpha, \alpha+1\}^n$, where $N = 3^n$. For a vector $\mathbf{e} \in \mathbb{F}_{2^k}^n$, we denote $\text{wt}(\mathbf{e})$ as its Hamming weight. Furthermore, for a polynomial f with n variables, we use $\text{wt}(f)$ to denote the Hamming weight of the corresponding coefficient vector $\text{wt}(\mathbf{f})$. A polynomial $f \in \mathcal{R}$ with $\text{wt}(f) = t$ is denoted as a t -sparse polynomial.

For two vectors $\mathbf{u} = (u_1, \dots, u_n), \mathbf{v} = (v_1, \dots, v_n) \in R^n$ for some ring R , we denote $\mathbf{u} \otimes \mathbf{v} \in R^{n^2}$ as their tensor product, i.e., $\mathbf{u} \otimes \mathbf{v} = (u_i \cdot v_j)_{i,j \in [n]} = (v_1 \cdot \mathbf{u}, \dots, v_n \cdot \mathbf{u})$. Similarly, we write $\mathbf{u} \boxplus \mathbf{v} \in R^{n^2}$ to denote the outer sum of two vectors, i.e., $\mathbf{u} \boxplus \mathbf{v} = (u_i + v_j)_{i,j \in [n]} = (v_1 + \mathbf{u}, \dots, v_n + \mathbf{u})$. We use $\mathbf{u} + \mathbf{v} \in R^n$ and $\mathbf{u} \star \mathbf{v} \in R^n$ 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 Galois Rings

We briefly introduce Galois rings, and refer to the lecture notes [57] for more details. Let p be a prime and $\mathbb{Z}_{p^\ell}$ be the integer residue classes modulo p^ℓ .

Definition 2.1 (Galois Ring). *The Galois ring is a ring extension of $\mathbb{Z}_{p^\ell}$ and has characteristic p^ℓ . Specifically, let $k \geq 1$ be an integer and $g(x) \in \mathbb{Z}_{p^\ell}[x]$ be a monic polynomial of degree k such that $\bar{g}(x) := g(x) \bmod p$ is an irreducible polynomial over $\mathbb{F}_p$. The Galois ring of extension degree k is defined as*

$$\mathbb{GR}(p^\ell, k) = \mathbb{Z}_{p^\ell}[x]/(g(x)).$$

Let $\zeta = x + (g(x)) \in \mathbb{GR}(p^\ell, k)$ be a root of $g(x)$. Then any $a \in \mathbb{GR}(p^\ell, k) = \mathbb{Z}_{p^\ell}[\zeta]$ can be expressed uniquely in the form of $a = a_0 + a_1\zeta + \dots + a_{k-1}\zeta^{k-1}$, $a_i \in \mathbb{Z}_{p^\ell}$.

Definition 2.2 (Monic Basic Primitive Polynomials). *Let $g(x)$ be a monic polynomial over $\mathbb{Z}_{p^\ell}$. We say $g(x) \in \mathbb{Z}_{p^\ell}[x]$ is a monic basic primitive, if $\bar{g}(x) := g(x) \bmod p$ is a monic primitive polynomial over $\mathbb{F}_p$, i.e., $\bar{g}(x)$ is irreducible and any of its roots generates the multiplication group of $\mathbb{F}_{p^k}$, where k is the degree of $\bar{g}$.*

Lemma 2.3 (Hensel's Lemma). [57, Lemma 13.6] *Let $f(x)$ be a monic polynomial in $\mathbb{Z}_{p^\ell}[x]$ and assume that $f = \bar{f}_1 \bar{f}_2$ in $\mathbb{F}_p[x]$, where $\bar{f} = f \bmod p$ and $\bar{f}_1, \bar{f}_2$ are coprime monic polynomials in $\mathbb{F}_p[x]$. Then there exist coprime monic polynomials f_1, f_2 in $\mathbb{Z}_{p^\ell}[x]$ such that*

$$f = f_1 f_2 \text{ in } \mathbb{Z}_{p^\ell}[x], \text{ with } \bar{f}_1 = f_1 \bmod p, \text{ and } \bar{f}_2 = f_2 \bmod p.$$

In this work, we focus on $p = 2$, and denote $\mathbb{GR}(2^\ell, k)$ by $\mathbb{GR}_k$. We crucially rely on the following ring isomorphism between $\mathcal{R} = \mathbb{GR}_k[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^d - 1, \dots, \mathbf{x}_n^d - 1)$ and $\mathbb{GR}_k^{d^n}$, where d divides $2^k - 1$. The interested reader can find the proof in [46, Lemma 4.4].

Lemma 2.4 (The Multivariate Ring Isomorphism [46]). *Let $\mathbb{GR}_k := \mathbb{GR}(2^\ell, k) = \mathbb{Z}_{2^\ell}(x)/(g(x))$, where $g(x)$ is the Hensel lift of a degree- k monic primitive polynomial over $\mathbb{F}_2$. Let $\mathcal{R} := \mathbb{GR}_k[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^d - 1, \dots, \mathbf{x}_n^d - 1)$ with $d \mid (2^k - 1)$. Then there exists $\phi : \mathcal{R} \rightarrow \mathbb{GR}_k^N$, where $N = d^n$, such that ϕ is a ring isomorphism.*

2.3 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 2.5 (Group Ring). *Let R be a ring, and $\mathbb{G}$ be a group. Define the set $R[\mathbb{G}] = \{\sum_{g \in \mathbb{G}} a_g g \mid a_g \in R, \text{ and there are finitely many } a_g \neq 0\}$. 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:*

$$\begin{aligned} \left(\sum_{g \in \mathbb{G}} a_g g\right) + \left(\sum_{g \in \mathbb{G}} b_g g\right) &= \sum_{g \in \mathbb{G}} (a_g + b_g)g, \\ \left(\sum_{g \in \mathbb{G}} a_g g\right) \cdot \left(\sum_{g \in \mathbb{G}} b_g g\right) &= \sum_{g \in \mathbb{G}} \left(\sum_{h \in \mathbb{G}} (a_h b_{h^{-1}g})\right)g. \end{aligned}$$

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

Quasi-Abelian codes. Let $\mathbb{GR}_k = \mathbb{GR}(2^\ell, k)$ be a Galois ring and $\mathbb{G}$ a finite abelian group. Let c be a positive integer, and consider the free $\mathbb{GR}_k[\mathbb{G}]$ -module of rank c :

$$(\mathbb{GR}_k[\mathbb{G}])^c := \mathbb{GR}_k[\mathbb{G}] \oplus \dots \oplus \mathbb{GR}_k[\mathbb{G}] = \{(a_1, \dots, a_c) \mid a_i \in \mathbb{GR}_k[\mathbb{G}]\}.$$

Any $\mathbb{GR}_k[\mathbb{G}]$ -submodule of $(\mathbb{GR}_k[\mathbb{G}])^c$ is called a quasi- $\mathbb{G}$ code of index c . If $\mathbb{G}$ is abelian, then a quasi- $\mathbb{G}$ code is quasi-abelian. Let $G \in (\mathbb{GR}_k[\mathbb{G}])^{r \times c}$ be a generating matrix of rate $0 < r < 1$. Then the quasi- $\mathbb{G}$ code given by G is defined as follows:

$$\mathfrak{C} := \{\mathbf{m} \cdot G \mid \mathbf{m} = (m_1, \dots, m_{cr}) \in (\mathbb{GR}_k[\mathbb{G}])^{cr}\}.$$

The quasi- $\mathbb{G}$ code $\mathfrak{C}$ of index c can be seen as a linear code of length $|\mathbb{G}| \times c$ over $\mathbb{GR}_k$, where the generating matrix is formed out of $cr \times c$ square blocks of size $|\mathbb{G}|$ (induced by G). Assume $|\mathbb{G}| = N$, fix an ordering $g_1, \dots, g_N$ of elements in $\mathbb{G}$ and define the isomorphism $\varphi : \mathbb{GR}_k[\mathbb{G}] \rightarrow \mathbb{GR}_k^N$. Let $\text{HW}_t(\mathbb{GR}_k[\mathbb{G}])$ denote the set $\{a \in \mathbb{GR}_k[\mathbb{G}] \mid \text{wt}(\varphi(a)) = t\}$, 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 $H \in (\mathbb{GR}_k[\mathbb{G}])^2$ of the form $H = (1 \mid a)$, where a is uniformly distributed in $\mathbb{GR}_k[\mathbb{G}]$.

Syndrome decoding of quasi-abelian codes. As in [5, 45], we consider QA-SD problem over the group ring $\text{GR}_k[\mathbb{Z}_d^n]$, which is isomorphic to $\text{GR}_k[x_1, \dots, x_n]/(x_1^d - 1, \dots, x_n^d - 1) \simeq \text{GR}_k^N$, where $d \mid 2^k - 1$, and $N = d^n$.

Definition 2.6 (Search QA-SD). Let $\mathcal{R} = \text{GR}_k[x_1, \dots, x_n]/(x_1^d - 1, \dots, x_n^d - 1)$, where $d \mid 2^k - 1$. Let $c \geq 2$ be some constant integer called the compression factor. Let $\mathbf{a} = (1, a_1, \dots, a_{c-1})$, where $a_i \xleftarrow{\$} \mathcal{R}$, $i \in [c-1]$. Let $e_0, e_1, \dots, e_{c-1}$ be random t -sparse elements of $\mathcal{R}$. Given access to a pair of the form $(\mathbf{a}, \langle \mathbf{a}, \mathbf{e} \rangle)$, the goal is to recover $\mathbf{e}$.

Definition 2.7 (Decisional QA-SD). Let $\mathcal{R} = \text{GR}_k[x_1, \dots, x_n]/(x_1^d - 1, \dots, x_n^d - 1)$, where $d \mid 2^k - 1$. Let $c \geq 2$ be some constant integer called the compression factor. The goal is to distinguish the following two distributions:

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

$\mathcal{D}_1: \{(a_1, \dots, a_{c-1}, \langle \mathbf{a}, \mathbf{e} \rangle + e_0)\}$, where $a_i \xleftarrow{\$} \mathcal{R}$, e_0, e_i are random t -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 extension field $\mathbb{F}_{2^k}$ or a Galois ring $\text{GR}_k = \mathbb{GR}(2^\ell, k)$, where k is an arbitrary even integer and set $d = 3$. Since $\mathcal{R}$ is essentially the same as $\text{GR}_k[\mathbb{Z}_d^n]$, we also abuse the notion QA-SD($c, \text{GR}_k[\mathbb{Z}_d^n], t$) as of QA-SD($c, \mathcal{R}, t$).

3 Fast Multivariate Transforms over Galois Rings

In this section, we focus on Galois rings $\mathbb{GR}(2^\ell, k)$ for even k . We start with the FFT algorithms over degree-2 Galois rings $\mathbb{GR}(2^\ell, 2)$ in Section 3.1. These algorithms are adapted from FFT algorithms over $\mathbb{F}_4$, which are presented in Section B for completeness. With these algorithms as a building block, we present a paradigm of FFT over Galois rings $\mathbb{GR}(2^\ell, k)$ in Section 3.2.

3.1 Fast Fourier Transform over Degree-2 Galois Rings

We first lift the ternary transform from $\mathbb{F}_4$ to the Galois ring $\text{GR}_2 = \mathbb{GR}(2^\ell, 2)$. Our recursive structure is adapted from the ternary transform over $\mathbb{F}_4$ in [4]. When $\ell = 1$, this ring coincides with $\mathbb{F}_4$. For $\ell > 1$, the characteristic becomes 2^ℓ , so the simplifications relying on the identity $x = -x$ over $\mathbb{F}_4$ no longer apply. Consequently, the resulting divide-and-conquer algorithm uses different evaluation points and incurs a slightly larger number of additions and subtractions.

To obtain efficient FFT and IFFT algorithms over GR_2 , the first goal is to define the evaluation points. As discussed in Section 1.2, the ternary transform

over $\mathbb{F}_4$ evaluates at the points $\{1, \alpha, \alpha + 1\}^n$, where α is the root of irreducible polynomial $x^2 + x + 1 \in \mathbb{F}_2[x]$. To obtain the analogous transform over GR_2 , we lift these points via Hensel's lifting (Lemma 2.3). Let $h(x) = x^2 + x + 1 \in \mathbb{Z}_{2^e}[x]$ and $\bar{h} = h \bmod 2$, where $\bar{h}$ is a primitive polynomial over $\mathbb{F}_2$ and it splits over $\mathbb{F}_4$ as $\bar{h} = (x - \alpha)(x - \alpha - 1)$. By Hensel lifting, the root α has a unique lift $\beta \in \text{GR}_2$ satisfying that $\beta \equiv \alpha \bmod 2$, $\beta^2 + \beta + 1 = 0$. Hence $\beta^2 = -\beta - 1$, and the three roots of $x^3 - 1 \in \text{GR}_2[x]$ are $\{1, \beta, -\beta - 1\}$. Let $\mathcal{R}_2 = \text{GR}_2[x_1, \dots, x_n]/(x_1^3 - 1, \dots, x_n^3 - 1)$ and $N = 3^n$. We use $\mathbf{f} \in \text{GR}_2^N$ to denote the coefficients of $f \in \mathcal{R}_2$. The evaluations $\hat{\mathbf{f}} \in \text{GR}_2^N$ of f at $\{1, \beta, -\beta - 1\}^n$ can be computed from Eq.(1):

$$\hat{\mathbf{f}} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \beta & \beta^2 \\ 1 & \beta^2 & \beta \end{pmatrix}^{\otimes n} \mathbf{f} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \beta & -\beta - 1 \\ 1 & -\beta - 1 & \beta \end{pmatrix}^{\otimes n} \mathbf{f} \quad (1)$$

Based on Eq.(1), we present FFT over GR_2 in Algorithm 1. To see the correctness, write the multivariate polynomial f as $f = f_L + f_M x_n + f_R x_n^2$. Algorithm 1 recursively evaluates f_L, f_M, f_R over the first $n - 1$ variables and combines the results for $x_n \in \{1, \beta, \beta^2\}$. Using $\beta^2 = -\beta - 1$ yields the evaluation matrix in Eq.(1) and formulas in Step 6 of Algorithm 1.

Algorithm 1 Fast Fourier Transform over GR_2 : FFT_{GR_2}

Input: Polynomial $f \in \mathcal{R}_2 = \text{GR}_2[x_1, \dots, x_n]/(x_1^3 - 1, \dots, x_n^3 - 1)$ with coefficient vector $\mathbf{f} = (f_0, \dots, f_{N-1}) \in \text{GR}_2^N$, where $N = 3^n$

Output: Evaluation vector $\hat{\mathbf{f}} \in \text{GR}_2^N$ of f at all points in $\{1, \beta, -\beta - 1\}^n$

- 1: **if** $n = 0$ **then**
- 2: Return $\mathbf{f}$
- 3: **else**
- 4: Split $\mathbf{f}$ into three vectors of length $m = 3^{n-1}$:

$$\begin{aligned} \mathbf{f}_L &= (f_0, f_1, \dots, f_{m-1}) \\ \mathbf{f}_M &= (f_m, f_{m+1}, \dots, f_{2m-1}) \\ \mathbf{f}_R &= (f_{2m}, f_{2m+1}, \dots, f_{3m-1}) \end{aligned}$$

- 5: Compute $\hat{\mathbf{f}}_L = \text{FFT}_{\text{GR}_2}(\mathbf{f}_L)$, $\hat{\mathbf{f}}_M = \text{FFT}_{\text{GR}_2}(\mathbf{f}_M)$ and $\hat{\mathbf{f}}_R = \text{FFT}_{\text{GR}_2}(\mathbf{f}_R)$
- 6: Compute:

$$\begin{aligned} \mathbf{y}_1 &= \hat{\mathbf{f}}_L + \hat{\mathbf{f}}_M + \hat{\mathbf{f}}_R \\ \mathbf{y}_2 &= \hat{\mathbf{f}}_L - \hat{\mathbf{f}}_R + (\hat{\mathbf{f}}_M - \hat{\mathbf{f}}_R)\beta \\ \mathbf{y}_3 &= \hat{\mathbf{f}}_L - \hat{\mathbf{f}}_M - (\hat{\mathbf{f}}_M - \hat{\mathbf{f}}_R)\beta \end{aligned}$$

- 7: Return $\hat{\mathbf{f}} = (\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3)$
 - 8: **end if**
-

Theorem 3.1. *Algorithm 1 requires $\frac{7}{3}N \log_3 N$ additions and $\frac{1}{3}N \log_3 N$ multiplications over GR_2 .*

Proof. Let T_n^+ and $T_n^\times$ denote the number of additions/subtractions and multiplications to compute $\text{FFT}_{\text{GR}_2}(f)$ for $f \in \mathcal{R}_2$. From the description of Algorithm 1, we observe that we need to compute 7 additions and 1 multiplication over $\text{GR}_2^{3^{n-1}}$ in Step 6:

$$\begin{aligned} \mathbf{t}_1 &:= \widehat{\mathbf{f}}_L + \widehat{\mathbf{f}}_M & \mathbf{y}_1 &:= \mathbf{t}_1 + \widehat{\mathbf{f}}_R & \mathbf{t}_2 &:= \widehat{\mathbf{f}}_M - \widehat{\mathbf{f}}_R & \mathbf{t}_3 &:= \mathbf{t}_2 \cdot \beta \\ \mathbf{t}_4 &:= \widehat{\mathbf{f}}_L + \mathbf{t}_3 & \mathbf{t}_5 &:= \widehat{\mathbf{f}}_L - \mathbf{t}_3 & \mathbf{y}_2 &:= \mathbf{t}_4 + \widehat{\mathbf{f}}_R & \mathbf{y}_3 &:= \mathbf{t}_5 - \widehat{\mathbf{f}}_M \end{aligned}$$

Then we have that $T_n^+ = 3T_{n-1}^+ + 7 \cdot 3^{n-1}$ and $T_n^\times = 3T_{n-1}^\times + 3^{n-1}$. By applying this result recursively, we conclude that $T_n^+ = \frac{7}{3} \cdot n \cdot 3^n = \frac{7}{3}N \log_3 N$ and $T_n^\times = n \cdot 3^{n-1} = \frac{1}{3}N \log_3 N$. $\square$

To compute the coefficients $\mathbf{f}$ from $\widehat{\mathbf{f}}$, we apply equation Eq.(2):

$$\mathbf{f} = 3^{-n} \begin{pmatrix} 1 & 1 & 1 \\ 1 & \beta^2 & \beta \\ 1 & \beta & \beta^2 \end{pmatrix}^{\otimes n} \widehat{\mathbf{f}} = 3^{-n} \begin{pmatrix} 1 & 1 & 1 \\ 1 - \beta - 1 & \beta & \\ 1 & \beta & -\beta - 1 \end{pmatrix}^{\otimes n} \widehat{\mathbf{f}} \quad (2)$$

Note that in (2), we multiply a scalar 3^{-n} . This is due to

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & \beta & -\beta - 1 \\ 1 - \beta - 1 & \beta & \end{pmatrix}^{\otimes n} \begin{pmatrix} 1 & 1 & 1 \\ 1 - \beta - 1 & \beta & \\ 1 & \beta & -\beta - 1 \end{pmatrix}^{\otimes n} = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{pmatrix}^{\otimes n} = 3^n \cdot I_{3^n}$$

To reduce the number of multiplications, we present an unnormalized IFFT algorithm over $\text{GR}_2 = \mathbb{GR}(2^\ell, 2)$ that outputs $3^n \cdot f$ instead of f , which is deferred to Algorithm 7 in Section B in the Supplementary Material.

3.2 Fast Fourier Transform over Galois Rings

With a ternary transform over the base ring $\text{GR}_2 = \mathbb{GR}(2^\ell, 2)$ at hand, we now use it to evaluate multivariate polynomials over a larger ring $\text{GR}_k = \mathbb{GR}(2^\ell, k)$ for an even integer k . The key point is that GR_k is a free GR_2 -module of rank $k/2$. For a multivariate polynomial $f \in \mathcal{R} = \text{GR}_k[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$, we denote its coefficient representation as

$$f = \sum_{(i_1, \dots, i_n) \in \{0, 1, 2\}^n} f_{i_1, \dots, i_n} \mathbf{x}_1^{i_1} \cdots \mathbf{x}_n^{i_n}$$

where $f_{i_1, \dots, i_n} \in \text{GR}_k$.

For simplicity, we map each n -dimensional multi-index $\mathbf{i} = (i_1, \dots, i_n) \in \{0, 1, 2\}^n$ into an integer $\phi(\mathbf{i}) = \sum_{j \in [n]} i_j \cdot 3^{j-1}$. Conversely, ϕ^{-1} gives the unique ternary decomposition of an integer $i \in [0, N)$. In this case, we can equivalently

represent f as $f = \sum_{i \in [0, N-1]} f_i \cdot \mathbf{X}^i$, where $\phi^{-1}(i) = (i_1, \dots, i_n)$, $f_i = f_{i_1, \dots, i_n}$ and $\mathbf{X}^i = \mathbf{x}_1^{i_1} \cdots \mathbf{x}_n^{i_n}$.

Since GR_k is a degree- $(k/2)$ Galois ring extension of $\text{GR}_2 = \mathbb{GR}(2^\ell, 2)$, we can rewrite each $a \in \text{GR}_k$ as $a = a_0 + a_1\gamma + \cdots + a_{k/2-1}\gamma^{k/2-1}$, where $\{1 = \gamma^0, \gamma, \dots, \gamma^{k/2-1}\}$ forms a basis of GR_k over GR_2 . For any $f \in \mathcal{R}$, we have that

$$\begin{aligned} f &= \sum_{i \in [0, N-1]} f_i \cdot \mathbf{X}^i \\ &= \sum_{i \in [0, N-1]} \left(\sum_{j \in [0, k/2-1]} f_{i,j} \gamma^j \right) \mathbf{X}^i \\ &= \sum_{j \in [0, k/2-1]} \left(\sum_{i \in [0, N-1]} f_{i,j} \cdot \mathbf{X}^i \right) \gamma^j \end{aligned}$$

Let $f^{(j)} \in \text{GR}_2[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3-1, \dots, \mathbf{x}_n^3-1)$ be defined as $f^{(j)} = \sum_{i=0}^{N-1} f_{i,j} \mathbf{X}^i$. Since all evaluation points belong to GR_2 , we can obtain all evaluations of $f^{(j)}$ by applying FFT for multivariate polynomials over GR_2 . Then we combine the evaluation of these sub-polynomials to obtain the final results. The concrete fast evaluation algorithm is summarized as [Algorithm 2](#) as follows.

Algorithm 2 Fast Fourier Transform over GR_k : FFT_{GR_k}

Input: Polynomial $f \in \mathcal{R} = \text{GR}_k[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3-1, \dots, \mathbf{x}_n^3-1)$ with coefficient vector $\mathbf{f} = (f_0, \dots, f_{N-1}) \in \text{GR}_k^N$, where $N = 3^n$.

Output: Evaluation vector $\hat{\mathbf{f}} \in \text{GR}_k^N$ of f at all points in $\{1, \beta, -\beta-1\}^n$

- 1: **for** $i \in [0, N-1]$ **do**
 - 2: Decompose coefficient f_i as $\sum_{j \in [0, k/2-1]} f_{i,j} \gamma^j$, where $\{1 = \gamma^0, \gamma, \dots, \gamma^{k/2-1}\}$ forms a basis of GR_k over $\text{GR}_2 = \mathbb{GR}(2^\ell, 2)$.
 - 3: **end for**
 - 4: **for** $j \in [0, k/2-1]$ **do**
 - 5: Define polynomial $f^{(j)} \in \text{GR}_2[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3-1, \dots, \mathbf{x}_n^3-1)$ with coefficient vector $(f_{0,j}, \dots, f_{N-1,j})$.
 - 6: Compute $\hat{\mathbf{f}}^{(j)} = \text{FFT}_{\text{GR}_2}(\mathbf{f}^{(j)})$.
 - 7: **end for**
 - 8: Return $\hat{\mathbf{f}} = \sum_{j \in [0, k/2-1]} \hat{\mathbf{f}}^{(j)} \cdot \gamma^j$
-

Theorem 3.2 (Theorem 1.1). When $\ell = 1$, [Algorithm 2](#) requires $\frac{5}{6}kN \log_3 N$ additions and $\frac{1}{6}kN \log_3 N$ multiplications over $\mathbb{F}_4$. Otherwise, [Algorithm 2](#) requires $\frac{7}{6}kN \log_3 N$ additions and $\frac{1}{6}kN \log_3 N$ multiplications over $\text{GR}_2 = \mathbb{GR}(2^\ell, 2)$.

Proof. Assume that the coefficients of input in $\mathcal{R}$ are represented in the GR_2 -basis $\{1 = \gamma^0, \gamma, \dots, \gamma^{k/2-1}\}$, and the output evaluations are represented in

the same basis. Under this assumption, we observe that the computation is dominant by $k/2$ invocations of FFT_{GR_2} in Step 6, while other steps only involve data rearrangement. The proof is completed with [Theorem B.1](#) and [Theorem 3.1](#). $\square$

By replacing FFT_{GR_2} with $\text{IFFT}_{\text{GR}_2}$ for $\text{GR}_2 = \mathbb{GR}(2^\ell, 2)$, we can obtain the IFFT algorithm over GR_k , which is deferred to [Algorithm 8](#) in [Section B](#) in the Supplementary Material. Note that we normalize the result by multiplying 3^{-n} in the last step.

4 PCGs over Binary Extension Fields

In this section, we show how to utilize the algorithms presented in [Section 3](#) to build efficient PCGs. While the FFT algorithms are developed over general Galois rings $\text{GR}(2^\ell, k)$, we present the main PCG constructions over binary extension field $\mathbb{F}_{2^k}$, corresponding to the case $\ell = 1$. This is the primary implementation target in this work and enables a direct comparison with prior PCG constructions. The constructions can naturally extend to general Galois rings and we state the corresponding complexity in [Corollary 4.2](#) and [Corollary 4.4](#).

Algorithm 3 Computing syndromes of QA codes

Input: The input message $e_1, \dots, e_c \in \text{GR}_k[\mathbb{Z}_3^n] \simeq \text{GR}_k[x_1, \dots, x_n]/(x_1^3 - 1, \dots, x_n^3 - 1)$, and evaluations of $a_1, \dots, a_c$ at $\{1, \beta, -\beta - 1\}^n$, denoted as $\hat{\mathbf{a}}_1, \dots, \hat{\mathbf{a}}_c$, where $\{1, \beta, -\beta - 1\}$ are the roots of $x^3 - 1$ over $\text{GR}_2 = \mathbb{GR}(2^\ell, 2)$.

Output: A syndrome $a_1 e_1 + \dots + a_c e_c \in \text{GR}_k[\mathbb{Z}_3^n]$.

- 1: For $i \in [c]$, compute $\hat{\mathbf{e}}_i = \text{FFT}_{\text{GR}_k}(e_i)$.
 - 2: Compute $\hat{\mathbf{b}} = \sum_{i=1}^c \hat{\mathbf{a}}_i \star \hat{\mathbf{e}}_i \in \text{GR}_k^N$, where $N = 3^n$.
 - 3: Compute $b = a_1 e_1 + \dots + a_c e_c = \text{IFFT}_{\text{GR}_k}(\hat{\mathbf{b}})$.
-

4.1 PCG for VOLE

In VOLE, two parties (P_0 and P_1) hold additive sharings of $\mathbf{x} \cdot \Delta$, where $\mathbf{x}$ is a length- N vector held by P_0 and Δ is a scalar held by P_1 . We first construct PCGs for VOLE over an extension field $\mathbb{F}_{2^k}$ for any even $k \in \mathbb{N}$ under QA-SD assumptions ([Definition 2.7](#)). The high-level idea is as follows:

1. The two parties are given short seeds that specifies additive sharings of $s \cdot \Delta, e \cdot \Delta$, denoted by $[s \cdot \Delta], [e \cdot \Delta]$, where s, e held by P_0 are t -sparse elements in $\mathbb{F}_{2^k}[\mathbb{Z}_3^n]$, and $\Delta \in \mathbb{F}_{2^k}$ is held by P_1 . This procedure can be achieved by PPRFs in a standard way [\[9, 11, 12\]](#).
2. Let $a \in \mathbb{F}_{2^k}[\mathbb{Z}_3^n]$ be a random public value. By linearity of additive secret sharing, the two parties can compute

$$[(a \cdot s + e) \cdot \Delta] \leftarrow a \cdot [s \cdot \Delta] + [e \cdot \Delta].$$

By QA-SD assumptions, $a \cdot s + e \in \mathbb{F}_{2^k}[\mathbb{Z}_3^n]$ is computationally indistinguishable from random.

3. Output $\left[\text{FFT}_{\mathbb{F}_{2^k}}((a \cdot s + e) \cdot \Delta) \right]$, which is a VOLE correlation over $\mathbb{F}_{2^k}$ of length $N = 3^n$ as $\mathbb{F}_{2^k}[\mathbb{Z}_3^n] \simeq \mathbb{F}_{2^k}^N$.

Note that $\mathbb{F}_{2^k}[\mathbb{Z}_3^n]$ is essentially the same as $\mathbb{F}_{2^k}[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$. Thus we can use [Algorithm 3](#) to compute $[(a \cdot s + e) \cdot \Delta]$. Moreover, as we are interested in VOLE over $\mathbb{F}_{2^k}$ and $\text{FFT}_{\mathbb{F}_{2^k}} \circ \text{IFFT}_{\mathbb{F}_{2^k}}$ is an identity map, it suffices to end up with Step 2 in [Algorithm 3](#). We give a self-contained PCG construction in [Cons_{VOLE} ^{\$\mathbb{F}_{2^k}\$}](#) .

Construction 1: Cons_{VOLE} ^{$\mathbb{F}_{2^k}$}

PARAMETER: Security parameter λ , Hamming weight $t = t(\lambda)$, $N = 3^n$, $\mathcal{R} = \mathbb{F}_{2^k}[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$. A PPRF scheme (PPRF.Gen, PPRF.Puncture, PPRF.Eval) for sums of t point functions, with domain $[0, N]^a$ and range $\mathbb{F}_{2^k}$.

PUBLIC INPUT: A uniformly random $a \in \mathcal{R}$ and its evaluation $\hat{\mathbf{a}} = \text{FFT}_{\mathbb{F}_{2^k}}(a)$.

CORRELATION: After expansion, outputs $(\mathbf{x}, \mathbf{z}_0) \in \mathbb{F}_{2^k}^N \times \mathbb{F}_{2^k}^N$ and $(\Delta, \mathbf{z}_1) \in \mathbb{F}_{2^k}^N \times \mathbb{F}_{2^k}^N$ such that $\Delta \cdot \mathbf{x} = \mathbf{z}_0 + \mathbf{z}_1$.

Gen: On input 1^λ :

1. For $\sigma = 0$, sample random vectors $\mathbf{A} = (a_1, \dots, a_t), \mathbf{B} = (b_1, \dots, b_t) \leftarrow [0, N]^t$ and $\mathbf{s} = (s_1, \dots, s_t), \mathbf{e} = (e_1, \dots, e_t) \leftarrow \mathbb{F}_{2^k}^t$.
2. For $\sigma = 1$, sample $\Delta \in \mathbb{F}_{2^k}$.
3. Sample two PPRF keys $\mathbf{k}_s, \mathbf{k}_e \leftarrow \text{PPRF.Gen}(1^\lambda)$, and compute two punctured keys $\mathbf{k}_s^{\mathbf{A}} \leftarrow \text{PPRF.Puncture}(\mathbf{k}_s, \mathbf{A})$ and $\mathbf{k}_e^{\mathbf{B}} \leftarrow \text{PPRF.Puncture}(\mathbf{k}_e, \mathbf{B})$.
4. For $i \in [t]$, let $s'_i = \Delta \cdot s_i - \text{PPRF.Eval}(\mathbf{k}_s, a_i)$ and $e'_i = \Delta \cdot e_i - \text{PPRF.Eval}(\mathbf{k}_e, b_i)$.
5. Let $\mathbf{k}_0 = ((\mathbf{k}_s^{\mathbf{A}}, \mathbf{A}, ([s'_i]_{i \in [t]}), (\mathbf{k}_e^{\mathbf{B}}, \mathbf{B}, ([e'_i]_{i \in [t]})))$ 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 $((\mathbf{k}_s^{\mathbf{A}}, \mathbf{A}, ([s'_i]_{i \in [t]}), (\mathbf{k}_e^{\mathbf{B}}, \mathbf{B}, ([e'_i]_{i \in [t]})))$.
 - For $j \in [0, N)$, define $\mathbf{v} \in \mathbb{F}_{2^k}^N$ as $\mathbf{v}[j] = s'_i$ if $j \in \mathbf{A}$ and $\mathbf{v}[j] = -\text{PPRF.Eval}(\mathbf{k}_s^{\mathbf{A}}, j)$ otherwise, and define $\mathbf{u} \in \mathbb{F}_{2^k}^N$ as $\mathbf{u}[j] = e'_i$ if $j \in \mathbf{B}$ and $\mathbf{u}[j] = -\text{PPRF.Eval}(\mathbf{k}_e^{\mathbf{B}}, j)$ 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 corresponding polynomials u and v , respectively.

- Compute $\widehat{\mathbf{v}} = \text{FFT}_{\mathbb{F}_{2^k}}(v)$, $\widehat{\mathbf{u}} = \text{FFT}_{\mathbb{F}_{2^k}}(u)$, and $\widehat{\mathbf{s}} = \text{FFT}_{\mathbb{F}_{2^k}}(s)$,
 $\widehat{\mathbf{e}} = \text{FFT}_{\mathbb{F}_{2^k}}(e)$.
- Let $\mathbf{x} = \widehat{\mathbf{a}} \star \widehat{\mathbf{s}} + \widehat{\mathbf{e}}$ and $\mathbf{z}_0 = \widehat{\mathbf{a}} \star \widehat{\mathbf{v}} + \widehat{\mathbf{u}}$
- Output $(\mathbf{x}, \mathbf{z}_0)$.
- 2. For $\sigma = 1$, parse $\mathbf{k}_1$ as $(\mathbf{k}_s, \mathbf{k}_e, \Delta)$.
 - Compute $\mathbf{w}_0 \leftarrow \text{PPRF.FullEval}(\mathbf{k}_s)$ and $\mathbf{w}_1 \leftarrow \text{PPRF.FullEval}(\mathbf{k}_e)$
 in $\mathbb{F}_{2^k}^N$, and interpret them as the corresponding polynomials w_0
 and w_1 , respectively.
 - Compute $\widehat{\mathbf{w}}_0 = \text{FFT}_{\mathbb{F}_{2^k}}(w_0)$, $\widehat{\mathbf{w}}_1 = \text{FFT}_{\mathbb{F}_{2^k}}(w_1)$ and let $\mathbf{z}_1 =$
 $\widehat{\mathbf{a}} \star \widehat{\mathbf{w}}_0 + \widehat{\mathbf{w}}_1$.
 - Output $(\Delta, \mathbf{z}_1)$.

^a Each j corresponds to a basis $\prod_{i \in [n]} \mathbf{x}_i^{j_i}$ of $\mathcal{R}$, where $j_i \in \{0, 1, 2\}$.

Theorem 4.1. Assume a secure PPRF scheme for sums of point functions and $QA\text{-}SD(\mathbb{F}_{2^k}, t)$ is hard. Then there exists a PCG construction $\text{Cons}_{\text{VOLE}}^{\mathbb{F}_{2^k}}$ that generates VOLE correlations over $\mathbb{F}_{2^k}$. If the PPRF is based on a $\text{PRG} : \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda}$ via the PRG-based construction from [11], for generating $N = 3^n$ VOLE correlations over $\mathbb{F}_{2^k}$, we have that:

- For party with $\sigma = 0$, the seed size is at most: $2t(\lambda \lceil \log_2 \frac{N}{t} \rceil + \lceil \log_2 N \rceil + k)$ bits.
- For party with $\sigma = 1$, the seed size is at most: $2t(\lambda + k)$ bits.
- For party with $\sigma = 0$, the computation of **Expand** can be done with at most $(4 + \lceil \frac{k}{\lambda} \rceil)N$ PRG operations, $\frac{10}{3}kN \log_3 N$ additions and $\frac{2}{3}kN \log_3 N$ multiplications over $\mathbb{F}_4$, and $2N$ additions and $2N$ multiplications over $\mathbb{F}_{2^k}$.
- For party with $\sigma = 1$, the computation of **Expand** can be done with at most $(4 + \lceil \frac{k}{\lambda} \rceil)N$ PRG operations, $\frac{5}{3}kN \log_3 N$ additions and $\frac{1}{3}kN \log_3 N$ multiplications over $\mathbb{F}_4$, and N additions and N multiplications over $\mathbb{F}_{2^k}$.

Proof. We defer the proof to Section C.1 in the Supplementary Material. $\square$

Corollary 4.2. Construction $\text{Cons}_{\text{VOLE}}^{\mathbb{F}_{2^k}}$ could be generalized to a Galois ring $\text{GR}_k = \mathbb{GR}(2^\ell, k)$ for any even integer k . To generate $N = 3^n$ VOLE correlations over GR_k , we have that:

- For party with $\sigma = 0$, the seed size is at most: $2t(\lambda \lceil \log_2 \frac{N}{t} \rceil + \lceil \log_2 N \rceil + k\ell)$ bits.
- For party with $\sigma = 1$, the seed size is at most: $2t(\lambda + k\ell)$ bits.
- For party with $\sigma = 0$, the computation of **Expand** can be done with at most $(4 + \lceil \frac{k\ell}{\lambda} \rceil)N$ PRG operations, $\frac{14}{3}kN \log_3 N$ additions and $\frac{2}{3}kN \log_3 N$ multiplications over $\text{GR}_2 = \mathbb{GR}(2^\ell, 2)$, and $2N$ additions and $2N$ multiplications over GR_k .
- For party with $\sigma = 1$, the computation of **Expand** can be done with at most $(4 + \lceil \frac{k\ell}{\lambda} \rceil)N$ PRG operations, $\frac{7}{3}kN \log_3 N$ additions and $\frac{1}{3}kN \log_3 N$ multiplications over GR_2 , and N additions and N multiplications over GR_k .

4.2 PCG for OLE

In OLE, two parties (P_0 and P_1) hold additive sharings of $\mathbf{x}_0 * \mathbf{x}_1$, where $\mathbf{x}_\sigma$ is a length- N vector held by P_σ for $\sigma \in \{0, 1\}$. We construct PCGs for OLE over an extension field $\mathbb{F}_{2^k}$ for any even $k \in \mathbb{N}$ under QA-SD assumptions. The high-level idea is similar to [Section 4.1](#), and given as follows.

1. The two parties are given short seeds that specify additive sharings of $s_0s_1, s_0e_1, e_0s_1, e_0e_1$, where s_σ, e_σ held by P_σ are t -sparse elements in $\mathbb{F}_{2^k}[\mathbb{Z}_3^n]$, for $\sigma \in \{0, 1\}$. This procedure can be achieved by DPFs in a standard way [\[12, 14, 5\]](#). Here the indices are directly represented as vectors in $\mathbb{Z}_3^n$, and their additions are preformed component-wise modulo 3, without binary decomposition or carry propagation.
2. Let $a \in \mathbb{F}_{2^k}[\mathbb{Z}_3^n]$ be a random public value. By linearity of additive secret sharing and [Algorithm 3](#), the two parties can compute

$$[(a \cdot s_0 + e_0) \cdot (a \cdot s_1 + e_1)] \leftarrow a^2 \cdot [s_0s_1] + a \cdot ([s_0e_1] + [s_1e_0]) + [e_0e_1].$$

By QA-SD assumption, $(a \cdot s_\sigma + e_\sigma) \in \mathbb{F}_{2^k}[\mathbb{Z}_3^n]$ is computationally indistinguishable from random, for $\sigma \in \{0, 1\}$.

3. Output $\left[\text{FFT}_{\mathbb{F}_{2^k}}((a \cdot s_0 + e_0) \cdot (a \cdot s_1 + e_1)) \right]$, which is an OLE correlation over $\mathbb{F}_{2^k}$ of length $N = 3^n$ as $\mathbb{F}_{2^k}[\mathbb{Z}_3^n] \simeq \mathbb{F}_{2^k}^N$.

Similarly, we can omit performing $\text{IFFT}_{\mathbb{F}_{2^k}}$ in [Algorithm 3](#) and the above Step 3.

We present a self-contained PCG construction in [Cons_{OLE} ^{\$\mathbb{F}_{2^k}\$}](#) . Besides the extension field $\mathbb{F}_{2^k}$, we can apply the trace function over a multivariate polynomial ring [\[46\]](#) to construct PCG for OLE over an integer ring $\mathbb{Z}_{2^\ell}$, which is deferred to [Section D](#) in the Supplementary Material.

Construction 2: Cons_{OLE} ^{$\mathbb{F}_{2^k}$}

PARAMETER: Security parameter λ , Hamming weight $t = t(\lambda)$, $N = 3^n$, $\mathcal{R} = \mathbb{F}_{2^k}[\mathbf{x}_1, \dots, \mathbf{x}_n] / (\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$. An FSS scheme ($\text{SPFSS.Gen}, \text{SPFSS.FullEval}$) for sums of t^2 point functions, with domain $[0, N]^a$ and range $\mathbb{F}_{2^k}$.

PUBLIC INPUT: A uniformly random $a \in \mathcal{R}$ and its evaluation $\hat{\mathbf{a}} = \text{FFT}_{\mathbb{F}_{2^k}}(a)$, $\hat{\mathbf{a}} \star \hat{\mathbf{a}}$.

CORRELATION: After expansion, outputs $(\mathbf{x}_0, \mathbf{z}_0) \in \mathbb{F}_{2^k}^N \times \mathbb{F}_{2^k}^N$ and $(\mathbf{x}_1, \mathbf{z}_1) \in \mathbb{F}_{2^k}^N \times \mathbb{F}_{2^k}^N$ such that $\mathbf{x}_0 \star \mathbf{x}_1 = \mathbf{z}_0 + \mathbf{z}_1$.

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 $\mathbf{s}_\sigma, \mathbf{e}_\sigma \leftarrow \mathbb{F}_{2^k}^t$.
2. Sample FSS keys:

$$\begin{aligned} (K_0^0, K_1^0) &\stackrel{\$}{\leftarrow} \text{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0 \boxplus \mathbf{A}_1^0, \mathbf{s}_0 \otimes \mathbf{s}_1), \\ (K_0^1, K_1^1) &\stackrel{\$}{\leftarrow} \text{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1 \boxplus \mathbf{A}_1^0, \mathbf{e}_0 \otimes \mathbf{s}_1), \\ (K_0^2, K_1^2) &\stackrel{\$}{\leftarrow} \text{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0 \boxplus \mathbf{A}_1^1, \mathbf{s}_0 \otimes \mathbf{e}_1), \\ (K_0^3, K_1^3) &\stackrel{\$}{\leftarrow} \text{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1 \boxplus \mathbf{A}_1^1, \mathbf{e}_0 \otimes \mathbf{e}_1), \end{aligned}$$

3. For $\sigma \in \{0, 1\}$, let $\mathbf{k}_\sigma = \left((K_\sigma^i)_{i \in [0, 3]}, (\mathbf{A}_\sigma^0, \mathbf{s}_\sigma), (\mathbf{A}_\sigma^1, \mathbf{e}_\sigma) \right)$.
4. Output $(\mathbf{k}_0, \mathbf{k}_1)$.

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

1. Parse $\mathbf{k}_\sigma$ as $\left((K_\sigma^i)_{i \in [0, 3]}, (\mathbf{A}_\sigma^0, \mathbf{s}_\sigma), (\mathbf{A}_\sigma^1, \mathbf{e}_\sigma) \right)$.
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 $u_{\sigma, i} \leftarrow \text{SPFSS.FullEval}(\sigma, K_\sigma^i)$, viewed as $\mathcal{R}$ elements.
4. Compute $\hat{\mathbf{u}}_{\sigma, i} = \text{FFT}_{\mathbb{F}_{2^k}}(u_{\sigma, i})$ for $i \in [0, 3]$, $\hat{\mathbf{s}}_\sigma = \text{FFT}_{\mathbb{F}_{2^k}}(s_\sigma)$ and $\hat{\mathbf{e}}_\sigma = \text{FFT}_{\mathbb{F}_{2^k}}(e_\sigma)$.
5. Compute

$$\mathbf{z}_\sigma = \hat{\mathbf{a}} \star \hat{\mathbf{a}} \star \hat{\mathbf{u}}_{\sigma, 0} + \hat{\mathbf{a}} \star (\hat{\mathbf{u}}_{\sigma, 1} + \hat{\mathbf{u}}_{\sigma, 2}) + \hat{\mathbf{u}}_{\sigma, 3}, \quad \mathbf{x}_\sigma = \hat{\mathbf{a}} \star \hat{\mathbf{s}}_\sigma + \hat{\mathbf{e}}_\sigma$$

6. Output $(\mathbf{x}_\sigma, \mathbf{z}_\sigma)$.

^a Each j corresponds to a basis $\prod_{i \in [n]} \mathbf{x}_i^{j_i}$ of $\mathcal{R}$, where $j_i \in \{0, 1, 2\}$. The addition on $[0, N)$ is actually the addition on $\mathbb{G} = \mathbb{Z}_3^n$.

Theorem 4.3. Assume a secure FSS scheme SPFSS for sums of points functions and $\text{QA-SD}(\mathbb{F}_{2^k}, t)$ is hard. Then there exists a PCG construction $\text{Cons}_{\text{OLE}}^{\mathbb{F}_{2^k}}$ that generates OLE correlations over $\mathbb{F}_{2^k}$. If the SPFSS is based on a PRG : $\{0, 1\}^\lambda \rightarrow \{0, 1\}^{3(\lambda+1)}$ via the PRG-based construction from [4], for generating $N = 3^n$ OLE correlations over $\mathbb{F}_{2^k}$, we have that:

- The seed size of each party is around $4t^2 \left(3(\lambda + 1) \log_3 \frac{N}{t} + \lambda + k \right) + 2t(\lceil \log_2 N \rceil + k)$ bits.
- The computation of **Expand** can be done with at most $(1.5 + \lceil \frac{k}{\lambda} \rceil)4Nt$ PRG operations, $5kN \log_3 N$ additions and $kN \log_3 N$ multiplications over $\mathbb{F}_4$, and $4N$ additions and $3N$ multiplications over $\mathbb{F}_{2^k}$.

Proof. We defer the proof to Section C.2 in the Supplementary Material. $\square$

Corollary 4.4. Construction $\text{Cons}_{\text{OLE}}^{\mathbb{F}_{2^k}}$ could be generalized to a Galois ring $\text{GR}_k = \mathbb{GR}(2^\ell, k)$ for any even integer k . To generate $N = 3^n$ OLE instances over GR_k , we have that:

- The seed size of each party is around $4t^2 \left(3(\lambda + 1) \log_3 \frac{N}{t} + \lambda + k\ell \right) + 2t(k\ell + \lceil \log_2 N \rceil)$ bits.
- The computation of **Expand** can be done with at most $(1.5 + \lceil \frac{k\ell}{\lambda} \rceil)4Nt$ PRG operations, $7kN \log_3 N$ additions and $kN \log_3 N$ multiplications over $\mathbb{GR}(2^\ell, 2)$, and $4N$ additions and $3N$ multiplications over GR_k .

5 Applications in MPC and ZK

5.1 Applications in SNARKs

Linear error-correcting codes have been playing a crucial role in building practically efficient SNARKs (succinct non-interactive arguments of knowledge), leading to efficient implementations and numerous real-world applications. Thanks to the rich SNARK literature [2, 6, 7, 37, 59, 3, 19, 44], there is a modular approach to building SNARKs from any linear code, which is also known as the Brakedown's template [37]. Essentially, what we need is merely a fast linear code with a concretely good minimum distance from Brakedown's template, where a fast encoding algorithm implies fast proving, and a high minimum distance translates to a small proof size and verification time. We refer the interested readers to [55] for more details.

From our efficient FFT algorithms given in Section 3, we design a fast encoding algorithm for QA codes over binary extension fields as well as Galois rings. Although most SNARKs focus on a large prime field, binary extension fields and Galois rings have recently received attention due to their compatibility with modern CPU architecture [19, 39, 58, 38]. Let $\text{GR}_k = \mathbb{GR}(2^\ell, k)$, where k is an even integer. Note that for $\ell = 1$, $\mathbb{GR}(2^\ell, k) = \mathbb{F}_{2^k}$ is a binary extension field. We consider a general QA code $\mathcal{C}_{a_1, \dots, a_c} \subseteq (\text{GR}_k[\mathbb{Z}_3^n])^c$ with rank 1 and index c , which is defined as follows:

$$\mathcal{C}_{a_1, \dots, a_c} = \{(ba_1, \dots, ba_c) : b \in \text{GR}_k[\mathbb{Z}_3^n]\} \in (\text{GR}_k[\mathbb{Z}_3^n])^c,$$

where $a_1, \dots, a_c \in \text{GR}_k[\mathbb{Z}_3^n]$. Our encoding algorithm is given in Algorithm 4, where we utilize the fact that $\text{GR}_k[\mathbb{Z}_3^n]$ is essentially nothing else than the polynomial ring $\text{GR}_k[x_1, \dots, x_n]/(x_1^3 - 1, \dots, x_n^3 - 1)$.

Algorithm 4 Encoding of QA code: Enc

Input: The input message $b \in \text{GR}_k[\mathbb{Z}_3^n] \simeq \text{GR}_k[x_1, \dots, x_n]/(x_1^3 - 1, \dots, x_n^3 - 1)$, and evaluations of $a_1, \dots, a_c$ at $\{1, \beta, -\beta - 1\}^n$, denoted as $\hat{a}_1, \dots, \hat{a}_c$, where $\{1, \beta, -\beta - 1\}$ are the roots of $x^3 - 1$ over $\text{GR}_2 = \mathbb{GR}(2^\ell, 2)$.

Output: The codeword $(ba_1, \dots, ba_c)$

- 1: Compute $\hat{b} = \text{FFT}_{\text{GR}_k}(b)$.
 - 2: Compute $\mathbf{b}_i = \hat{b} \star \hat{a}_i$ for $i \in [c]$.
 - 3: Compute $ba_i = \text{IFFT}_{\text{GR}_k}(\mathbf{b}_i)$ for $i \in [c]$.
-

Theorem 5.1 (Theorem 1.2). *It takes $O(ckn \cdot 3^n)$ operations in GR_2 and $c \cdot 3^n$ multiplications in GR_k to encode a QA code $\mathcal{C}_{a_1, \dots, a_c} \subseteq (\text{GR}_k[\mathbb{Z}_3^n])^c$ of rank 1 and index c by Algorithm 4.*

Proof. Efficiency directly follows from the cost of the component-wise multiplications in GR_k and the complexities of the FFT and inverse FFT over GR_k ,

implemented via FFT and inverse FFT computations over GR_2 . The FFT and inverse FFT algorithms are given in [Algorithm 2](#) and [Algorithm 8](#), respectively. Correctness follows from the isomorphism $\text{GR}_k[\mathbb{Z}_3^n] \rightarrow \text{GR}_k^{3^n}$ given by the polynomial evaluation at $\{1, \beta, -\beta - 1\}^n$. $\square$

Remark 5.2. Note that a naive encoding algorithm performs $O(c \cdot 3^{2n})$ operations in GR_k , which is substantially more expensive than that of [Theorem 5.1](#). Concretely, [Algorithm 4](#) performs $\frac{7}{6}k(c+1)n \cdot 3^n$ additions and $\frac{1}{6}k(c+1)n \cdot 3^n$ multiplications in GR_2 (from 1 FFT call and c IFFT calls), and $c \cdot 3^n$ multiplications in GR_k . We remark that in fact the normalization step in IFFT (i.e., multiplying by 3^{-n}) is not necessary, as scalar multiplication preserves distance. Hence, performing IFFT has the same costs as FFT. In addition, by assuming a systematic form, i.e., setting $a_1 = 1$, one can further save one IFFT call and 3^n multiplications in GR_k from [Theorem 5.1](#).

Moreover, as the complexity of multiplication in GR_k scales quadratically with both ℓ, k , [Theorem 5.1](#) is also faster than Reed-Solomon codes over Galois rings, which need $O(cn \cdot 3^n)$ operations over GR_k [39]. Thus, we can plug in our QA code into [39] to obtain faster SNARKs. Beyond that, [39] requires GR_k to have a large 3-smooth multiplicative subgroup for their FFT, and they suggest using $k = 162$ for 128-bit security. In contrast, our encoding algorithm has no such restriction, and we can use $k = 128$. We refer to concrete encoding time in [Table 8](#).

5.2 MPC with Preprocessing

While most existing MPC protocols focus on arithmetic circuits over large prime fields, a series of works [26, 28, 32] consider integer rings $\mathbb{Z}_{2^\ell}$. On the one hand, integer rings $\mathbb{Z}_{2^\ell}$ is naturally compatible with CPU architecture. On the other hand, Boolean circuit can be regarded as a special case when $\ell = 1$, which is particularly suitable for non-linear operations such as comparison.

For dishonest majority MPC over integer rings, there exist two technical lines: The first method [26] lifts the MAC of authenticated sharing from original ring $\mathbb{Z}_{2^\ell}$ to $\mathbb{Z}_{2^{\ell+s}}$, where s denotes a statistical security parameter. Notably, our PCG for OLE over integer rings $\mathbb{Z}_{2^\ell}$ in [Section D](#) can be adapted to generate authenticated Beaver triples of [26] by following [46]. However, its communication cost depends on the security parameter, which is not suitable for the case of small ℓ . To cope with this problem, we can lift elements of $\mathbb{Z}_{2^\ell}$ to a large Galois ring $\text{GR}_k = \mathbb{GR}(2^\ell, k)$ to achieve a small soundness error 2^{-k} . To amortize the overhead of this lift, we follow [32] to apply the reverse multiplication-friendly embedding (RMFE) technique [21, 27, 31].

Definition 5.3 (Reverse Multiplication-Friendly Embedding). Let $\mathbb{Z}_{2^\ell}$ be an integer ring and let $\text{GR}_k = \mathbb{GR}(2^\ell, k)$ be a Galois ring. An (m, k) -RMFE is a pair of $\mathbb{Z}_{2^\ell}$ -linear maps $\phi : \mathbb{Z}_{2^\ell}^m \rightarrow \text{GR}_k$ and $\psi : \text{GR}_k \rightarrow \mathbb{Z}_{2^\ell}^m$, such that for any $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_{2^\ell}^m$, it holds that $\psi(\phi(\mathbf{x}) \cdot \phi(\mathbf{y})) = \mathbf{x} \star \mathbf{y}$, where $\cdot$ is multiplication in GR_k and $\star$ is element-wise product in $\mathbb{Z}_{2^\ell}^m$.

The application of RMFE reduces the overhead of lift to a small ratio $\rho = \frac{k}{m}$, which is shown to be around 4.92 in the asymptotic setting [27, 31]. However, this introduction brings a new problem of re-encoding after each multiplication. This is because RMFE only supports a single multiplication⁵. To solve this problem, we follow [32] to produce multiplication quintuples instead of Beaver triples. Let $\llbracket \cdot \rrbracket$ denote the authenticated sharing, then a multiplication quintuple is $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket \tau(a) \rrbracket, \llbracket \tau(b) \rrbracket, \llbracket \tau(a)\tau(b) \rrbracket)$, where $a, b \xleftarrow{\$} \text{GR}_k$, $\tau : \text{GR}_k \rightarrow \text{GR}_k$ is defined as $\tau := \phi \circ \psi$. By consuming multiplication quintuples, each party locally executes the re-encoding operation.

The generation of multiplication quintuples is based on the production of random authenticated sharings and Beaver triples, which can be obtained by invoking VOLE and OLE protocols over Galois rings, respectively. To generate the MAC for a given additive sharing $[x]$, all parties consume a random authenticated sharing $\llbracket r \rrbracket$ by opening $d \leftarrow [x] - [r]$ and recompute $\llbracket x \rrbracket = d + \llbracket r \rrbracket$. To prevent corrupted parties from delivering an incorrect share, we follow [32] to add a verification after authentication. We summarize the steps of the generation of multiplication quintuples as follows:

1. All parties obtain two random authenticated sharings as $\llbracket a \rrbracket$ and $\llbracket b \rrbracket$ from VOLE correlations.
2. All parties locally execute re-encode operation to obtain $[\tau(a)]$ and $[\tau(b)]$ from $\llbracket a \rrbracket$ and $\llbracket b \rrbracket$, respectively.
3. All parties obtain one Beaver triple $[x], [y], [z]$ from OLE correlations.
4. All parties obtain authenticated sharings $\llbracket \tau(a) \rrbracket, \llbracket \tau(b) \rrbracket, \llbracket x \rrbracket, \llbracket y \rrbracket, \llbracket z \rrbracket$ by consuming 5 random authenticated sharings.
5. All parties open $d \leftarrow [\tau(a)] - [x]$ and $e \leftarrow [\tau(b)] - [y]$, then locally compute

$$\llbracket \tau(a)\tau(b) \rrbracket = \llbracket z \rrbracket + d\llbracket y \rrbracket + e\llbracket x \rrbracket + d \cdot e$$

In our protocol, the generation of a single multiplication quintuple requires 7 openings, which incurs $7nkl = 7n\ell pm = 34.44nml$ bits of communication. Note that for two-party computation, we can directly generate authenticated Beaver triples with sublinear communication [14], thus it is unnecessary to authenticate $[x], [y], [z]$ anymore. In this setting, the communication cost is reduced to $4 \times 2\ell pm = 39.36\ell m$ bits. In contrast, [32] generates a single multiplication quintuple based on OT, which requires $8161.6n(n-1)m\ell$ bits of communication. In conclusion, the communication complexity of [32] is around 415 and $237(n-1)$ times larger than our protocol in two-party and general n -party settings, respectively.

6 Security of QA-SD Assumption

In this section, we analyze QA-SD assumptions in the well-known linear test framework [13]. Since the security of QA-SD over Galois rings reduces to QA-

⁵ Although degree- D RMFE [31] supports any number of multiplications, this increases the ratio ρ rapidly.

SD over finite fields [46], it suffices to consider $\mathbb{F}_{2^k}$. We refer analyses of concrete attacks on QA-SD assumptions to [5, 4, 45, 8, 46, 43].

6.1 Linear Test Framework over Large Fields

The linear test framework provides a unified template to analyze the security of variants of the LPN or syndrome decoding assumptions against the most common attacks. It was first explicitly formalized for linear codes in [13]. In this work, we consider syndrome decoding assumptions, which is also known as dual-LPN.

Informally, the dual-LPN assumption states that the syndrome of a sparse vector is pseudorandom: for a random parity-check matrix H , a random sparse vector $\mathbf{e}$ and a uniform random vector $\mathbf{b}$, the distributions $(H, H\mathbf{e})$ and $(H, \mathbf{b})$ are computationally indistinguishable. In the linear test framework, a PPT adversary $\mathcal{B}$ performs an attack against a dual-LPN instance $(H, \mathbf{b})$ in two stages:

1. First, $\mathcal{B}$ is given access to an (unbounded) adversary $\mathcal{A}$, who on input H returns a (nonzero) test vector $\mathbf{v} = \mathcal{A}(H)$.
2. Second, $\mathcal{B}$ computes $\mathbf{v}^\top \cdot \mathbf{b}$. The advantage of the adversary $\mathcal{B}$ in the linear test framework is the bias of the induced distribution $\mathbf{v}^\top \cdot H \cdot \mathbf{e}$.

The bias of a distribution over a field $\mathbb{F}$ is defined as follows:

Definition 6.1 (Bias of a Distribution). *Given a distribution $\mathcal{D}$ over $\mathbb{F}^n$ and a vector $\mathbf{u} \in \mathbb{F}^n$, the bias of $\mathcal{D}$ with respect to $\mathbf{u}$ is defined as follows:*

$$\text{bias}_{\mathbf{u}}(\mathcal{D}) = |\mathbb{E}_{\mathbf{x} \sim \mathcal{D}}[\mathbf{u}^\top \cdot \mathbf{x}] - \mathbb{E}_{\mathbf{x} \sim \mathcal{U}_n}[\mathbf{u}^\top \cdot \mathbf{x}]| = |\mathbb{E}_{\mathbf{x} \sim \mathcal{D}}[\mathbf{u}^\top \cdot \mathbf{x}] - \frac{1}{|\mathbb{F}|}|,$$

where $\mathcal{U}_n$ denotes the uniform distribution over $\mathbb{F}^n$. The bias of $\mathcal{D}$, is the maximum bias of $\mathcal{D}$ with respect to any nonzero vector $\mathbf{u}$.

Security against linear tests is formally defined as follows:

Definition 6.2 (Security against Linear Tests). *Let $\mathcal{R}$ be a ring, $\mathcal{D} = \{\mathcal{D}_n\}$ be a family of noise distributions over $\mathcal{R}^n$, and $\mathbf{C}$ be a probabilistic code generation algorithm such that $\mathbf{C}(m, n)$ outputs a generating matrix $H \in \mathcal{R}^{m \times n}$. Let $\epsilon, \eta : \mathbb{N} \mapsto [0, 1]$ be two functions. We say that the $(\mathcal{D}, \mathbf{C}, \mathcal{R})$ -LPN(m, n) assumption is (ϵ, η) -secure against linear tests if for any (possibly inefficient) adversary $\mathcal{A}$ which, on input H outputs a non-zero $\mathbf{v} \in \mathcal{R}^n$, it holds that*

$$\Pr[H \xleftarrow{\$} \mathbf{C}(m, n), \mathbf{v} \leftarrow \mathcal{A}(H) : \text{bias}_{\mathbf{v}}(\mathcal{D}_H) > \epsilon(\lambda)] \leq \eta(\lambda),$$

where $\mathcal{D}_H$ denotes the distribution induced by sampling $\mathbf{e} \leftarrow \mathcal{D}_n$, and outputting the LPN samples $H \cdot \mathbf{e}$.

Let $\mathbf{d}(H)$ denote the minimum distance of H , i.e., $\mathbf{d}(H) = \min_{\mathbf{u} \in \mathcal{R}^n \neq \mathbf{0}} \text{wt}(\mathbf{u}^\top \cdot H)$. Following [13, 25, 10], we have the following theorem that translates minimum distance into security under the linear test framework.

Theorem 6.3. Let $\mathcal{R}, \mathcal{D}, \mathbf{C}$ be defined as in [Definition 6.2](#). For any integer d , the $(\mathcal{D}, \mathbf{C}, \mathcal{R})$ -LPN(m, n) assumption is (ϵ_d, η_d) -secure against linear tests, where

$$\epsilon_d = \max_{\text{wt}(\mathbf{v}) \geq d} \text{bias}_{\mathbf{v}}(\mathcal{D}_n), \quad \eta_d = \Pr_{H \leftarrow \mathbf{C}(m, n)} [\delta(H) < d].$$

As in the literature, a typical choice of $\mathcal{D}_n$ is a Bernoulli distribution. Let $\mathcal{R}$ be a field $\mathbb{F}_q$ of q elements. We denote by $\text{Ber}_r(\mathbb{F}_q)$ the Bernoulli distribution that output a random value in $\mathbb{F}_q \setminus \{0\}$ with probability r , and 0 with probability $1 - r$. Then $\mathcal{D}_n = \text{Ber}_r(\mathbb{F}_q^n)$, i.e. the distribution obtained by independently sampling $\text{Ber}_r(\mathbb{F}_q)$ for n times. To compute the bias of the Bernoulli distribution, the well-known piling-up lemma is required. In the following, we show a special case of $\mathbb{F}_2$ as well as the general case.

Lemma 6.4 (Piling-up Lemma over $\mathbb{F}_2$). For any $0 < r < 1/2$ and any integer n , given n random variables $X_1, \dots, X_n$ i.i.d to $\text{Ber}_r(\mathbb{F}_2)$, it holds that $\Pr[\bigoplus_{i=1}^n X_i = 0] = 1/2 + (1 - 2r)^n/2$.

Lemma 6.5 (Piling-up Lemma over $\mathbb{F}_q$). For any $0 < r < 1 - 1/q$ and any integer n , given n random variables $X_1, \dots, X_n$ i.i.d to $\text{Ber}_r(\mathbb{F}_q)$, it holds that $\Pr[\sum_{i=1}^n X_i = 0] = \frac{1}{q} + \frac{q-1}{q} \cdot (1 - \frac{q}{q-1} \cdot r)^n$. Moreover, for any $y \in \mathbb{F}_q \neq 0$, $\Pr[\sum_{i=1}^n X_i = y] = \frac{1}{q} - \frac{1}{q} \cdot (1 - \frac{q}{q-1} \cdot r)^n$.

By [Lemma 6.4](#) & [Lemma 6.5](#), one can upper bound ϵ_d for arbitrary finite fields [\[13\]](#). The results are summarized as follows:

Proof. The proof is deferred to [Section C.3](#) in the Supplementary Material. $\square$

- For $\mathcal{R} = \mathbb{F}_2$, $\mathcal{D} = \text{Ber}_r$, and any $\mathbf{v} \in \mathbb{F}_2^n$ with $\text{wt}(\mathbf{v}) \geq d$, we have $\epsilon_d = \max_{\text{wt}(\mathbf{v}) \geq d} \text{bias}_{\mathbf{v}}(\text{Ber}_r(\mathbb{F}_2^n)) = |\Pr[X_i \sim \text{Ber}_r(\mathbb{F}_2) : \bigoplus_{i=1}^{\text{wt}(\mathbf{v})} X_i = 0] - 1/2| \leq (1 - 2r)^d/2 < e^{-2dr}$.
- For $\mathcal{R} = \mathbb{F}_q$, $\mathcal{D} = \text{Ber}_r$, and any $\mathbf{v} \in \mathbb{F}_q^n$ with $\text{wt}(\mathbf{v}) \geq d$, we have $\epsilon_d = \max_{\text{wt}(\mathbf{v}) \geq d} \text{bias}_{\mathbf{v}}(\text{Ber}_r(\mathbb{F}_q^n)) = |\Pr[X_i \sim \text{Ber}_r(\mathbb{F}_q) : \sum_{i=1}^{\text{wt}(\mathbf{v})} X_i = 0] - 1/q| \leq \frac{q-1}{q} \cdot (1 - \frac{q}{q-1} \cdot r)^d < e^{-\frac{q}{q-1} \cdot dr} < e^{-dr}$, for sufficiently large q .

Basically, (ϵ_d, η_d) -security under the linear test framework means that given an unbounded adversary $\mathcal{A}$ that can produce a test vector $\mathbf{v}$ from H , with probability at least $1 - \eta_d$ (over the randomness of H), adversary $\mathcal{B}$ can distinguish a dual-LPN sample $(H, \mathbf{b})$ from uniformly random with advantage at most ϵ_d . Note that computing an inner product $\mathbf{v}^\top \cdot \mathbf{b}$ requires n operations. As in the literature, we require ϵ_d to be as small as $2^{-\lambda + \log_2 n}$, where λ is the security parameter. In other words, if one wants $(2^{-\lambda + \log_2 n}, \eta_d)$ -security under the linear test framework, it suffices to set the expected noise weight $rn \geq \frac{(\lambda - \log_2 n) \cdot \ln(2)}{2d/n}$ in the binary case.

6.2 QA-SD under Linear Test Framework

Very recently, Li et al. [44] analyzed the concrete minimum distance of rank-1 QA codes, with a special focus on group ring $\mathbb{F}_p[\mathbb{Z}_2^n]$, where $\mathbb{F}_p$ is a prime field. Their arguments essentially apply to QA codes over a general $\mathbb{F}_q[\mathbb{Z}_d^n]$, where $\mathbb{F}_q$ is a finite field of q elements and $d \mid q - 1$, hence fit our case of $\mathbb{F}_{2^k}[\mathbb{Z}_3^n]$. We restate their results as follows:

Theorem 6.6. *Let $H_q(\cdot)$ be the q -ary entropy function and $c \in \mathbb{N}$. Let $\mathbb{G} = \mathbb{Z}_d^n$ with $N := d^n$ and $\gcd(q, N) = 1$. Assume that $\epsilon := 1 - H_q(\delta) - \frac{1 + \log_q N}{c} > 0$. Then, a random (rank-1) QA code $\mathcal{C} \subseteq (\mathbb{F}_q[\mathbb{G}])^c$ has rate $1/c$ and relative minimum distance $\geq \delta$ with probability*

$$\max \left\{ 1 - \frac{c(c-1)N}{2q^2} - \frac{q^{-\lceil \frac{c-1}{c\delta} \rceil c\epsilon}}{(1 - q^{-c\epsilon})(q-1)}, \quad 1 - \frac{cN}{q} - \frac{q^{-\lceil \frac{1}{\delta} \rceil c\epsilon}}{(1 - q^{-c\epsilon})(q-1)} \right\}.$$

By Theorem 6.6, we derive a concrete minimum distance of QA codes over $\mathbb{F}_{2^k}[\mathbb{Z}_3^n]$, which allows us to analyze the security of QA-SD assumptions under the linear test framework. We then suggest various parameters for QA-SD($c, \mathbb{F}_{2^k}[\mathbb{Z}_3^n], t$) (Definition 2.7) in Table 4, targeting a reasonable security level. Thanks to the strong distance bound, we achieve 127-bit security level of η , which is substantially larger than those in the literature [10, 52, 19, 42]. We remark that here we choose t such that $t \mid N$, which allows us to use a regular noise distribution that will improve concrete efficiency in our PCG applications.

$N = 3^n$	k	Index c	Rate r	Distance δ	ct	$-\log_2(\epsilon)$	η
3^{13}	128	2	1/2	0.4118	162	96.2	2^{-127}
		3	1/3	0.6054	81	70.7	2^{-127}
		4	1/4	0.7028	108	109.5	2^{-127}
		5	1/5	0.7616	135	148.3	2^{-127}
3^{16}	128	2	1/2	0.3933	162	91.9	2^{-127}
		3	1/3	0.5930	81	69.3	2^{-127}
		4	1/4	0.6935	108	109.0	2^{-127}
		5	1/5	0.7540	135	146.8	2^{-127}

Table 4: Conservative Parameters for QA-SD over $\mathbb{F}_{2^k}[\mathbb{Z}_3^n]$

Aggressive parameters. We remark that parameters suggested by the linear test framework are very conservative, since the complexity of finding a vector $\mathbf{v}$ such that $\mathbf{v}^\top \cdot H$ is a low-weight codeword is not taken into account. However, for many codes such as EA/EC/BA/QA codes, it is believed be a very hard task, which, like random linear codes, should take $O(|\mathbb{F}|^{O(m)})$ time by enumeration.

Hence, in the PCG literature, many aggressive parameters are also suggested for better practical efficiency. According to concrete attack analyses on QA-SD over large fields [43] as well as Table 4, it is reasonable to use QA-SD($3, \mathbb{F}_q, 27$) for 128-bit computational security, when $\mathbb{F}_q$ is at least 128-bit.

QA-SD over modest-sized fields. QA-SD assumption was first proposed to build programmable PCGs over small fields [5], such as $\mathbb{F}_3, \mathbb{F}_4$. Follow-up works [4, 45] develop techniques for constructing PCGs over $\mathbb{F}_2$ by relying on QA-SD over $\mathbb{F}_4$. However, a recent work [8] breaks QA-SD($c, \mathbb{F}_q[\mathbb{Z}_d^n], t$) when $\mathbb{F}_q$ is a small field. Specifically, their attack works only for the case that

$$\log_d(q) < \frac{n-1}{d(c-1)}.$$

As long as $q \geq 16$, their attack fails in most parameters of interest and our cases ($q \geq 2^{64}$) are immune to it. On the other hand, analyzing the hardness of QA-SD over modest-sized fields ($16 \leq q \leq 2^{32}$) is of great importance, because [45]’s trace-based approach for $\mathbb{F}_2$ can rely on QA-SD over arbitrary $\mathbb{F}_{2^k}$, and a smaller k would yield better computational efficiency.

However, Theorem 6.6 does not lead to a good minimum distance for such q . Towards understanding QA-SD over modest-sized fields, we give some concrete analyses under the linear test framework for $q = 2^{32}$, which is the smallest field for which we can get good results, since we need either $2|\mathbb{F}_q|^2 \gg c(c-1)N$ or $|\mathbb{F}_q| \gg cN$ by Theorem 6.6. The results are summarized as follows:

- A random QA code over $\mathbb{F}_{2^{32}}[\mathbb{Z}_3^9]$ of index $c = 3$, has relative minimum distance 0.3828 with probability at least $1 - 2^{-42}$. Hence, QA-SD($3, \mathbb{F}_{2^{32}}[\mathbb{Z}_3^9], 27$) is $(2^{-44.9}, 2^{-42})$ -secure against linear tests.
- A random QA code over $\mathbb{F}_{2^{32}}[\mathbb{Z}_3^{13}]$ of index $c = 5$, has relative minimum distance 0.5149 with probability at least $1 - 2^{-52}$. Hence, QA-SD($5, \mathbb{F}_{2^{32}}[\mathbb{Z}_3^{13}], 27$) is $(2^{-100.3}, 2^{-52})$ -secure against linear tests.

7 Efficiency Analysis

We implement the multivariate polynomial evaluation based on FFT algorithms in C, which is an essential part of our PCG construction. Our implementation focuses on the extension field $\mathbb{F}_{2^{128}}$. This is because as a comparison, EC, EA, BA and RAA codes are only available over fields instead of Galois rings. 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. We refer to <https://github.com/zhli271828/ExtensionFieldPCG> for our full-fledged implementation of PCG and PCS over $\mathbb{F}_{2^{128}}$.

Efficiency of Syndrome Encoding. We note that it is not practical to implement the FFT for the ring $\mathcal{R} = \mathbb{F}_{2^k}[\mathbf{X}]/(\mathbf{X}^N - 1)$ as the smooth factor of $2^k - 1$ is very small. Thus, we only benchmark the syndrome encoding time of our QA

code and compare it with linear-time encodable codes. We implement the syndrome encoding of QA code with the compression factor $c = 2$. The syndrome encoding of QA code consists of two FFT transforms and a single component-wise multiplication. To compare with state-of-the-art VOLE protocols, we use the libOTe library [50] to benchmark the encoding time of expand-accumulate (EA) code [10], expand-convolute (EC) code [52] and block-accumulate (BA) code [42].⁶ 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 other codes depends on more parameters. To make a fair comparison, we choose the parameters of these codes with the same security level. For EA code, we set the expander weight and rate to 21 and 0.2, respectively. For EC code, we set the expander weight and convolution state size to 7 and 24, respectively. For BA code, we set the depth and state size as 3 and 32, respectively. The rate of BA and 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 7.7x-21.3x, 3.9x-5.8x, 2.7x-3.3x faster than EA, BA and EC code, respectively.

N	EA	EC	BA	QA
3^{13}	302 ms	104 ms	208 ms	39 ms
3^{14}	975 ms	354 ms	724 ms	124 ms
3^{15}	3.06 s	1.14 s	1.72 s	0.39 s
3^{16}	10.47 s	3.58 s	6.65 s	1.26 s
3^{17}	46.44 s	11.18 s	15.49 s	3.89 s
3^{18}	256.53 s	40.43 s	66.89 s	12.25 s

Table 5: The syndrome encoding time of expand-accumulate (EA) code, expand-convolute (EC) code, block-accumulate (BA) code and quasi-abelian (QA) code to generate a syndrome of length N over $\mathbb{F}_{2^{128}}$. 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 $R = 0.5$. For BA code, we set depth $t = 3$, state size $\sigma = 32$ and rate $R = 0.5$.

Efficiency of VOLE. We use libOTe library to fully implement our new VOLE protocol in Section 4.1. 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, EC and BA codes. Following Table 4, we

⁶ We use class block in CoeffCtx file to carry out the operations over $\mathbb{F}_{2^{128}}$. However, since our QA code is defined over $\mathbb{F}_{4^{64}}$ to be compatible with FFT algorithm, we also add the support for field $\mathbb{F}_{4^{64}}$ in CoeffCtx file.

choose $t = 81$ and $c = 2$ to achieve 128-bit computational security. We directly follow libOTe to set the Hamming weight t of EA, EC and BA 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 5.3x-9.7x, 1.9x-2.5x and 1.7x-2.0x faster than VOLE based on EA, BA and EC code, respectively.

N	EA	EC	BA	QA
3^{13}	1.44 s	0.47 s	0.68 s	0.27 s
3^{14}	4.60 s	1.55 s	2.36 s	0.97 s
3^{15}	14.35 s	4.88 s	5.76 s	2.90 s
3^{16}	44.85 s	15.53 s	21.21 s	8.98 s
3^{17}	189.89 s	47.63 s	51.86 s	27.47 s
3^{18}	803.94 s	163.76 s	210.89 s	83.94 s

Table 6: The runtime of VOLE protocols over $\mathbb{F}_{2^{128}}$ based on expand-accumulate (EA) code, expand-convolute (EC) code, block-accumulate (BA) 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 $t = 81$. 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 $R = 0.5$. For BA code, we set depth $t = 3$, state size $\sigma = 32$ and rate $R = 0.5$. The Hamming weight t of EA, EC and BA code is 248, which is computed by the libOTe library.

Efficiency of OLE. We use libOTe library to fully implement our new OLE protocol over $\mathbb{F}_{2^{128}}$ with a distributed setup in Section 4.2. Note that there are two difficulties for implementing OLE protocol based on Ring-LPN assumption. The first one is that we can not find a large smooth factor for an integer of the form $2^k - 1$. Even if we overcome this obstacle, the next challenge is to construct an efficient DPF protocol to be compatible with Ring-LPN assumption. It is worth to mention that such DPF function requires expensive BitAdd calls that are used to compute bit-wise shares over $\mathbb{Z}_N$ as required by [14]. This DPF function seems to have not been implemented yet. On the other hand, our QA code is compatible with ternary DPF function which was first proposed in [4] and was available in libOTe. Table 7 implies that our OLE protocol generates roughly 42,000 OLEs over $\mathbb{F}_{2^{128}}$ per second.

Efficiency of Encoding for SNARKs. In code-based SNARKs, the efficiency is mainly dominated by the time of encoding. To our best knowledge, the state-of-the-art code constructed for large binary extension fields is RAA codes [19]. We compare the performance of our QA code with RAA codes over $\mathbb{F}_{2^{128}}$ in terms of the runtime. We benchmark RAA codes without puncturing, with rates

N	$(c, t) = (2, 81)$		$(c, t) = (3, 27)$	
	Runtime of OLE (s)	Communication cost (MB)	Runtime of OLE (s)	Communication cost (MB)
3^{10}	2.02	44.13	1.81	12.84
3^{11}	6.17	51.35	4.17	14.65
3^{12}	17.66	58.57	12.48	16.45
3^{13}	50.99	65.79	37.40	18.26
3^{14}	162.82	73.01	111.07	20.07
3^{15}	544.24	80.23	340.20	21.87

Table 7: Comparison of OLE protocols with different (c, t) parameters to generate N OLE instances over $\mathbb{F}_{2^{128}}$.

1/2, 1/4 and 1/8. We use systematic QA code of rate 1/2 whose encoding consists of one FFT, one inverse FFT algorithm and one component-wise product. **Table 8** shows our QA code is even 1.9x-2.75x faster than RAA code of rate 1/2. Note that the current distance analyses for such RAA codes are restricted to rate $\leq 1/4$ ⁷.

Message length	QA	RAA (1/2)	RAA (1/4)	RAA (1/8)
3^{13}	38 ms	74 ms	156 ms	321 ms
3^{14}	121 ms	238 ms	491 ms	1013 ms
3^{15}	0.38 s	0.75 s	1.54 s	3.14 s
3^{16}	1.22 s	2.33 s	4.73 s	9.80 s
3^{17}	3.80 s	7.19 s	16.49 s	45.08 s
3^{18}	12.15 s	31.01 s	75.97 s	NA

Table 8: Comparison of encoding time for QA codes of rate 1/2, RAA codes of rate 1/2, 1/4, 1/8.

Acknowledgment. The authors would like to thank anonymous reviewers at ASIACRYPT 2026 for their helpful suggestions. The work was supported in part by the National Key Research and Development Program of China

⁷ Rate-1/2 RAA codes with linear minimum distance can be alternatively obtained by puncturing a low-rate mother code. We refer to [41] for details. Such punctured constructions are not considered in our implementation and may incur additional encoding overhead.

under Grants 2022YFA1004900, 2023YFE0123900 and 2025YFA1017202, and the National Natural Science Foundation of China under Grants 12361141818, 12426302, 12271084 and 12571577. Zhe Li is supported in part by Natural Science Basic Research Program of Shaanxi (Program No. 2026JC-YBMS-0779).

References

1. Agarwal, A., Raghuraman, S., Rindal, P.: Fully distributed multi-point functions for PCGs and beyond. *Cryptology ePrint Archive*, Paper 2025/2294 (2025), <https://eprint.iacr.org/2025/2294> 5
2. Ames, S., Hazay, C., Ishai, Y., Venkatasubramanian, M.: Liger: Lightweight sub-linear arguments without a trusted setup. In: *CCS 2017*. pp. 2087–2104. ACM (2017) 20
3. Block, A.R., Fang, Z., Katz, J., Thaler, J., Waldner, H., Zhang, Y.: Field-agnostic snarks from expand-accumulate codes. In: *A CRYPTO 2024*. LNCS, vol. 14929, pp. 276–307. Springer (2024) 20
4. Bombar, M., Bui, D., Couteau, G., Couvreur, A., Ducros, C., Servan-Schreiber, S.: FOLEAGE: $\mathbb{F}_4$ OLE-based multi-party computation for boolean circuits. In: *ASIACRYPT 2024*. LNCS, vol. 15489, pp. 69–101. Springer (2024) 2, 11, 19, 23, 26, 28, 38, 43, 47
5. Bombar, M., Couteau, G., Couvreur, A., Ducros, C.: Correlated pseudorandomness from the hardness of quasi-abelian decoding. In: *CRYPTO 2023*. LNCS, vol. 14084, pp. 567–601. Springer (2023) 2, 5, 7, 11, 18, 23, 26, 43
6. Bootle, J., Cerulli, A., Ghadafi, E., Groth, J., Hajiabadi, M., Jakobsen, S.K.: Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In: *ASIACRYPT 2017*. LNCS, vol. 10626, pp. 336–365. Springer (2017) 20
7. Bootle, J., Chiesa, A., Liu, S.: Zero-knowledge iops with linear-time prover and polylogarithmic-time verifier. In: *EUROCRYPT 2022*. LNCS, vol. 13276, pp. 275–304. Springer (2022) 20
8. Bouillaguet, C., Delaplace, C., Hamdad, M., Vergnaud, D.: Practical cryptanalysis of pseudorandom correlation generators based on quasi-abelian syndrome decoding. In: *ASIACRYPT 2025*. *Lecture Notes in Computer Science*, vol. 16248, pp. 436–465. Springer (2025) 23, 26
9. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: *CCS 2018*. pp. 896–912. ACM (2018) 2, 15
10. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Resch, N., Scholl, P.: Correlated pseudorandomness from expand-accumulate codes. In: *CRYPTO 2022*. LNCS, vol. 13508, pp. 603–633. Springer (2022) 2, 23, 25, 27
11. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round OT extension and silent non-interactive secure computation. In: *CCS 2019*. pp. 291–308. ACM (2019) 2, 5, 15, 17, 34, 36, 42
12. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: *CRYPTO 2019*. LNCS, vol. 11694, pp. 489–518. Springer (2019) 2, 5, 15, 18, 34
13. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Correlated pseudorandom functions from variable-density LPN. In: Irani, S. (ed.) *FOCS 2020*. pp. 1069–1080. IEEE (2020) 22, 23, 24
14. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators from ring-LPN. In: *CRYPTO 2020*. LNCS, vol. 12171, pp. 387–416. Springer (2020) 2, 5, 6, 18, 22, 28, 43

15. Boyle, E., Gilboa, N., Hamilis, M., Ishai, Y., Tu, Y.: Improved constructions for distributed multi-point functions. In: 2025 IEEE Symposium on Security and Privacy (SP). pp. 44–44. IEEE Computer Society (2024) [5](#), [36](#)
16. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: EUROCRYPT 2015. LNCS, vol. 9057, pp. 337–367. Springer (2015) [35](#)
17. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: CCS 2016. pp. 1292–1303. ACM (2016) [35](#), [36](#)
18. Braun, L., de Saint Guilhem, C.D., Jadoul, R., Orsini, E., Smart, N.P., Tanguy, T.: Zk-for-z2k: Mpc-in-the-head zero-knowledge proofs for $\mathbb{Z}_{2^k}$. In: IMACC 2023. LNCS, vol. 14421, pp. 137–157. Springer (2023) [3](#)
19. Brehm, M., Chen, B., Fisch, B., Resch, N., Rothblum, R.D., Zeilberger, H.: Blaze: Fast snarks from interleaved RAA codes. In: EUROCRYPT 2025. LNCS, vol. 15604, pp. 123–152. Springer (2025) [3](#), [4](#), [20](#), [25](#), [28](#)
20. Cantor, D.G.: On arithmetical algorithms over finite fields. J. Comb. Theory A **50**(2), 285–300 (1989) [6](#)
21. Cascudo, I., Cramer, R., Xing, C., Yuan, C.: Amortized complexity of information-theoretically secure MPC revisited. In: CRYPTO 2018. LNCS, vol. 10993, pp. 395–426. Springer (2018) [3](#), [21](#)
22. Cascudo, I., Giunta, E.: On interactive oracle proofs for boolean R1CS statements. In: FC 2022. LNCS, vol. 13411, pp. 230–247. Springer (2022) [3](#)
23. Cascudo, I., Gundersen, J.S.: A secret-sharing based MPC protocol for boolean circuits with good amortized complexity. In: TCC 2020. LNCS, vol. 12551, pp. 652–682. Springer (2020) [3](#), [6](#)
24. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex fourier series. Mathematics of computation **19**(90), 297–301 (1965) [6](#)
25. Couteau, G., Rindal, P., Raghuraman, S.: Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In: CRYPTO 2021. LNCS, vol. 12827, pp. 502–534. Springer (2021) [2](#), [23](#)
26. Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.: Spd $\mathbb{Z}_{2^k}$: Efficient MPC mod $\mathbb{Z}_{2^k}$ for dishonest majority. In: CRYPTO 2018. LNCS, vol. 10992, pp. 769–798. Springer (2018) [21](#)
27. Cramer, R., Rambaud, M., Xing, C.: Asymptotically-good arithmetic secret sharing over $\mathbb{Z}/p^\ell\mathbb{Z}$ with strong multiplication and its applications to efficient MPC. In: CRYPTO. LNCS, vol. 12827, pp. 656–686. Springer (2021) [3](#), [21](#), [22](#)
28. Damgård, I., Escudero, D., Frederiksen, T.K., Keller, M., Scholl, P., Volgushev, N.: New primitives for actively-secure MPC over rings with applications to private machine learning. In: 2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19–23, 2019. pp. 1102–1120. IEEE (2019) [21](#)
29. Diamond, B.E., Posen, J.: Succinct arguments over towers of binary fields. In: EUROCRYPT 2025. LNCS, vol. 15604, pp. 93–122. Springer (2025) [3](#)
30. Divsalar, D., Jin, H., McEliece, R.J.: Coding theorems for “turbo-like” codes. In: Proceedings of the annual Allerton Conference on Communication control and Computing. vol. 36, pp. 201–210. University Of Illinois (1998) [3](#)
31. Escudero, D., Hong, C., Liu, H., Xing, C., Yuan, C.: Degree-d reverse multiplication-friendly embeddings: Constructions and applications. In: ASIACRYPT 2023. LNCS, vol. 14438, pp. 106–138. Springer (2023) [21](#), [22](#)
32. Escudero, D., Xing, C., Yuan, C.: More efficient dishonest majority secure computation over $\mathbb{Z}_{2^k}$ via galois rings. In: CRYPTO 2022. LNCS, vol. 13507, pp. 383–412. Springer (2022) [3](#), [6](#), [21](#), [22](#)
33. Ganesh, C., Nitulescu, A., Soria-Vazquez, E.: Rinocchio: Snarks for ring arithmetic. J. Cryptol. **36**(4), 41 (2023) [3](#)

34. Gao, S., Mateer, T.D.: Additive fast fourier transforms over finite fields. *IEEE Trans. Inf. Theory* **56**(12), 6265–6272 (2010) [6](#)
35. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: *EUROCRYPT 2014*. LNCS, vol. 8441, pp. 640–658. Springer (2014) [35](#)
36. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. *J. ACM* **33**(4), 792–807 (1986) [37](#)
37. Golovnev, A., Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic snarks for R1CS. In: *CRYPTO 2023*. LNCS, vol. 14082, pp. 193–226. Springer (2023) [5](#), [20](#)
38. Huang, M.M., Mao, X., Zhang, J.: Sublinear proofs over polynomial rings. *IACR Cryptol. ePrint Arch.* p. 199 (2025), <https://eprint.iacr.org/2025/199> [20](#)
39. Jia, Y., Li, S., Xing, C., Yao, Y., Yuan, C.: Polynomial commitments for galois rings and applications to snarks over $\mathbb{Z}_{2^k}$. In: *CRYPTO 2025*. LNCS, vol. 16005, pp. 515–548. Springer (2025) [3](#), [5](#), [20](#), [21](#)
40. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: *CCS 2013*. pp. 669–684. ACM (2013) [36](#)
41. Kliewer, J., Zigangirov, K.S., Costello Jr, D.J.: New results on the minimum distance of repeat multiple accumulate codes. In: *Proc. 45th Annual Allerton Conf. Commun., Control, and Computing*. vol. 3, p. 8 (2007) [29](#)
42. Kolesnikov, V., Pecený, S., Rachuri, R., Raghuraman, S., Rindal, P., Shah, H.: Block-accumulate codes: Accelerated linear codes for pcgs and zk. In: *CRYPTO 2026*. pp. 397–429. Springer (2026) [2](#), [25](#), [27](#)
43. Li, Z., Liu, H., Xing, C., Yao, Y., Yuan, C.: Faster pseudorandom correlation generators via walsh-hadamard transform. In: *CRYPTO 2026*. pp. 430–461. Springer (2026) [2](#), [4](#), [23](#), [26](#)
44. Li, Z., Liu, H., Xing, C., Yao, Y., Yuan, C.: More efficient SNARKs via quasi-abelian codes: Faster, smaller, and field-agnostic. *Cryptology ePrint Archive*, Paper 2026/939 (2026), <https://eprint.iacr.org/2026/939> [4](#), [5](#), [20](#), [25](#)
45. Li, Z., Xing, C., Yao, Y., Yuan, C.: Efficient pseudorandom correlation generators for any finite field. In: *EUROCRYPT 2025*. LNCS, vol. 15605, pp. 145–175. Springer (2025) [2](#), [11](#), [23](#), [26](#)
46. Li, Z., Xing, C., Yao, Y., Yuan, C.: Efficient pseudorandom correlation generators over $\mathbb{Z}/p^k\mathbb{Z}$. In: *CRYPTO 2025*. LNCS, vol. 16003, pp. 207–240. Springer (2025) [2](#), [5](#), [10](#), [18](#), [21](#), [23](#), [44](#), [45](#), [47](#)
47. Lin, F., Xing, C., Yao, Y.: More efficient zero-knowledge protocols over $\mathbb{Z}_{2^k}$ via galois rings. In: *CRYPTO 2024*. LNCS, vol. 14928, pp. 424–457. Springer (2024) [3](#), [6](#)
48. Lin, S., Al-Naffouri, T.Y., Han, Y.S.: FFT algorithm for binary extension finite fields and its application to reed-solomon codes. *IEEE Trans. Inf. Theory* **62**(10), 5343–5358 (2016) [6](#)
49. Lin, S., Al-Naffouri, T.Y., Han, Y.S., Chung, W.: Novel polynomial basis with fast fourier transform and its application to reed-solomon erasure codes. *IEEE Trans. Inf. Theory* **62**(11), 6284–6299 (2016) [3](#), [6](#)
50. Peter Rindal, L.R.: libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. <https://github.com/osu-crypto/libOTe> [27](#)
51. Rachuri, R., Scholl, P.: Le mans: Dynamic and fluid MPC for dishonest majority. In: *CRYPTO 2022*. LNCS, vol. 13507, pp. 719–749. Springer (2022) [6](#)
52. Raghuraman, S., Rindal, P., Tanguy, T.: Expand-convolute codes for pseudorandom correlation generators from LPN. In: *CRYPTO 2023*. LNCS, vol. 14084, pp. 602–632. Springer (2023) [2](#), [25](#), [27](#)

53. Setty, S.T.V.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: CRYPTO 2020. LNCS, vol. 12172, pp. 704–737. Springer (2020) [5](#)
54. Spielman, D.A.: Linear-time encodable and decodable error-correcting codes. IEEE Trans. Inf. Theory **42**(6), 1723–1731 (1996) [3](#)
55. Thaler, J.: Proofs, arguments, and zero-knowledge. Found. Trends Priv. Secur. **4**(2-4), 117–660 (2022) [20](#)
56. Von Zur Gathen, J., Gerhard, J.: Modern computer algebra. Cambridge university press (2003) [4](#), [6](#)
57. Wan, Z.X.: Lectures on finite fields and galois rings (2003) [9](#)
58. Wei, Y., Zhang, X., Deng, Y.: Transparent snarks over galois rings. In: PKC 2025. LNCS, vol. 15674, pp. 418–451. Springer (2025) [3](#), [5](#), [20](#)
59. Xie, T., Zhang, Y., Song, D.: Orion: Zero knowledge proof with linear prover time. In: CRYPTO 2022. LNCS, vol. 13510, pp. 299–328. Springer (2022) [20](#)
60. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated OT with small communication. In: CCS 2020. pp. 1607–1626. ACM (2020) [2](#)

Supplementary Material

A More Preliminaries

A.1 Pseudorandom Correlation Generators

The notion of pseudorandom correlation generators PCG was first introduced in [11, 12]. 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.

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 $\ell \in \text{poly}(\lambda)$.*

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*

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

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 $(\text{PCG.Gen}, \text{PCG.Expand})$ with the following syntax:*

- $\text{PCG.Gen}(1^\lambda)$ is a PPT algorithm that given the security parameter λ , outputs a pair of seeds $(\mathbf{k}_0, \mathbf{k}_1)$.
- $\text{PCG.Expand}(\sigma, \mathbf{k}_\sigma)$ 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 $(\text{PCG.Gen}, \text{PCG.Expand})$ should satisfy the following:

- **Correctness.** The following distribution

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

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:

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

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 $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ for some groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$).

Definition A.4 (Programmability). *We say a PCG $\text{PCG} = (\text{PCG.Gen}, \text{PCG.Expand})$ for a simple bilinear 2-party correlation $\mathcal{C}$ satisfies programmability, if $\text{PCG.Gen}(1^\lambda)$ takes additional random inputs $\rho_0, \rho_1 \in \{0, 1\}^\ell$, for a $\text{poly}(\lambda)$ size ℓ , such that:*

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

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

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

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

as well as

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

A.2 Function Secret Sharing and Puncturable PRF

Function secret sharing (FSS) [16, 17] allows to share functions succinctly among two parties. More concisely, for some secret function $f : I \rightarrow \mathbb{G}$ 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) [35, 16] is a useful and fundamental kind of FSS schemes, specialized for sharing point functions. We give a formal definition of DPF as follows.

Definition A.5 (Distributed point function). Given an input domain $I = [0, N)$, and an Abelian group $(\mathbb{G}, +)$, the point function $f_{\alpha, \beta} : I \rightarrow \mathbb{G}$ 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.

- $\text{DPF.Gen}(1^\lambda, \alpha, \beta)$ 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 $(\mathbf{k}_0^{\text{dpf}}, \mathbf{k}_1^{\text{dpf}})$.
- $\text{DPF.Eval}(\sigma, \mathbf{k}_\sigma^{\text{dpf}}, x)$ is a polynomial-time algorithm that, given a key $\mathbf{k}_\sigma^{\text{dpf}}$ 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 $z_\sigma \leftarrow \text{DPF.Eval}(\sigma, \mathbf{k}_\sigma^{\text{dpf}}, x)$ for $\sigma \in \{0, 1\}$ and $(\mathbf{k}_0^{\text{dpf}}, \mathbf{k}_1^{\text{dpf}}) \xleftarrow{\$} \text{DPF.Gen}(1^\lambda, \alpha, \beta)$.
- **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 $\{(\mathbf{k}_\sigma^{\text{dpf}} : (\mathbf{k}_0^{\text{dpf}}, \mathbf{k}_1^{\text{dpf}}) \xleftarrow{\$} \text{DPF.Gen}(1^\lambda, \alpha, \beta))\}$ and $\{\mathbf{k}_\sigma^{\text{dpf}} \xleftarrow{\$} \mathcal{S}(1^\lambda, I, \mathbb{G})\}$ are computationally indistinguishable.

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

The typical DPF construction [17] uses an arbitrary pseudorandom generator (PRG) $G : \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda+2}$ 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 [15] 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 [40] and then serve as an improvement to replace distributed point function to design PCG for VOLE and OT [11].

Definition A.6 (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 $(F.\text{Gen}, F.\text{Puncture}, F.\text{Eval})$ such that

1. $F.\text{Gen}(1^\lambda)$ outputs a random key $K \in \mathcal{K}$.

2. $\text{F.Puncture}(K, S)$ receives the input $S \subset \mathcal{X}$ of size t and K and outputs a punctured key $K\{S\} \in \mathcal{K}_p$.
3. $\text{F.Eval}(K\{S\}, x)$ 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 [36]. Recall that in VOLE, the sender holds the vector $\mathbf{v} = (v_1, \dots, v_N)$ 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.7. *Let R be a ring, $\mathbb{G}$ be a multiplicative group, and $R[\mathbb{G}]$ be a group ring in Definition 2.5. 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 Galois ring $\text{GR}_k := \mathbb{GR}(2^\ell, k)$ and $\mathbb{G}$ an Abelian group. By fixing an ordering $g_1, \dots, g_n$ of the elements in $\mathbb{G}$, $\text{GR}_k[\mathbb{G}]$ is isomorphic to GR_k^n with respect to $\varphi : \text{GR}_k[\mathbb{G}] \rightarrow \text{GR}_k^n, \sum_{i=1}^n a_i g_i \mapsto (a_1, \dots, a_n)$, where the multiplication over GR_k^n is defined as follows:

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

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

$$\mathbf{A} = \begin{pmatrix} \varphi(a \cdot g_1) \\ \vdots \\ \varphi(a \cdot g_n) \end{pmatrix} \in \text{GR}_k^{n \times n}.$$

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

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

Lemma A.8. *Let $\mathbb{G}_1, \mathbb{G}_2$ be two finite Abelian groups. Then*

$$\text{GR}_k[\mathbb{G}_1 \times \mathbb{G}_2] \simeq \text{GR}_k[\mathbb{G}_1] \otimes_{\text{GR}_k} \text{GR}_k[\mathbb{G}_2],$$

where $\times$ denotes the direct product, and $\otimes_{\text{GR}_k}$ denotes the tensor product over GR_k .

Examples. We give concrete examples of group rings.

- Let $\mathbb{G} = \{1\}$ be the trivial group consisting of only an identity. Then $\text{GR}_k[G]$ is isomorphic to GR_k .
- 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 $\text{GR}_k[\mathbb{G}]$ is essentially the polynomial ring $\text{GR}_k[\mathbf{X}]/(\mathbf{X}^n - 1)$, with respect to the ordering $\bar{0}, \bar{1}, \dots, \overline{n-1}$ and the isomorphism $\varphi : \text{GR}_k[\mathbb{G}] \rightarrow \text{GR}_k[\mathbf{X}]/(\mathbf{X}^n - 1)$, $\sum_{i=0}^{n-1} a_i \bar{i} \mapsto \sum_{i=0}^{n-1} a_i \mathbf{X}^i$.
- Let $\mathbb{G} = \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$ with n_1, n_2 coprime to p . Then by [Lemma A.8](#),

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

Moreover, assuming n divides $p - 1$, and $\alpha_0, \dots, \alpha_{n-1}$ are n roots of $\mathbf{X}^n - 1$ over GR_k . Then, $\phi : f(\mathbf{X}) \mapsto (f(\alpha_0), \dots, f(\alpha_{n-1}))$ is a ring isomorphism of $\text{GR}_k[\mathbf{X}]/(\mathbf{X}^n - 1)$ and the vector space GR_k^n . Hence, $\phi \circ \varphi : \text{GR}_k[\mathbb{G}] \rightarrow \text{GR}_k^n$ is a ring isomorphism. Finally let $n \mid p - 1$ and $\mathbb{G} = \mathbb{Z}_n^\ell$, then by [Lemma A.8](#), $\text{GR}_k[\mathbb{G}] \simeq \text{GR}_k[\mathbf{X}_1, \dots, \mathbf{X}_\ell]/(\mathbf{X}_1^n - 1, \dots, \mathbf{X}_\ell^n - 1) \simeq \text{GR}_k^N$, where $N = n^\ell$.

B Missing Algorithms

In this section, we first follow [\[4\]](#) to recall the ternary transform over $\mathbb{F}_4$. Let $\alpha \in \mathbb{F}_4$ be the root of irreducible polynomial $x^2 + x + 1 \in \mathbb{F}_2[x]$, satisfying $\mathbb{F}_4 = \mathbb{F}_2(\alpha)$. We work in the ring $\mathcal{R}_2 := \mathbb{F}_4[\mathbf{X}_1, \dots, \mathbf{X}_n]/(\mathbf{X}_1^3 - 1, \dots, \mathbf{X}_n^3 - 1)$ with $N = 3^n$.

Let $\mathbf{f} = (f_0, \dots, f_{N-1}) \in \mathbb{F}_4^N$ denote the coefficients of $f \in \mathcal{R}_2$. The Fast-Fourier transform is to compute $\hat{\mathbf{f}} \in \mathbb{F}_4^N$, i.e., the evaluations of f at $\{1, \alpha, \alpha+1\}^n$. We have that

$$\hat{\mathbf{f}} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 \\ 1 & \alpha^2 & \alpha^4 \end{pmatrix}^{\otimes n} \mathbf{f} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha+1 \\ 1 & \alpha+1 & \alpha \end{pmatrix}^{\otimes n} \mathbf{f} \quad (3)$$

where $H^{\otimes n}$ represents n tensor products of H . We construct the efficient FFT algorithm based on [Eq.\(3\)](#), which is displayed in [Algorithm 5](#) below.

Theorem B.1. *[Algorithm 5](#) requires $\frac{5}{3}N \log_3 N$ additions and $\frac{1}{3}N \log_3 N$ multiplications over $\mathbb{F}_4$.*

Algorithm 5 Fast Fourier Transform over $\mathbb{F}_4$: $\text{FFT}_{\mathbb{F}_4}$

Input: Polynomial $f \in \mathcal{R}_2 = \mathbb{F}_4[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$ with coefficient vector $\mathbf{f} = (f_0, \dots, f_{N-1}) \in \mathbb{F}_4^N$, where $N = 3^n$

Output: Evaluation vector $\widehat{\mathbf{f}} \in \mathbb{F}_4^N$ of f at all points in $\{1, \alpha, \alpha + 1\}^n$

1: **if** $n = 0$ **then**

2: Return $\mathbf{f}$

3: **else**

4: Split $\mathbf{f}$ into three vectors of length $m = 3^{n-1}$: //Recall that the coefficients of f are in the lexicographic order of the monomials.

$$\mathbf{f}_L = (f_0, f_1, \dots, f_{m-1})$$

$$\mathbf{f}_M = (f_m, f_{m+1}, \dots, f_{2m-1})$$

$$\mathbf{f}_R = (f_{2m}, f_{2m+1}, \dots, f_{3m-1})$$

5: Compute $\widehat{\mathbf{f}}_L = \text{FFT}_{\mathbb{F}_4}(\mathbf{f}_L)$, $\widehat{\mathbf{f}}_M = \text{FFT}_{\mathbb{F}_4}(\mathbf{f}_M)$ and $\widehat{\mathbf{f}}_R = \text{FFT}_{\mathbb{F}_4}(\mathbf{f}_R)$

6: Compute:

$$\mathbf{y}_1 = \widehat{\mathbf{f}}_L + \widehat{\mathbf{f}}_M + \widehat{\mathbf{f}}_R$$

$$\mathbf{y}_2 = \widehat{\mathbf{f}}_L + \widehat{\mathbf{f}}_R + (\widehat{\mathbf{f}}_M + \widehat{\mathbf{f}}_R)\alpha$$

$$\mathbf{y}_3 = \widehat{\mathbf{f}}_L + \widehat{\mathbf{f}}_M + (\widehat{\mathbf{f}}_M + \widehat{\mathbf{f}}_R)\alpha$$

7: Return $\widehat{\mathbf{f}} = (\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3)$

8: **end if**

Proof. The proof is similar to that of [Theorem 3.1](#). The only difference is the number of additions/subtraction in Step 6, which is listed as follows:

$$\mathbf{t}_1 := \widehat{\mathbf{f}}_M + \widehat{\mathbf{f}}_R \quad \mathbf{t}_2 := \mathbf{t}_1 \cdot \alpha \quad \mathbf{t}_3 := \mathbf{t}_2 + \widehat{\mathbf{f}}_L$$

$$\mathbf{y}_1 := \mathbf{t}_1 + \widehat{\mathbf{f}}_L \quad \mathbf{y}_2 := \mathbf{t}_3 + \widehat{\mathbf{f}}_R \quad \mathbf{y}_3 := \mathbf{t}_3 + \widehat{\mathbf{f}}_M$$

Then we have that $T_n^+ = 3T_{n-1}^+ + 5 \cdot 3^{n-1}$ and $T_n^\times = 3T_{n-1}^\times + 3^{n-1}$. By applying this result recursively, we conclude that $T_n^+ = \frac{5}{3} \cdot n \cdot 3^n = \frac{5}{3}N \log_3 N$ and $T_n^\times = n \cdot 3^{n-1} = \frac{1}{3}N \log_3 N$. $\square$

Remark B.2. Note that the multiplication over $\mathbb{F}_4$ can be replaced by a table lookup, because the table for $\mathbb{F}_4$ multiplication is simple.

Remark B.3. By setting $\ell = 1$, we have that $1 \equiv -1 \pmod{2}$ and $3^{-1} \equiv 1 \pmod{2}$, thus [Algorithm 1](#) and [Algorithm 7](#) are equivalent to [Algorithm 5](#) and [Algorithm 6](#), respectively.

In a similar way, we could define an inverse FFT algorithm over $\mathbb{F}_4$, which corresponds to the multivariate polynomial interpolation. For $f \in \mathcal{R}_2 = \mathbb{F}_4[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$ and $N = 3^n$, given the evaluations $\widehat{\mathbf{f}} \in \mathbb{F}_4^N$ at $\{1, \alpha, \alpha + 1\}^n$, we

can compute the coefficients $\mathbf{f}$ as follows:

$$\mathbf{f} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha^2 & \alpha \\ 1 & \alpha & \alpha^2 \end{pmatrix}^{\otimes n} \quad \widehat{\mathbf{f}} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha + 1 & \alpha \\ 1 & \alpha & \alpha + 1 \end{pmatrix}^{\otimes n} \widehat{\mathbf{f}} \quad (4)$$

The correctness of this IFFT algorithm is due to

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha + 1 & \alpha \\ 1 & \alpha & \alpha + 1 \end{pmatrix}^{\otimes n} \begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha + 1 \\ 1 & \alpha + 1 & \alpha \end{pmatrix}^{\otimes n} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}^{\otimes n} = I_{3^n} \quad (5)$$

where I_{3^n} denotes a $3^n \times 3^n$ identity matrix. The first equation in [Eq.\(5\)](#) holds since $A^{\otimes n} B^{\otimes n} = (AB)^{\otimes}$. Note that the matrix in [Eq.\(3\)](#) has a similar form as that in [Eq.\(4\)](#), thus we can follow [Algorithm 5](#) to instantiate IFFT over $\mathbb{F}_4$, which is shown in [Algorithm 6](#).

Algorithm 6 Inverse Fast Fourier Transform over $\mathbb{F}_4$: $\text{IFFT}_{\mathbb{F}_4}$

Input: Evaluations $\widehat{\mathbf{f}} \in \mathbb{F}_4^N$ of polynomial $f \in \mathcal{R}_2 = \mathbb{F}_4[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$ at $\{1, \alpha, \alpha + 1\}^n$, where $N = 3^n$.

Output: Polynomial f

- 1: **if** $n = 0$ **then**
- 2: Return $\widehat{\mathbf{f}}$
- 3: **else**
- 4: Split $\widehat{\mathbf{f}}$ into three vectors of length $m = 3^{n-1}$:

$$\begin{aligned} \widehat{\mathbf{f}}_L &= (\widehat{f}_0, \widehat{f}_1, \dots, \widehat{f}_{m-1}) \\ \widehat{\mathbf{f}}_M &= (\widehat{f}_m, \widehat{f}_{m+1}, \dots, \widehat{f}_{2m-1}) \\ \widehat{\mathbf{f}}_R &= (\widehat{f}_{2m}, \widehat{f}_{2m+1}, \dots, \widehat{f}_{3m-1}) \end{aligned}$$

- 5: Compute $\mathbf{f}_L = \text{IFFT}_{\mathbb{F}_4}(\widehat{\mathbf{f}}_L)$, $\mathbf{f}_M = \text{IFFT}_{\mathbb{F}_4}(\widehat{\mathbf{f}}_M)$ and $\mathbf{f}_R = \text{IFFT}_{\mathbb{F}_4}(\widehat{\mathbf{f}}_R)$
- 6: Compute:

$$\begin{aligned} \mathbf{y}_1 &= \mathbf{f}_L + \mathbf{f}_M + \mathbf{f}_R \\ \mathbf{y}_2 &= \mathbf{f}_L + \mathbf{f}_M + (\mathbf{f}_M + \mathbf{f}_R)\alpha \\ \mathbf{y}_3 &= \mathbf{f}_L + \mathbf{f}_R + (\mathbf{f}_M + \mathbf{f}_R)\alpha \end{aligned}$$

- 7: Return $f \in \mathcal{R}_2$ with coefficients $\mathbf{f} = (\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3) \in \mathbb{F}_4^N$
 - 8: **end if**
-

For completeness, we present full-fledged algorithms for IFFT over degree-2 Galois rings $\text{GR}_2 = \mathbb{GR}(2^\ell, 2)$ and general Galois rings $\text{GR}_k = \mathbb{GR}(2^\ell, k)$ in [Algorithm 7](#) and [Algorithm 8](#), respectively.

Algorithm 7 Inverse Fast Fourier Transform over GR_2 : $\text{IFFT}_{\text{GR}_2}$

Input: Evaluations $\widehat{\mathbf{f}} \in \text{GR}_2^N$ of polynomial $f \in \mathcal{R}_2 = \text{GR}_2[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$ at $\{1, \beta, -\beta - 1\}^n$, where $N = 3^n$.

Output: Polynomial $3^n \cdot f$

- 1: **if** $n = 0$ **then**
- 2: Return $\widehat{\mathbf{f}}$
- 3: **else**
- 4: Split $\widehat{\mathbf{f}}$ into three vectors of length $m = 3^{n-1}$:

$$\widehat{\mathbf{f}}_L = (\widehat{f}_0, \widehat{f}_1, \dots, \widehat{f}_{m-1})$$

$$\widehat{\mathbf{f}}_M = (\widehat{f}_m, \widehat{f}_{m+1}, \dots, \widehat{f}_{2m-1})$$

$$\widehat{\mathbf{f}}_R = (\widehat{f}_{2m}, \widehat{f}_{2m+1}, \dots, \widehat{f}_{3m-1})$$

- 5: Compute $\mathbf{f}_L = \text{IFFT}_{\text{GR}_2}(\widehat{\mathbf{f}}_L)$, $\mathbf{f}_M = \text{IFFT}_{\text{GR}_2}(\widehat{\mathbf{f}}_M)$ and $\mathbf{f}_R = \text{IFFT}_{\text{GR}_2}(\widehat{\mathbf{f}}_R)$
- 6: Compute:

$$\mathbf{y}_1 = \mathbf{f}_L + \mathbf{f}_M + \mathbf{f}_R$$

$$\mathbf{y}_2 = \mathbf{f}_L - \mathbf{f}_M - (\mathbf{f}_M - \mathbf{f}_R)\beta$$

$$\mathbf{y}_3 = \mathbf{f}_L - \mathbf{f}_R + (\mathbf{f}_M - \mathbf{f}_R)\beta$$

- 7: Return $f \in \mathcal{R}_2$ with coefficients $\mathbf{f} = (\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3) \in \text{GR}_2^N$
 - 8: **end if**
-

Algorithm 8 Inverse Fast Fourier Transform over GR_k : $\text{IFFT}_{\text{GR}_k}$

Input: Evaluation vector $\widehat{\mathbf{f}} \in \text{GR}_k^N$ of $f \in \mathcal{R} = \text{GR}_k[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$ at all points in $\{1, \beta, -\beta - 1\}^n$ with $N = 3^n$.

Output: Polynomial f

- 1: **for** $i \in [0, N - 1]$ **do**
 - 2: Decompose evaluation $\widehat{f}_i$ as $\sum_{j \in [0, k/2-1]} \widehat{f}_{i,j} \gamma^j$, where $\{1 = \gamma^0, \gamma, \dots, \gamma^{k/2-1}\}$ forms a basis of GR_k over $\text{GR}_2 = \mathbb{GR}(2^\ell, 2)$.
 - 3: **end for**
 - 4: **for** $j \in [0, k/2 - 1]$ **do**
 - 5: Define $\widehat{\mathbf{f}}^{(j)} = (\widehat{f}_{0,j}, \dots, \widehat{f}_{N-1,j})$.
 - 6: Compute $f^{(j)} = \text{IFFT}_{\text{GR}_2}(\widehat{\mathbf{f}}^{(j)})$.
 - 7: **end for**
 - 8: Return $f = \sum_{j \in [0, k/2-1]} f^{(j)} \cdot 3^{-n} \cdot \gamma^j$.
-

C Missing Proofs

C.1 Proof of PCG for VOLE

Theorem C.1 (Theorem 4.1). *Assume a secure PPRF scheme for sums of point functions and $QA\text{-}SD(\mathbb{F}_{2^k}, t)$ is hard. Then there exists a PCG construction $\text{Cons}_{\text{VOLE}}^{\mathbb{F}_{2^k}}$ that generates VOLE correlations over $\mathbb{F}_{2^k}$. If the PPRF is based on a PRG : $\{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda}$ via the RPG-based construction from [11], for generating $N = 3^n$ VOLE correlations over $\mathbb{F}_{2^k}$, we have that:*

- For party with $\sigma = 0$, the seed size is at most: $2t(\lambda \lceil \log_2 \frac{N}{t} \rceil + \lceil \log_2 N \rceil + k)$ bits.
- For party with $\sigma = 1$, the seed size is at most: $2t(\lambda + k)$ bits.
- For party with $\sigma = 0$, the computation of **Expand** can be done with at most $(4 + \lceil \frac{k}{\lambda} \rceil)N$ RPG operations, $\frac{10}{3}kN \log_3 N$ additions and $\frac{2}{3}kN \log_3 N$ multiplications over $\mathbb{F}_4$, and $2N$ additions and $2N$ multiplications over $\mathbb{F}_{2^k}$.
- For party with $\sigma = 1$, the computation of **Expand** can be done with at most $(4 + \lceil \frac{k}{\lambda} \rceil)N$ RPG operations, $\frac{5}{3}kN \log_3 N$ additions and $\frac{1}{3}kN \log_3 N$ multiplications over $\mathbb{F}_4$, and N additions and N multiplications over $\mathbb{F}_{2^k}$.

Proof. We assume the QA-SD assumption holding for regular noise distribution to optimize our parameters. We follow [11] to estimate the seed size and PRG calls of PPRF. The correctness is straightforward and the security analysis is based on the a QA-SD assumption and the underlying PPRF scheme. Let $\mathbf{k}_0, \mathbf{k}_1 \xleftarrow{\$} \text{PCG.Gen}$ with associated expanded outputs $(\mathbf{x}, \mathbf{z}_0)$ and $(\Delta, \mathbf{z}_1)$. We first consider the view of P_0 , i.e., $\sigma = 0$. We need to show that:

$$\{(\mathbf{k}_0, (\Delta, \mathbf{z}_1))\} \equiv \left\{ (\mathbf{k}_0, (\tilde{\Delta}, \tilde{\mathbf{z}}_1)) \mid \begin{array}{l} \tilde{\Delta} \xleftarrow{\$} \mathbb{F}_{2^k}, \\ \tilde{\mathbf{z}}_1 = -\mathbf{z}_0 + \tilde{\Delta} * \mathbf{x} \end{array} \right\}$$

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

- Replace $\mathbf{z}_1$ by $\Delta * \mathbf{x} - \mathbf{z}_0$.
- Replace punctured PPRF keys $\mathbf{k}_s^A, \mathbf{k}_e^B$ 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 keys are independent of the output, we can then replace Δ with a fresh $\tilde{\Delta}$.
- Reverse step 2 by using PPRF security property once again.

Similarly, we consider the view of P_1 as follows:

$$\{(\mathbf{k}_1, (\mathbf{x}, \mathbf{z}_0))\} \equiv \left\{ (\mathbf{k}_1, (\tilde{\mathbf{x}}, \tilde{\mathbf{z}}_0)) \mid \begin{array}{l} \tilde{\mathbf{x}} \xleftarrow{\$} \mathbb{F}_{2^k}^N, \\ \tilde{\mathbf{z}}_1 = \Delta * \tilde{\mathbf{x}} - \mathbf{z}_0 \end{array} \right\}$$

- Replace $\mathbf{z}_0$ by $\Delta * \mathbf{x} - \mathbf{z}_1$.

- Replace PPRF keys $\mathbf{k}_s, \mathbf{k}_e$ in $\mathbf{k}_1$ 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 keys are independent of the output, we can then replace $\mathbf{x}$ with a fresh $\tilde{\mathbf{x}}$ by the QA-SD assumption.
- Reverse step 2 by using PPRF security property once again.

□

C.2 Proof of PCG for OLE

Theorem C.2 (Theorem 4.3). *Assume a secure FSS scheme SPFSS for sums of points functions and QA-SD($\mathbb{F}_{2^k}, t$) is hard. Then there exists a PCG construction $\text{Cons}_{\text{OLE}}^{\mathbb{F}_{2^k}}$ that generates OLE correlations over $\mathbb{F}_{2^k}$. If the SPFSS is based on a PRG : $\{0, 1\}^\lambda \rightarrow \{0, 1\}^{3(\lambda+1)}$ via the PRG-based construction from [4], for generating $N = 3^n$ OLE correlations over $\mathbb{F}_{2^k}$, we have that:*

- The seed size of each party is around $4t^2 (3(\lambda + 1) \log_3 \frac{N}{t} + \lambda + k) + 2t(\lceil \log_2 N \rceil + k)$ bits.
- The computation of **Expand** can be done with at most $(1.5 + \lceil \frac{k}{\lambda} \rceil)4Nt$ PRG operations, $5kN \log_3 N$ additions and $kN \log_3 N$ multiplications over $\mathbb{F}_4$, and $4N$ additions and $3N$ multiplications over $\mathbb{F}_{2^k}$.

Proof. Regarding the size of the different parameters, we also follow optimizations of [14, 5, 4], e.g., assuming the QA-SD assumption holds also for regular noise distributions. As for SPFSS scheme, since our domain size is $N = 3^n$, we follow the protocol of [4] to estimate the seed size and PRG calls of ternary DPF.

Correctness is straightforward given by the Construction $\text{Cons}_{\text{OLE}}^{\mathbb{F}_{2^k}}$. Security is based on a QA-SD assumption and the underlying SPFSS scheme, and we prove in a similar way to those of [14, 5, 4]. For completeness, we give a sketched proof as follows. It suffices to consider Party P_0 's view (i.e., $\sigma = 0$). Let $(\mathbf{k}_0, \mathbf{k}_1) \xleftarrow{\$} \text{PCG.Gen}(1^\lambda)$ with associated expanded outputs $(\mathbf{x}_0, \mathbf{z}_0)$ and $(\mathbf{x}_1, \mathbf{z}_1)$. We need to show that

$$\{(\mathbf{k}_0, (\mathbf{x}_1, \mathbf{z}_1))\} \equiv \left\{ (\mathbf{k}_0, (\tilde{\mathbf{x}}_1, \tilde{\mathbf{z}}_1)) \mid \begin{array}{l} \tilde{\mathbf{x}}_1 \xleftarrow{\$} \mathbb{F}_{2^k}^N, \\ \tilde{\mathbf{z}}_1 = \mathbf{x}_0 * \tilde{\mathbf{x}}_1 - \mathbf{z}_0 \end{array} \right\}$$

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 2.4](#) implies that $\mathbf{x}_1$ computed from a random $x_1 \in \mathcal{R}$ are uniformly distributed in $\mathbb{F}_{2^k}^N$. $\square$

C.3 Proof of Piling-up Lemma

Lemma C.3 (Lemma 6.5). *For any $0 < r < 1 - 1/q$ and any integer n , given n random variables $X_1, \dots, X_n$ i.i.d to $\text{Ber}_r(\mathbb{F}_q)$, it holds that $\Pr[\sum_{i=1}^n X_i = 0] = \frac{1}{q} + \frac{q-1}{q} \cdot (1 - \frac{q}{q-1} \cdot r)^n$. Moreover, for any $y \in \mathbb{F}_q \neq 0$, $\Pr[\sum_{i=1}^n X_i = y] = \frac{1}{q} - \frac{1}{q} \cdot (1 - \frac{q}{q-1} \cdot r)^n$.*

Proof. We prove by deduction. Let $P_n = \Pr[\sum_{i=1}^n X_i = 0]$. It immediately follows that $P_1 = 1 - r$. Since $X_1, \dots, X_n$ are sampled independently, we have that

$$P_{n+1} = P_n \cdot (1 - r) + (1 - P_n) \cdot \frac{r}{q-1} = P_n \cdot (1 - \frac{q}{q-1} \cdot r) + \frac{r}{q-1}.$$

Hence, $P_{n+1} - \frac{1}{q} = (P_n - \frac{1}{q}) \cdot (1 - \frac{q}{q-1} \cdot r)$, and

$$P_n = (P_1 - \frac{1}{q}) \cdot (1 - \frac{q}{q-1} \cdot r)^{n-1} + \frac{1}{q} = \frac{1}{q} + \frac{q-1}{q} \cdot (1 - \frac{q}{q-1} \cdot r)^n.$$

For any $y \in \mathbb{F}_q \neq 0$, it follows by symmetry that $\Pr[\sum_{i=1}^n X_i = y] = \frac{1-P_n}{q-1}$. $\square$

D PCG for OLE over Integer Rings

In this section, we follow [\[46\]](#) to construct PCG for OLE over an integer ring $\mathbb{Z}_{2^\ell}$. This construction is based on QA-SD assumption over a Galois ring $\text{GR}_k = \mathbb{GR}(2^\ell, k)$ for an even k , which is a ring extension of $\mathbb{Z}_{2^\ell}$. Our construction relies on a trace function over a polynomial ring $\mathcal{R} = \text{GR}_k[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$, which acts as a bridge between $\mathbb{Z}_{2^\ell}$ and GR_k . For completeness, we present the definition of generalized Frobenius map and trace function over a multivariate polynomial ring as follows.

Definition D.1 (Generalized Frobenius map). *Let $\text{GR}_k = \mathbb{GR}(2^\ell, k)$ and let $\mathcal{R} = \text{GR}_k[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$. For $f \in \mathcal{R}$, f can be uniquely represented in the coefficient form*

$$f = \sum_{(i_1, \dots, i_n) \in \{0,1,2\}^n} f_{i_1, \dots, i_n} \mathbf{x}_1^{i_1} \cdots \mathbf{x}_n^{i_n}$$

The generalized Frobenius map $\tau_{\mathcal{R}} : \mathcal{R} \rightarrow \mathcal{R}$ is defined by

$$\tau_{\mathcal{R}}(f) = \sum_{(i_1, \dots, i_n) \in \{0,1,2\}^n} \tau_{\text{GR}_k}(f_{i_1, \dots, i_n}) \mathbf{x}_1^{2 \cdot i_1 \bmod 3} \cdots \mathbf{x}_n^{2 \cdot i_n \bmod 3}$$

where τ_{GR_k} denotes the generalized Frobenius automorphism for Galois ring GR_k .

Definition D.2 (Trace function over multivariate polynomial). For $\text{GR}_k = \mathbb{GR}(2^\ell, k)$ and $\mathcal{R} = \mathbb{GR}_k[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$, the trace function $\text{Tr}_{\mathcal{R}} : \mathcal{R} \rightarrow \mathcal{R}$ is defined as:

$$\text{Tr}_{\mathcal{R}}(f) = \sum_{j=0}^{k-1} \tau_{\mathcal{R}}^j(f)$$

where $\tau_{\mathcal{R}}^j = \tau_{\mathcal{R}} \circ \tau_{\mathcal{R}}^{j-1}$ denotes the composition of $\tau_{\mathcal{R}}$.

Assume that each party P_σ holds a QA-SD instance $((s_\sigma, e_\sigma), b_\sigma := as_\sigma + e_\sigma)$, where $a \xleftarrow{\$} \mathcal{R}$ and s_σ, e_σ are t -sparse elements of $\mathcal{R}$. At a high level, our goal is to share the product of $\text{Tr}_{\mathcal{R}}(b_0) \cdot \text{Tr}_{\mathcal{R}}(b_1)$, and then locally convert the relation over $\mathcal{R}$ to multiple OLE instances over $\mathbb{Z}_{2^\ell}$. Furthermore, we can extract more OLE instances over $\mathbb{Z}_{2^\ell}$ by computing the product of $\text{Tr}_{\mathcal{R}}(\zeta^{2^j} b_0) \cdot \text{Tr}_{\mathcal{R}}(\zeta^{2^j} b_1)$ for each $j \in [0, k-1]$, where ζ is the generator of GR_k such that $\text{GR}_k = \mathbb{Z}_{2^\ell}(\zeta)$. Following [46], we compute each $\text{Tr}_{\mathcal{R}}(\zeta^{2^j} b_0) \cdot \text{Tr}_{\mathcal{R}}(\zeta^{2^j} b_1)$ with Eq.(6).

$$\begin{aligned} & \text{Tr}_{\mathcal{R}}(\zeta^{2^j} b_0) \cdot \text{Tr}_{\mathcal{R}}(\zeta^{2^j} b_1) \\ &= \sum_{i=0}^{k-1} \text{Tr}_{\mathcal{R}}(\zeta^{2^j} \cdot (as_0 + e_0) \cdot \tau_{\mathcal{R}}^i(\zeta^{2^j}) \cdot (as_1 + e_1)) \\ &= \sum_{i=0}^{k-1} \text{Tr}_{\mathcal{R}}(\zeta^{2^j} \cdot (as_0 + e_0) \cdot \tau_{\mathcal{R}}^i(\zeta^{2^j}) \cdot \tau_{\mathcal{R}}^i(as_1 + e_1)) \\ &= \sum_{i=0}^{k-1} \text{Tr}_{\mathcal{R}}\left(\zeta^{2^j+2^{j+i}} \cdot (a\tau_{\mathcal{R}}^i(s_0)\tau_{\mathcal{R}}^i(s_1) + \tau_{\mathcal{R}}^i(a)e_0\tau_{\mathcal{R}}^i(s_1) + as_0\tau_{\mathcal{R}}^i(e_1) + e_0\tau_{\mathcal{R}}^i(e_1))\right) \end{aligned} \tag{6}$$

Note that there exist $4k$ cross-terms of

$$(s_0 \cdot \tau_{\mathcal{R}}^i(s_1), e_0 \cdot \tau_{\mathcal{R}}^i(s_1), s_0 \cdot \tau_{\mathcal{R}}^i(e_1), e_0 \cdot \tau_{\mathcal{R}}^i(e_1))$$

which can be computed with SPFSS scheme. We present a self-contained construction in $\text{Cons}_{\text{OLE}}^{\mathbb{Z}_{2^\ell}}$.

Construction 3: $\text{Cons}_{\text{OLE}}^{\mathbb{Z}_{2^\ell}}$

PARAMETER: Security parameter λ , Hamming weight $t = t(\lambda)$, $N = 3^n$. Let Galois ring $\text{GR}_k = \mathbb{GR}(2^\ell, k) = \mathbb{Z}_{2^\ell}/(f(x))$, where $f(x)$ is the Hensel lift of a primitive polynomial of an even degree k over $\mathbb{F}_2$. Let $\zeta \in \text{GR}$ be the root of $f(x)$ with order $2^k - 1$. Then $\text{GR}_k = \mathbb{Z}_{2^\ell}(\zeta)$ and define $\mathcal{R} = \text{GR}_k[\mathbf{x}_1, \dots, \mathbf{x}_n]/(\mathbf{x}_1^3 - 1, \dots, \mathbf{x}_n^3 - 1)$. An FSS scheme (SPFSS.Gen, SPFSS.FullEval) for sums of t^2 point functions, with domain $[0, N]^a$ and range GR_k .

PUBLIC INPUT: A uniformly random $a \in \mathcal{R}$ and its evaluation $\hat{\mathbf{a}} = \text{FFT}_{\text{GR}_k}(a)$.

CORRELATION: After expansion, outputs $(\mathbf{x}_0, \mathbf{z}_0) \in \mathbb{Z}_{2^\ell}^{kN} \times \mathbb{Z}_{2^\ell}^{kN}$ and $(\mathbf{x}_1, \mathbf{z}_1) \in \mathbb{Z}_{2^\ell}^{kN} \times \mathbb{Z}_{2^\ell}^{kN}$ such that $\mathbf{x}_0 \star \mathbf{x}_1 = \mathbf{z}_0 + \mathbf{z}_1$.

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 $\mathbf{s}_\sigma, \mathbf{e}_\sigma \leftarrow \text{GR}_k^t$.
2. For $i \in [0, k-1]$, sample FSS keys:

$$(K_0^{4i}, K_1^{4i}) \xleftarrow{\$} \text{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0 \boxplus (\mathbf{A}_1^0)^{2^i}, \mathbf{s}_0 \otimes \tau_{\mathcal{R}}^i(\mathbf{s}_1)),$$

$$(K_0^{4i+1}, K_1^{4i+1}) \xleftarrow{\$} \text{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1 \boxplus (\mathbf{A}_1^0)^{2^i}, \mathbf{e}_0 \otimes \tau_{\mathcal{R}}^i(\mathbf{s}_1)),$$

$$(K_0^{4i+2}, K_1^{4i+2}) \xleftarrow{\$} \text{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0 \boxplus (\mathbf{A}_1^1)^{2^i}, \mathbf{s}_0 \otimes \tau_{\mathcal{R}}^i(\mathbf{e}_1)),$$

$$(K_0^{4i+3}, K_1^{4i+3}) \xleftarrow{\$} \text{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1 \boxplus (\mathbf{A}_1^1)^{2^i}, \mathbf{e}_0 \otimes \tau_{\mathcal{R}}^i(\mathbf{e}_1)),$$
3. For $\sigma \in \{0, 1\}$, let $\mathbf{k}_\sigma = \left((K_\sigma^i)_{i \in [0, 4k-1]}, (\mathbf{A}_\sigma^0, \mathbf{s}_\sigma), (\mathbf{A}_\sigma^1, \mathbf{e}_\sigma) \right)$.
4. Output $(\mathbf{k}_0, \mathbf{k}_1)$.

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

1. Parse $\mathbf{k}_\sigma$ as $\left((K_\sigma^i)_{i \in [0, 4k-1]}, (\mathbf{A}_\sigma^0, \mathbf{s}_\sigma), (\mathbf{A}_\sigma^1, \mathbf{e}_\sigma) \right)$.
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 $i \in [0, 4k-1]$, compute $u_{\sigma,i} \leftarrow \text{SPFSS.FullEval}(\sigma, K_\sigma^i)$, viewed as $\mathcal{R}$ elements.
4. Compute $\widehat{\mathbf{u}}_{\sigma,i} = \text{FFT}_{\text{GR}_k}(u_{\sigma,i})$ for $i \in [0, 4k-1]$, $\widehat{\mathbf{s}}_\sigma = \text{FFT}_{\text{GR}_k}(s_\sigma)$ and $\widehat{\mathbf{e}}_\sigma = \text{FFT}_{\text{GR}_k}(e_\sigma)$.
5. For $i \in [0, k-1]$, compute^b

$$\mathbf{y}_\sigma^{(i)} = \widehat{\mathbf{a}} \star \tau_{\text{GR}_k^N}^i(\widehat{\mathbf{a}}) \star \widehat{\mathbf{u}}_{\sigma,4i} + \tau_{\text{GR}_k^N}^i(\widehat{\mathbf{a}}) \star \widehat{\mathbf{u}}_{\sigma,4i+1} + \widehat{\mathbf{a}} \star \widehat{\mathbf{u}}_{\sigma,4i+2} + \widehat{\mathbf{u}}_{\sigma,4i+3}$$

6. For $j \in [0, k-1]$, compute

$$\mathbf{x}_\sigma^{(j)} = \text{Tr}_{\text{GR}_k^N} \left(\zeta^{2^j} \cdot (\widehat{\mathbf{a}} \star \widehat{\mathbf{s}}_\sigma + \widehat{\mathbf{e}}_\sigma) \right)$$

$$\mathbf{z}_\sigma^{(j)} = \sum_{i \in [0, k-1]} \text{Tr}_{\text{GR}_k^N} \left(\zeta^{2^j + 2^{j+i}} \cdot \mathbf{y}_\sigma^{(i)} \right)$$

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

^a Each j corresponds to a basis $\prod_{i \in [m]} \mathbf{x}_i^{j_i}$ of $\mathcal{R}$, where $j_i \in \{0, 1, 2\}$. The addition on $[0, N]$ is actually the addition on $\mathbb{G} = \mathbb{Z}_3^n$.

^b We can prepare $\tau_{\text{GR}_k^N}^j(\widehat{\mathbf{a}})$ and $\tau_{\text{GR}_k^N}^j(\widehat{\mathbf{a}}) \star \widehat{\mathbf{a}}$ for $i \in [0, k-1]$ in advance.

Theorem D.3. Assume a secure FSS scheme SPFSS for sums of points functions and QA-SD(GR_k, t) is hard for $\text{GR}_k = \mathbb{GR}(2^\ell, k)$. Then there exists a PCG

construction $\text{Cons}_{\text{OLE}}^{\mathbb{Z}_{2^\ell}}$ that generates OLE correlations over $\mathbb{Z}_{2^\ell}$. If the SPFSS is based on a PRG : $\{0, 1\}^\lambda \rightarrow \{0, 1\}^{3(\lambda+1)}$ via the PRG-based construction from [4], for generating $N = k \cdot 3^n$ OLE correlations over $\mathbb{Z}_{2^\ell}$, we have that:

- The seed size of each party is around $4kt^2 (3(\lambda + 1) \log_3 \frac{N}{t} + \lambda + k\ell) + 2t(\lceil \log_2 N \rceil + k\ell)$ bits.
- The computation of **Expand** can be done with at most $(1.5 + \lceil \frac{k}{\lambda} \rceil)4Ntk$ PRG operations, $\frac{7}{3}(2k + 1)N \log_3 N$ additions and $\frac{1}{3}(2k + 1)N \log_3 N$ multiplications over $\mathbb{GR}(2^\ell, 2)$, and $O(kN)$ operations over $\mathbb{GR}_k$.

Proof. We follow Theorem 4.3 to calculate the seed size and computation complexity of our PCG construction. Note that compared to the construction of [46], the main difference is that we instantiate the isomorphism $\phi : \mathcal{R} \rightarrow \mathbb{GR}_k^N$ with our efficient FFT algorithm. The security analysis is the same as that in [46], since the FFT algorithm is only applied in the local computation. $\square$