

Pseudorandom Correlation Generators for Matrix Triples

Maxime Bombar* Geoffroy Couteau† Nikolas Melissaris‡ Anuja Modi§

Abstract

We introduce new pseudorandom correlation generators (PCGs) for matrix triples: shares of triples $(\mathbf{A}, \mathbf{B}, \mathbf{A} \cdot \mathbf{B})$ where $\mathbf{A}, \mathbf{B}$ are random matrices over $\mathbb{F}^{n \times n}$ for a field $\mathbb{F}$ and an integer n . Matrix triples are consumed in a wide variety of secure computation protocols that involve matrix arithmetic, which occurs commonly in settings such as privacy-preserving machine-learning and statistics, threshold cryptography, secure linear algebra, and more. Previous PCGs for matrix triples are inefficient: their seed grows at least linearly in the matrix dimension, and their expansion is slow. In this work, we introduce two new PCGs with improved performances:

- Our first construction, inspired by the recent work of (Gentry and Lee, Crypto 2026) on FHE for matrix arithmetic, relies on the quasi-abelian syndrome decoding (QASD) assumption introduced in (Bombar et al., Crypto 2023), an assumption which has since received a significant amount of attention. Our PCG generates N matrix triples over $\mathbb{F}^{n \times n}$ from a seed of size $\text{poly}(\lambda, \log n, \log N)$ using $\Theta(n^3 \log(n^3 N))$ field operations per triple, removing the linear dependency on n from the seed compared to previous works.
- Our second construction is more mathematically involved, and builds upon an elegant and surprising connection with the Weyl-Pauli twisted algebra, a mathematical structure originally studied in (Weyl, Zeitschrift für Physik, 1927) in the context of quantum kinematics. Our PCG relies on a new variant of the QASD assumption over the Weyl-Pauli twisted group algebra and exhibits impressive performances: it has the same seed size as our first construction and generates N matrix triples at a cost only a small constant time larger than the cost of computing N matrix triples *in the clear*.

We complement our results with an in-depth study of our new *twisted* QASD assumption and demonstrate the benefits of our PCGs in concrete applications such as privacy-preserving machine learning and threshold MAYO.

*Université de Bordeaux, maxime.bombar@math.u-bordeaux.fr

†CNRS, IRIF, couteau@irif.fr

‡CNRS, IRIF, nikolas@irif.fr

§IIT Madras, India, anujamodi97@gmail.com (Work initiated during a visit to IRIF, CNRS.)

Contents

1	Introduction	3
1.1	Our Contributions	4
1.2	Our Approach	5
1.3	Applications	6
1.4	Technical Overview	7
1.5	Organization	12
2	Preliminaries	12
2.1	Notations	12
2.2	Cryptographic Primitives	13
2.3	Group Algebras	17
2.4	Encoding and Decoding of Matrices	24
3	PCGs for Matrix Multiplication from QA-SD	25
3.1	Construction	26
3.2	Security Proof	29
3.3	Efficiency analysis of the QA-SD construction	30
4	PCGs for Matrix Multiplication from Twisted QA-SD	31
4.1	Construction	32
4.2	Security	36
4.3	Efficiency analysis of the twisted construction	40
5	Attacks by Folding	45
5.1	Folding Group Algebras	45
5.2	Obstructions for Twisted Group Algebras	46
5.3	Characterization of the Obstruction via Group Cohomology	48
5.4	Folding as a Projector	50
5.5	Extending Folding to Twisted Group Algebras	52
6	Optimizations and Efficiency Estimation	58
6.1	A common cost model	58
6.2	Elementary abelian index and batch groups	60
6.3	Instantiating the sum-of-points FSS	63
6.4	Parameter selection	65
6.5	Concrete estimates	66
7	Applications	70
7.1	Privacy-Preserving Machine Learning	71
7.2	Batched and Iterative Matrix Computations	72
7.3	Threshold MAYO	73

1 Introduction

Modern secure computation protocols (often abbreviated MPC, for MultiParty Computation) often rely on correlated randomness, generated in a preprocessing phase, to achieve a lightweight online phase. Over the past decade, pseudorandom correlation generators (PCG) [BCGI18, BCG⁺19b, BCG⁺19a] have emerged as a leading paradigm to distribute correlated randomness: following a low-overhead distribution of short correlated seeds, the participants can stretch them *silently* into large amounts of pseudorandom correlations, using only local work and no further interaction. Depending on the security model, the number of participants, and the target function, secure computation protocols can consume a wide variety of correlations. Over the past few years, increasingly efficient PCGs have been introduced for most of the leading correlations used in generic MPC, including vector oblivious linear evaluation (VOLE) [BCGI18], random oblivious transfers (OT) [BCG⁺19b, BCG⁺19a, CRR21, BCG⁺22, RRT23, KPRR25, KPR⁺26], oblivious linear evaluation (OLE) and (authenticated) Beaver triples [BCG⁺20b, AS22, BCCD23, BBC⁺24a, LXYY25a, MMST25, ARR25], as well as their generalizations over rings [LXYY25b, LLX⁺26], and other more exotic forms of correlated randomness [ABG⁺24, CDKSS25, CKM⁺26].

Perhaps one of the most fundamental correlations in applied secure computation is *matrix triples*: shares of a triple $(\mathbf{A}, \mathbf{B}, \mathbf{A} \cdot \mathbf{B})$ where $\mathbf{A}, \mathbf{B}$ are random matrices over $\mathbb{F}^{n \times n}$ for a field $\mathbb{F}$ and an integer n . Matrix triples enable secure multiplication of private n -by- n (shared) matrices $(\mathbf{U}, \mathbf{V})$ using low communication ($2n^2$ per party, the same cost as communicating their shares in the clear) and computation (2 matrix multiplications per party), using the natural generalization of Beaver’s reduction of random OT to OT [Bea92]. Matrix multiplication shows up everywhere: in secure machine learning, where it dominates the cost of inference [JVC18, KRC⁺20, HJLHD22], training [MZ17, MR18, WGC19, WWP22], graph neural networks [RXW⁺22, WZJ23, PRL⁺23] and, most recently, transformer and LLM evaluation [HLC⁺22, GJM⁺24, PZM⁺24, jLHG⁺25]; in secure linear algebra, where solving linear systems and computing determinants, ranks, inverses and pseudo-inverses all reduce to iterated matrix products [BIB89, CD01, NW06, CKP07, MW08, MV24]; in threshold post-quantum signatures such as UOV and MAYO, whose key generation and signing can be rewritten entirely in terms of matrix operations [CS19, CEN25]; and in privacy-preserving statistics and computational biology, such as ridge and linear regression [NWI⁺13, GSB⁺17], matrix factorization for recommender systems [NIW⁺13], and PCA-corrected genome-wide association studies [CWB18, BGPG20, CFC⁺25].

Consequently, a considerable effort has been devoted to designing efficient protocols to securely compute matrix triples. Yet, despite the extensive body of work on the topic and the remarkable progress over the past few years, existing protocols remain too expensive. Protocols built from standard secure computation techniques are communication-heavy: obtaining an $n \times n$ triple from n^3 scalar multiplication triples, or from oblivious transfer via Gilboa’s protocol [Gil99] even with the vectorization tricks of SecureML [MZ17], costs communication cubic in n , roughly $n^3 \ell^2$ bits for ℓ -bit entries. Concretely, a single 128×128 triple over a 64-bit ring costs about 1 GB of communication in the semi-honest setting, and about 47 GB per party with the authenticated triples of MASCOT [KOS16] over a 128-bit field; SecureML reports an OT-based offline phase of 19 GB for a linear regression on a $10^4 \times 10^3$ dataset, and of 80 hours (against 1.2 hours online) for training a small neural network on MNIST [MZ17], and Chen et al. estimate the SPDZ preprocessing of the convolutions of ResNet-50 at 5 TB per party [CKR⁺20]. Homomorphic encryption brings the communication down to a small multiple of the triple itself: the generators of [CKR⁺20, MG23, RRKK23, CHM⁺25] compute the triple as a single encrypted matrix product under a joint key and exchange 9 to 15 MB per party per authenticated 128×128 triple over a 128-bit prime field. The price is paid elsewhere: one-time evaluation keys of hundreds of megabytes to tens of gigabytes (1.45 GB in MatriGear [CHM⁺25], 27 GB in [CKR⁺20]), and a computational cost which, despite a decade of progress on homomorphic matrix multiplication [HS14, HS18, JKLS18, CYW⁺25, BCH⁺24, Par25, GL26, CKL26], still stands at one to

ten seconds per 128×128 triple on a single core [CHM⁺25, RRKK23], and at seven seconds to a minute at dimension 256, three orders of magnitude above the cost of the online multiplication it enables.

PCGs for matrix triples. Given the pervasiveness of matrix multiplication in secure computation, one could expect PCGs to have transformed the generation of matrix triples as they did for OT and OLE. This has not happened: the few existing constructions actually run into much the same barrier as the protocols above. Concretely, a programmable PCG for OLE [BCG⁺20b] yields matrix triples by composition, but the seed grows by an n^3 factor (where n is the matrix dimension). Boyle et al. [BCG⁺20b, Section 7.4] managed to reduce this overhead to n^2 by leveraging their ring-LPN PCG for inner products, but this remains limited to small matrices¹. Li, Xing, Yao and Yuan [LXY25a] give a direct construction from quasi-abelian syndrome decoding whose seed grows only linearly in n , but at the cost of moving to an extension field of degree n : every operation of the expansion is an arithmetic operation in $\mathbb{F}_{p^n}$, the number of PRG evaluations per triple grows quadratically in n . In all cases the seed size or the expansion cost scales with the matrix dimension, and PCGs stop being competitive with the interactive protocols beyond very small matrices. A “dream” PCG for matrix triples would have a seed size independent of *both* the number of triples and the matrix dimension, and a concretely efficient expansion procedure scaling even at high dimensions. This is the main goal of this work.

1.1 Our Contributions

In this work, we provide compelling solutions to this important open problem, obtaining PCGs for matrix triples with seed size $\text{poly}(\lambda, \log n, \log N)$ bits for producing N matrix triples over $\mathbb{F}^{n \times n}$ with hidden constants on par with those of the best known PCG constructions for OLE [BCG⁺20b, BCCD23]. Concretely, we contribute two new PCG constructions, each interesting in its own right:

- Our first construction (Section 3) relies on the same QA-SD assumption underlying several previous PCG constructions [BCCD23, BBC⁺24a, LLX⁺26]. Its expansion cost is $\Theta(n^3 N \log(n^3 N))$ field operations (Equation (18)), i.e. *cubic* in the matrix dimension n per triple. Compared with the best previous PCG for matrix triples [LXY25a], which relies on the same assumption over an extension field of degree n , this retains the same cubic per-triple expansion cost, but shaves *all* dependency on n from the seed.
- The main downside of the first construction is exactly this cubic per-triple cost. Our second construction overcomes it: its expansion cost is $(2c^2 - 1) \cdot \text{MM}(n)$ field operations per triple (Equation (19)), where $\text{MM}(n)$ denotes the cost of multiplying two $n \times n$ matrices, with any algorithm (e.g. Strassen’s), and c is the module rank of the assumption. For $c = 2$, this is only $7 \cdot \text{MM}(n)$: a factor 7 above computing the matrix triple *in the clear*. This comes at the cost of introducing a new assumption, twisted QA-SD, a natural generalization of QA-SD that captures the noncommutativity of the ring of $n \times n$ matrices.

We complement these constructions with an in-depth analysis of twisted QA-SD (Sections 5 and 6.4). In a nutshell, the twist has essentially no effect on standard, ISD-style attacks, and we generalize the folding framework, the main structural attack against QA-SD, to the twisted setting. Intriguingly, our analysis suggests that twisted QA-SD might even be *more* secure than QA-SD in some parameter regimes: over small fields, ordinary QA-SD is threatened by sparse-interpolation attacks [BDHV25, Jou26], which recover the noise from scalar points where every public multiplier vanishes simultaneously, a phenomenon inherent to evaluating commutative polynomials on a grid. No such points exist for the twisted construction, whose public multiplier is a noncommuting algebra element rather than a scalar-valued function. Whether

¹Boyle et al. report seeds of 83 MB expanding to 400 MB of 8×8 triples, and of 330 MB expanding to 810 MB of 16×16 triples, a compression factor of 5 and 2.5 that extrapolates to tens of gigabytes of seed at $n = 128$, and conclude that the approach “is only likely to be useful for small matrix dimensions”

some generalization of these attacks nonetheless applies to twisted QA-SD is, we believe, an intriguing open question.

Eventually, in Section 6, we analyze the concrete efficiency of our constructions when integrating several standard optimizations from the PCG literature and estimate their runtime on a sample of parameters using operation counts and micro-benchmarks on our machine (an Apple M1 Pro).

1.2 Our Approach

As in previous works, we start from the line of work on PCGs for OLEs from ring-LPN [BCG⁺20b] and quasi-abelian syndrome decoding (QA-SD) [BCCD23, BBC⁺24a, LXYY25a], and ask ourselves how these approaches should be adapted to generalize to matrix triples. The main technical hurdle boils down to the following observation: multivariate polynomial rings and group algebras (the algebraic structures in which these constructions operate) are commutative, and therefore cannot natively capture the non-commutativity of matrix multiplication. In these constructions, the parties obtain pseudorandom elements x_0, x_1 of a group algebra (each of which is a compressed encoding of a long vector), together with additive shares of their product $x_0 \cdot x_1$, which directly yields a large batch of OLEs. To generate matrix triples instead, we need a way to embed the non-commutativity of matrix multiplication in this framework. Concretely, we seek a replacement for the product of the group algebra with the following properties: elements of the algebra can be efficiently mapped to matrices, and when the replacement operation is applied to two elements encoding two matrices, the result is again an element that can be efficiently mapped to a matrix, and this matrix must be the product of the two encoded matrices. In other words, the encoding must turn the new operation into matrix multiplication. We consider two ways of achieving this.

Moving out of the group algebra. In the first approach, matrices are encoded as elements of $\mathbb{F}_q[G]$, but we replace the product with an operation that moves the operands *out* of the algebra: to multiply two encodings, we first move them into an intermediate group algebra, over a larger group with one additional coordinate. This coordinate plays the role of the summation index of a matrix product (the index k that is contracted when computing $\sum_k \mathbf{A}_{ik} \mathbf{B}_{kj}$): the first encoding is placed in the larger algebra with this coordinate in the position of the column index of $\mathbf{A}$, and the second one with this coordinate in the position of the row index of $\mathbf{B}$. We then multiply the two elements in the larger algebra, and come back to the original algebra by summing over the additional coordinate (a trace operation), which performs the contraction. The result encodes the product of the two matrices (up to a fixed scalar factor). The downside of this approach is that the parties have to work in a group algebra which is n times larger than the one in which the encodings live, and this factor n shows up in the cost of the expansion. The benefit is that the pseudorandom elements x_0, x_1 never leave the original group algebra $\mathbb{F}_q[G]$, so we can rely on exactly the same QA-SD assumption as previous works. This approach is directly inspired by a recent line of work on homomorphic encryption for matrix arithmetic [GL26], which multiplies encrypted matrices encoded in a multivariate ring through an auxiliary variable and a trace.

Twisting the product. Our second approach uses a different strategy. Rather than going out of the group algebra, we directly modify its internal product. Recall that a group algebra $\mathbb{F}_q[G]$ has a basis $\{u_g\}_{g \in G}$ indexed by the group elements, and that the product of two basis elements is the basis element indexed by the sum of their indices: $u_g \cdot u_h = u_{g+h}$. A *twisted* group algebra keeps the same basis, but multiplies the product of two basis elements by a scalar that depends on the two indices: $u_g \cdot u_h = \alpha(g, h) u_{g+h}$, where the twist α maps pairs of group elements to $\mathbb{F}_q^\times$ (and satisfies a compatibility condition which keeps the product associative). Twisting changes neither the additive structure of the algebra nor the notion of sparsity of its elements, but it can make the product non-commutative. The key observation is that there is a choice of twist for which the twisted group algebra of $G = (\mathbb{Z}/n\mathbb{Z})^2$ is exactly the algebra of

$n \times n$ matrices over $\mathbb{F}_q$. This observation is old, and was made in a very different context: at the birth of quantum mechanics, Weyl introduced the “clock” matrix $\mathbf{P}$ and the “shift” matrix $\mathbf{Q}$ in his study of quantum kinematics [Wey50]. They satisfy $\mathbf{PQ} = \omega\mathbf{QP}$ for a primitive n -th root of unity ω , and their products $\mathbf{P}^i\mathbf{Q}^j$ form a basis of the $n \times n$ matrices in which the product of two basis elements is another basis element, multiplied by a power of ω : this is precisely a twisted group algebra of $(\mathbb{Z}/n\mathbb{Z})^2$. This basis was later studied by Schwinger [Sch60], and it is known today as the generalized Pauli basis [Got99]. Adding a batch coordinate, we obtain a twisted group algebra together with an efficiently computable one-to-one mapping to batches of N matrices (it consists of Fourier transforms along two of the three coordinates, and costs a quasi-linear number of operations), such that the twisted product of two elements translates exactly to the matrix products of the matrices they encode. In this approach, there is no auxiliary coordinate and no trace: the algebra, and hence the domain of the function secret sharing scheme, has exactly the size of a batch of N matrices, as for PCGs for OLE producing the same number of field elements. The downside is that the pseudorandom elements now live in the twisted algebra, so we need a twisted variant of the QA-SD assumption, which we analyze in depth in Sections 5 and 6.4. We give a more detailed overview of the two constructions in Section 1.4.

1.3 Applications

As we mentioned above, secure matrix products are used in a wide variety of works on applied secure computation, and our new PCGs have the potential to significantly improve many of them, in the same way that PCGs for OT, VOLE, and OLE have already yielded significant speedups in many settings of secure computation [BCG⁺19b, BCG⁺19a, CRR21, BCG⁺22, RRT23, KPRR25, KPR⁺26, BCGI18, BCG⁺20b, AS22, BCCD23, BBC⁺24a, LXYY25a, MMST25, ARR25]. In this paper, we focus on two kinds of applications to illustrate this point.

Threshold MAYO. We show that our PCGs provide the missing ingredient needed to generate the correlated randomness required by a recent line of work on post-quantum threshold signatures for the MAYO signature scheme [CEN25, ABCN26]. The threshold protocol of Aranha, Borin, Celi, and Niot [ABCN26] reduces every interactive step of threshold MAYO signing to a handful of products of secret-shared matrices, but its analysis assumes that the matrix triples used for these products, together with their MACs, are supplied by a trusted party, and leaves their generation open. We show that our PCGs fill exactly this gap. For one of the round-2 parameter sets of MAYO, one signing attempt only needs 29 square matrix triples of size 81, which our twisted construction produces in about 17 core-milliseconds and 45 KiB of amortized seed per party, and this reduces the communication of the online phase of the protocol by a factor of about 13.9.

Privacy-preserving machine learning. We revisit several classical works on privacy-preserving machine learning, and show that our new PCGs would bring them significant improvements. Many of these works already reduce a large part of their computation to products of secret-shared matrices, and either generate the corresponding triples in a preprocessing phase or evaluate the products online using homomorphic encryption; this is the case, for example, of SecureML [MZ17] for training neural networks, of the maliciously secure inference protocol of Chen et al. [CKR⁺20], of BumbleBee [jLHG⁺25] for transformer inference, and of the secure Poisson regression protocol of Kelkar et al. [KLRS22]. As one example, evaluating the convolutions of a ResNet-50 model with the protocol of Chen et al. uses 3298 products of 128 by 128 matrices, and generating the corresponding matrix triples with their maliciously secure construction costs about 41 GB of communication per party and 7.4 hours of computation on 16 threads; the same triples, produced with our twisted PCG instead, would only cost about 1.3 MiB of amortized seed per party and 48 seconds of computation on a single core.

1.4 Technical Overview

Pseudorandom Correlation Generator. Informally, a PCG consists of two algorithms. $\text{Gen}(1^\lambda)$ outputs a pair of short seeds (k_0, k_1) , one per party; and the deterministic $\text{Expand}(\sigma, k_\sigma)$ stretches k_σ into a long string R_σ , for $\sigma \in \{0, 1\}$, with $|k_\sigma| \ll |R_\sigma|$. Correctness requires that (R_0, R_1) be computationally indistinguishable from an instance of target correlation $\mathcal{C}$. Security requires that a party holding k_σ learn nothing about the other party's output beyond what $\mathcal{C}$ already reveals: given k_σ , the string $R_{1-\sigma}$ must be indistinguishable from one reverse-sampled from $\mathcal{C}$ conditioned on R_σ .

Function Secret Sharing (FSS). Informally, FSS allows to share the function $f : \mathcal{I} \rightarrow \mathbb{G}$ for some group $\mathbb{G}$ into (f_0, f_1) such that f_σ computationally hides f and for all $x \in \mathcal{I}$, $f_0(x) + f_1(x) = f(x)$. This could be trivially achieved by sharing the truth table of the function f ; however, one would typically want the shares to be compact (with respect to the size of the truth table). For point functions, efficient constructions of FSS are known [GI14, BGI15, BGI16]. Hence, to succinctly share an t -sparse vector, just use FSS for the sum of t -point functions (SPFSS), and can therefore be shared with keys of size $O(t\lambda \log |\mathcal{I}|)$ for λ be the security parameter. Running FullEval on such a key returns the party's share at every point of $\mathcal{I}$ at once. For the type of FSS we consider, FullEval is significantly faster than the generic solution of running $|\mathcal{I}|$ instances of Eval .

A Template for constructing PCG using FSS. A general template to construct PCG for a target correlation $\mathcal{C}$ was used in many works [BCGI18, BCG⁺19b, BCG⁺20b, BCCD23, BBC⁺24a]. At a high level, each party's share of $\mathcal{C}$ is generated from a short sparse seed, made pseudorandom by an LPN-style assumption over a suitable ring. The correlation between the two shares reduces to a product, and expanding that product shows that its coefficients are public: the only secret part is a handful of cross terms, each of them sparse. Hence Gen samples the sparse seeds and generates SPFSS keys for those cross terms, setting k_σ to be the SPFSS keys together with its own sparse seed. In turn, $\text{Expand}(\sigma, k_\sigma)$ runs FullEval on each key to recover a share of each cross term, combines these with the public coefficients, and adds its own locally computable part. Here the seed stays short because the SPFSS keys are: the domain enters only through logarithm. Hence, the seed size depends on the SPFSS key size and the size of the sparse seeds. The expansion cost then depends on the evaluation using SPFSS keys and on other operations needed to output the additive share.

In ring-LPN-based constructions of PCG, the choice of the ring is a design parameter, and changing the ring can significantly affect the efficiency. To illustrate this, let us examine constructions of PCG from prior works that use different rings and consequently achieve varying levels of efficiency.

PCG for degree-two correlations. The goal of Boyle et al. [BCG⁺19b] was to get a correlation in which each party gets a pseudorandom share $\vec{x}_\sigma$ together with additive share of the tensor product $\vec{x}_0 \otimes \vec{x}_1$. The construction here uses dual form of LPN: for a public randomly sampled matrix H over $\mathbb{Z}_q$ and a t -sparse vector $\vec{e}_\sigma$ of length n over $\mathbb{Z}_q$, the vector $\vec{x}_\sigma = H\vec{e}_\sigma$ is pseudorandom. Now, Gen samples $\vec{e}_0$ and $\vec{e}_1$, observes that their tensor product has at most t^2 non-zero coordinates, and then uses FSS for the sum of t^2 -point functions to distribute it. The Expand algorithm uses bilinearity of tensor product i.e. $(H\vec{e}_0) \otimes (H\vec{e}_1) = (H \otimes H)(\vec{e}_0 \otimes \vec{e}_1)$. This allows each party, to recover its share of tensor product locally. Since $\vec{x}_\sigma$ is pseudorandom for each $\sigma \in \{0, 1\}$ under LPN, we have the desired tensor product correlation. However, efficiency was a bottleneck here. The SPFSS domain has size n^2 when the output has length n and the expansion is, therefore, inherently quadratic in the output length. A natural fix, observed by [BCG⁺20b], is to view each sparse vector as a sparse polynomial $\vec{e}_\sigma$ and to let $\vec{x}_\sigma$ be its evaluations at public points, so that the cross term becomes the polynomial product $\vec{e}_0 \cdot \vec{e}_1$, computable in quasilinear time. This is insecure because the resulting code can be efficiently distinguished by algebraic decoding.

Changing the ring. The resolution of [BCG⁺20b] was to have the same polynomial view but choose the underlying ring intelligently. Working in $\mathcal{R}_p = \mathbb{Z}_p[X]/F(X)$ for a degree- N modulus F , ring-LPN states that $(a, a \cdot e + f)$ is pseudorandom for sparse e, f and random $a \in \mathcal{R}_p$. The product $e_i f_j$ of two

t -sparse polynomials is again t^2 -sparse, but now it lives in a domain of size $2N$ rather than N^2 . Hence, each party's seed is logarithmic in N scaled by t^2 . Now, the expansion costs $\tilde{O}(N)$ ring operations using fast polynomial multiplication instead of the $\Omega(N^2)$ of the tensor route, since now each tensor product is only of length 2. Choosing F to split into N distinct linear factors turns one OLE over $\mathcal{R}_p$ into N OLEs over $\mathbb{Z}_p$.

From the prior works above, the point we are trying to convey is that the ring is a design parameter, and choosing it intelligently significantly improves efficiency. [BCCD23] pushed this further by replacing $\mathbb{Z}_p[X]/F(X)$ with a group algebra $\mathbb{F}_q[G]$ for a finite abelian G , so that the underlying code is quasi-abelian; the resulting QA-SD assumption supports small fields and underlies FOLEAGE [BBC⁺24a] and the multi-quadratic instantiations of [LLX⁺26].

What we want: PCG for matrix triples. Our target is the batched matrix-triple correlation where each party receives $(\mathbf{A}_\sigma, \mathbf{B}_\sigma, \mathbf{Z}_\sigma)$ each in $(\mathcal{M}_n(\mathbb{F}_q))^N$ such that $\mathbf{Z}_0 + \mathbf{Z}_1 = (\mathbf{A}_0 + \mathbf{A}_1) \times (\mathbf{B}_0 + \mathbf{B}_1)$. Here, $(\mathcal{M}_n(\mathbb{F}_q))^N$ is the matrix space of $n \times n$ matrices, batched N times, with each entry in $\mathbb{F}_q$, and the product is being taken component-wise across N matrix slots.

A trivial solution to this is to take the existing PCG for OLE and use them to build matrix triples. This would need n^3 scalar triples per matrix triple, so the seed and expansion time both inherit a factor of n^3 , which is clearly inefficient. Also, as mentioned in the introduction, Boyle et al. [BCG⁺20b, Section 7.4] managed to reduce this overhead to n^2 by leveraging their ring-LPN PCG for inner products, but this remains limited to small matrices. What we would like instead is a ring whose own multiplication already is matrix multiplication, so that a single ring LPN sample carries a matrix rather than a scalar.

Such a ring already exists! Such an encoding from matrix to ring was recently given by [GL26], where they use it to build an efficient Fully Homomorphic Encryption Scheme (FHE) from Ring Learning with Errors (RLWE) problem. Their starting point is the multivariate ring

$$\mathcal{R}_{\text{GL}} = \mathbb{Z}[i][X, Y, W] / \langle X^n - i, Y^n - i, \Phi_p(W) \rangle,$$

over the Gaussian integers. Let ζ_j be primitive $4n$ -th roots of unity satisfying $\zeta_j^n = i$ and η_ℓ be primitive p -th roots of unity. Let φ be the Euler totient function. A polynomial $m \in \mathcal{R}_{\text{GL}}$ encodes $\varphi(p)$ $n \times n$ matrices as follows.

$$\gamma_{\text{GL}}(m) = \left\{ \mathbf{M}_{jk}^{(\ell)} = m(\zeta_j, \zeta_k, \eta_\ell) : j, k \in [0, n), \ell \in [0, \varphi(p)) \right\},$$

Here, two matrix indices are carried by X and Y , and the batch index by W . Multiplication is then obtained by introducing an auxiliary variable Z to carry the contracted index and taking the *trace* over it. Let polynomial $a, b \in \mathcal{R}_{\text{GL}}$ encodes $\mathbf{A}^{(\ell)}, \mathbf{B}^{(\ell)}$ respectively, then multiplication of these two matrices is represented by

$$\text{Tr}_Z \left(a(X, Z, W) \cdot b(Y^{-1}, Z^{-1}, W^{-1}) \right) \quad \text{that encodes} \quad \frac{1}{n} \mathbf{A}^{(\ell)} \mathbf{B}^{(\ell)*}.$$

where Tr_Z is being defined as the degree-zero component in Z and $\mathbf{B}^*$ is the conjugate transpose of $\mathbf{B}$. They additionally show that this trace can be read off in the coefficient domain as a matrix product of coefficient matrices.

For security, they base the hardness on RLWE which is stated over *single-variable* cyclotomic rings. So a ring with several variables is usable only if it can be shown isomorphic to some single-variable one. Almost all of the machinery in their construction traces back to this one requirement. More technically, over the Gaussian integers one has $\mathbb{Z}[i][X]/\langle X^n - i \rangle \cong \mathbb{Z}[X]/\langle X^{2n} + 1 \rangle$, and this is what allows $\mathbb{Z}[X, W]/\langle \Phi_{4n}(X), \Phi_p(W) \rangle \cong \mathbb{Z}[X]/\langle \Phi_{4np}(X) \rangle$. Here $\gcd(4n, p) = 1$ and Φ_m denotes the m -th cyclotomic polynomial. They also observe that the naive alternative $\mathbb{Z}[X, Y]/\langle X^n + 1, Y^n + 1 \rangle$ does not provide an efficient matrix encoding². Taking $X^n - i$ then costs them three things at once.

²For further details regarding this, we refer the reader to [GL26].

- The coefficients must be Gaussian integers, simply because i has to live in the ring.
- The roots of unity used for the encoding satisfy $\zeta^n = i$ and therefore have order $4n$ rather than n .
- Exponents no longer cancel by plain addition, so one of the two factors must be inverted before multiplying. Over $\mathbb{C}$, inverting a root of unity is complex conjugation — which is why their trace identity yields the conjugate transpose $\mathbf{A}\mathbf{B}^*$ rather than the plain product $\mathbf{A}\mathbf{B}$, and why further steps are needed to recover ordinary transposes.

However, this encoding is designed under the RLWE constraints that we simply do not have and once those are removed the ring and operations are simplified.

Simplified Ring. We in turn use the following simplified ring over $\mathbb{F}_q$

$$\mathcal{R} = \mathbb{F}_q[X, Y, W] / \langle X^n - 1, Y^n - 1, W^N - 1 \rangle$$

with the mapping $\gamma : \mathcal{R} \rightarrow (\mathcal{M}_n(\mathbb{F}_q))^N$ such that

$$\gamma(m) = \left\{ \mathbf{M}_{jk}^{(\ell)} = m(\zeta_j, \zeta_k, \eta_\ell) : j, k \in [0, n), \ell \in [0, N) \right\},$$

where $\zeta \in \mathbb{F}_q$ is a fixed primitive n -th root of unity, $\eta \in \mathbb{F}_q$ a fixed primitive N -th root of unity, and $\zeta_j := \zeta^j$, $\eta_\ell := \eta^\ell$ (we require $n, N \mid q - 1$, so that $\mathbb{F}_q$ contains the required roots of unity). This simplification is because of the following reasons. In our setting, we base the hardness on Ring-LPN.

In fact, ring-LPN over the ring $\mathcal{R}$ defined above is not a new object. It is precisely the Quasi-Abelian Syndrome Decoding (QASD) assumption that is well-studied in [BCCD23, BBC⁺24a, LLX⁺26]. The QASD assumption informally states that

$$(a, ae + f) \approx (a, u)$$

where $a, u \xleftarrow{\$} \mathbb{F}_q[G]$ and $e, f \xleftarrow{\$} \mathcal{D}_t(\mathbb{F}_q[G])$ where $\mathcal{D}_t(\mathbb{F}_q[G])$ a noise distribution over $\mathbb{F}_q[G]$ such that $\text{wt}(x) = t$ when $x \leftarrow \mathcal{D}_t$. Here G is a finite abelian group. Hence, when $G = (\mathbb{Z}/n\mathbb{Z})^2 \times \mathbb{Z}/N\mathbb{Z}$, this $\mathcal{R} \cong \mathbb{F}_q[G]$.

We therefore take $X^n - 1$ directly, and all three costs disappear together. Concretely, with $X^n - 1$ the exponents live in $\mathbb{Z}/n\mathbb{Z}$ and cancel exactly when $u + v \equiv 0$, so it is enough to place the contracted variable Z in the first slot of the second factor. Therefore, the trace operation used in our setting is much simpler

$$\text{Tr}_Z(a(X, Z, W) b(Z, Y, W)), \quad \& \text{ it encodes } \frac{1}{n} \mathbf{A}^{(\ell)} \mathbf{B}^{(\ell)}.$$

Also [GL26] uses $\Phi_p(W)$ for the batch index and obtain $\varphi(p)$ slots; we use $W^N - 1$ and obtain all N .

PCG for matrix triples from QASD Assumption. The construction now just follows the template described above. Let x_σ^A, x_σ^B be such that of $\gamma(x_\sigma^A) = \mathbf{A}_\sigma, \gamma(x_\sigma^B) = \mathbf{B}_\sigma$ for each σ . We will treat these x_σ^A, x_σ^B as the ring-LPN sample over $\mathcal{R}$ described above i.e. $x_\sigma^\tau = \langle \vec{a}, \vec{e}_\sigma^\tau \rangle$ for t -sparse vectors $\vec{e}_\sigma^\tau$ where $\tau \in \{A, B\}$ and the first entry of $\vec{a}$ is 1. Note that given $\vec{e}_\sigma^\tau$ to party σ for all $\tau \in \{A, B\}$, that party can compute x_σ^A, x_σ^B since $\vec{a}$ is public and therefore, compute $\mathbf{A}_\sigma, \mathbf{B}_\sigma$ by using the map γ . It then computes $\mathbf{D}_\sigma = \mathbf{A}_\sigma \mathbf{B}_\sigma$. However, to compute cross products $\mathbf{A}_\sigma \mathbf{B}_{1-\sigma}$ for each σ , the construction uses SPFSS. In particular, using SPFSS for sum of t^2 point functions, we can share the cross terms $e_0^{i,A} \cdot e_1^{j,B}$ and $e_1^{i,A} \cdot e_0^{j,B}$ for i, j ranging over the size of the vector. Once, each party obtains the additive share of $e_0^{i,A} \cdot e_1^{j,B}$ and $e_1^{i,A} \cdot e_0^{j,B}$, they multiply it with the corresponding part of the public vector $\vec{a}$.

However, we cannot first multiply everything in $\mathcal{R}$ and apply the trace at the end. What we need is

$$\text{Tr}_Z(x_0^A(X, Z, W) \cdot x_1^B(Z, Y, W)) = \sum_{i,j} \text{Tr}_Z(a_i(X, Z, W) e_0^{i,A}(X, Z, W) \cdot a_j(Z, Y, W) e_1^{j,B}(Z, Y, W)), \quad (1)$$

in which a_i occupies the (X, Z, W) slot and a_j the (Z, Y, W) slot. Let $m \circledast m' := \text{Tr}_Z (m(X, Z, W) m'(Z, Y, W))$. The operation $\circledast$ is not bilinear with respect to ring multiplication: in general

$$(a_i a_j) \circledast (e_0^{i,A} e_1^{j,B}) \neq (a_i e_0^{i,A}) \circledast (a_j e_1^{j,B}),$$

so additive shares of the three-variable products $e_0^{i,A} \cdot e_1^{j,B} \in \mathcal{R}$ are not enough to compute shares of the right-hand side.

Therefore, to use trace operation meaningfully, we switch to 4-variable ring

$$\mathcal{R}' = \mathbb{F}_q[X, Y, Z, W] / \langle X^n - 1, Y^n - 1, Z^n - 1, W^n - 1 \rangle$$

Hence, using SPFSS we share the sparse lifted product $e_0^{i,A}(X, Z, W) \cdot e_1^{j,B}(Z, Y, W)$ and $e_1^{i,A}(X, Z, W) \cdot e_0^{j,B}(Z, Y, W)$, multiply by $a_i(X, Z, W) \cdot a_j(Z, Y, W)$ and then take the trace operation with respect to Z . Since the shared object now keeps the two slots separate, the parties can multiply their shares by $a_i(X, Z, W) a_j(Z, Y, W)$ in $\mathcal{R}'$ and only then take Tr_Z , which computes exactly the right-hand side of Equation (1).

Now, each party can apply $n \cdot \gamma$ map on the output of trace and get the shares $\tilde{\mathbf{Z}}_\sigma^1$ of $\mathbf{A}_0 \mathbf{B}_1$ and get the shares $\tilde{\mathbf{Z}}_\sigma^2$ of $\mathbf{A}_1 \mathbf{B}_0$. Each party then outputs $(\mathbf{A}_\sigma, \mathbf{B}_\sigma, \mathbf{Z}_\sigma = \tilde{\mathbf{Z}}_\sigma^1 + \tilde{\mathbf{Z}}_\sigma^2 + \mathbf{D}_\sigma)$. The correctness and security directly follow from the correctness and security of SPFSS scheme and ring-LPN assumption. For more details, please refer to Section 3.1 and Section 3.2.

Let us now analyze the efficiency. Note that the domain of SPFSS is $< 4n^3N$. Hence, the seed size is logarithmic in $4n^3N$ scaled by t^2 factor plus some additional $t \log(n), t \log(N)$ factors for sharing the sparse vectors. However, the bottleneck is in the expansion cost. The reason is structural rather than an artifact of our analysis. This is because of the trace operation taken over the Z dimension, which forces the SPFSS full-domain evaluations and the subsequent multiplications by $a_i(X, Z, W) a_j(Z, Y, W)$ to be carried out in $\mathcal{R}'$. Expansion therefore costs $O(n^3N \log(n^3N))$ operations in $\mathbb{F}_q$, a factor n more than the n^2N elements it produces.

The n^3 survives because we are simulating a non-commutative operation, matrix multiplication, inside a commutative ring, and the auxiliary variable is what pays for the mismatch. The question now is, can we remove this mismatch? Can we work in an algebra whose multiplication is matrix multiplication, with no auxiliary variable and no trace?

Making matrix multiplication native. The extra factor of n in the preceding construction comes from the auxiliary variable Z . We introduced Z because multiplication in an ordinary group algebra is commutative, whereas matrix multiplication contracts one index in an order-sensitive way. The trace over Z implements this contraction after the fact. To remove the extra dimension, we instead ask for an algebra in which matrix multiplication is already the native multiplication. At the same time, we must retain the sparse basis structure used by the PCG template: if r and s have weight t , their product should have weight at most t^2 , so that it can still be shared succinctly using SPFSS.

Twisted group algebras give us both properties. As a vector space, a twisted group algebra looks exactly like an ordinary group algebra because it has a basis $\{u_g\}_{g \in G}$. The difference is only in the multiplication. For a suitable 2-cocycle $\alpha: G \times G \rightarrow \mathbb{F}_q^\times$, we set $u_g u_h = \alpha(g, h) u_{g+h}$. Thus, the product of two basis elements is still a scalar multiple of a single basis element, and consequently $\text{wt}(rs) \leq \text{wt}(r) \text{wt}(s)$. Unlike an ordinary group algebra, however, the multiplication need not be commutative, since $\alpha(g, h)$ and $\alpha(h, g)$ may differ.

The particular twist we use comes from the clock-and-shift representation of matrices. Let $\omega \in \mathbb{F}_q$ be a primitive n -th root of unity, let $\mathbf{P}$ be the diagonal “clock” matrix with diagonal $1, \omega, \dots, \omega^{n-1}$, and let $\mathbf{Q}$ be the cyclic-shift matrix. They satisfy $\mathbf{PQ} = \omega \mathbf{QP}$, and the n^2 matrices $\{\mathbf{P}^i \mathbf{Q}^j\}_{i,j \in \mathbb{Z}/n\mathbb{Z}}$ form a basis of $\mathcal{M}_n(\mathbb{F}_q)$. Their multiplication obeys $(\mathbf{P}^i \mathbf{Q}^j)(\mathbf{P}^k \mathbf{Q}^\ell) = \omega^{-jk} \mathbf{P}^{i+k} \mathbf{Q}^{j+\ell}$. This is exactly the multiplication rule of a twisted group algebra over $(\mathbb{Z}/n\mathbb{Z})^2$. In particular, $\mathbb{F}_q^\alpha[(\mathbb{Z}/n\mathbb{Z})^2] \cong \mathcal{M}_n(\mathbb{F}_q)$.

We then append an ordinary abelian group H of size N for batching. Since $\mathbb{F}_q[H]$ splits into N field components for our choice of parameters, we obtain an efficiently computable algebra isomorphism

$$\Psi: \mathcal{A} := \mathbb{F}_q^\alpha[(\mathbb{Z}/n\mathbb{Z})^2 \times H] \longrightarrow \mathcal{M}_n(\mathbb{F}_q)^N.$$

This is the important difference from the construction above. An element of $\mathcal{A}$ already represents a batch of N matrices, and $\Psi(xy) = \Psi(x)\Psi(y)$, where the product on the right is component-wise matrix multiplication. There is no auxiliary contracted index, no lifting to a four-variable ring, and no trace operation.

A twisted QA-SD assumption. We next need the analogue of the pseudorandom sparse encodings used in the ordinary QA-SD construction. The natural candidate is to sample a uniform $a \in \mathcal{A}$ and sparse $e, f \in \mathcal{A}$ and assume that $(a, ae + f) \approx (a, u)$ for uniform $u \in \mathcal{A}$. When $n = 1$, the twist disappears and this is precisely ordinary two-block QA-SD. For $n > 1$, however, this is a new assumption over a noncommutative algebra; ordinary QA-SD does not by itself imply it. We therefore formulate it explicitly as *twisted QA-SD*.

Noncommutativity creates one issue that was not present before. If both parties formed their pseudorandom values in the same orientation, multiplying them would interleave the dense multiplier a with the sparse secret factors, and the terms that have to be shared would no longer be products of two sparse elements. We avoid this by using opposite orientations. Define $x_L = ae_L + f_L$ and $x_R = e_Ra + f_R$. Then the dense multipliers move to the outside without commuting any factors:

$$\begin{aligned} x_L x_R &= a(e_L e_R)a + a(e_L f_R) \\ &\quad + (f_L e_R)a + f_L f_R. \end{aligned}$$

The four secret expressions inside this formula are ordered products of two weight- t elements. Each has weight at most t^2 in the twisted basis and can be additively shared using SPFSS exactly as in the ordinary QA-SD construction. The surrounding multiplications by a are then performed locally during expansion. Although this argument uses both left-oriented samples $ae + f$ and right-oriented samples $ea + f$, for our Weyl twist the two assumptions are equivalent through the transpose anti-automorphism of the matrix algebra. Thus it suffices to posit one twisted QA-SD assumption.

From an oriented product to a matrix triple. This gives a simple subroutine for producing two pseudorandom algebra elements together with additive shares of their *ordered* product. To obtain a matrix triple we invoke it twice, since $(A_0 + A_1)(B_0 + B_1) = A_0B_0 + A_0B_1 + A_1B_0 + A_1B_1$.

In the first invocation, P_0 takes the left role and P_1 the right role. The parties obtain A_0 and B_1 , together with shares U_0, U_1 satisfying $U_0 + U_1 = A_0B_1$. In the second invocation the roles are reversed, giving A_1 and B_0 together with shares V_0, V_1 satisfying $V_0 + V_1 = A_1B_0$. Each party can compute its same-party product locally and sets $C_\sigma = A_\sigma B_\sigma + U_\sigma + V_\sigma$. Consequently,

$$C_0 + C_1 = (A_0 + A_1)(B_0 + B_1).$$

Finally, applying Ψ to $A_\sigma, B_\sigma, C_\sigma$ turns this algebra identity directly into N additively shared $n \times n$ matrix triples.

Security follows the same basic pattern as for the QA-SD construction. The SPFSS keys hide the sparse cross products, while twisted QA-SD lets us replace each oriented sparse encoding by an independent uniform algebra element. After these replacements, the operands are uniform and the product shares have exactly the distribution obtained by reverse-sampling an ideal matrix triple. The only additional care is the ordering of multiplication because at no point can factors be commuted, which is why the two oriented invocations are kept separate throughout the construction and proof.

Hardness of twisted QA-SD. At first glance, the cryptanalysis of the ordinary QA-SD assumption does not translate to the twisted setting, due to noncommutativity. More specifically, even if we choose a subgroup G' of G , the quotient map $G \rightarrow G/G'$ is in general not compatible with the twist. In

particular, it does not induce a compatible morphism between the corresponding twisted group algebras. Nevertheless, this description as morphism of algebras appears to be unnecessarily restrictive. Instead, we show that ordinary folding can be reformulated as a projection from the original twisted group algebra onto a lower-dimensional module (which is not necessarily a subalgebra anymore). We then show how this module-theoretic interpretation survives the twisting setting. In particular, the same folding attacks (which are then used as subroutines within decoding algorithms) provide essentially the same framework for twisted group algebras as they do for the usual setting.

On the other hand, recent interpolation-based attacks over small fields as introduced in [BDHV25, Jou26] still seem out of reach. More specifically, their starting point is to evaluate a QA-SD sample at points where all public polynomials vanish simultaneously, which exposes evaluations of one sparse error term. More algebraically, their attack crucially relies on the fact that evaluation is an algebra homomorphism to the underlying field $\mathbb{F}_q$. It then uses the multiplicative-character structure of $\mathbb{F}_q^\times$ to lift these evaluations to the field of complex numbers $\mathbb{C}$ and apply techniques from signal processing (namely compressed sensing). Neither step has a direct analogue in the twisted group algebra setting. Indeed, the corresponding components are matrix algebras rather than fields, and the subsequent character-based complex lifting does not naturally extend to noncommutative algebras. It was already observed that working over larger fields prevents the attack, and we believe that adding a twist also thwarts it.

What does the twist buy us? The algebra has dimension $L = n^2N$, which is exactly the number of field elements in a batch of N matrices. Each of the eight SPFSS instances used by the matrix-triple construction now has domain size L , rather than the $\Theta(n^3N)$ domain required by the four-variable trace construction. Thus the factor n caused by explicitly representing the contracted matrix index disappears from the FSS expansion altogether.

Now, each party performs eight full-domain SPFSS evaluations over a domain of size n^2N , followed by a constant number of products in $\mathcal{A}$ and conversions through Ψ . The latter conversion is a collection of Fourier transforms and is quasilinear in n^2N when the chosen groups admit fast transforms; dense multiplication in $\mathcal{A}$ is simply N ordinary $n \times n$ matrix multiplications under Ψ . In other words, the construction no longer pays an additional algebraic dimension just to express matrix multiplication. The work that remains is the cost of expanding the sparse correlations and carrying out the matrix products themselves.

1.5 Organization

Section 2 recalls the necessary background on FSS, PCGs, group algebras, and twisted group algebras. Section 3 presents our first construction, from QA-SD, and Section 4 our second construction, from twisted QA-SD. Section 5 extends the folding framework to twisted group algebras, Section 6 describes the optimizations and the concrete efficiency estimates, and Section 7 details the applications.

2 Preliminaries

2.1 Notations

We let λ denote a security parameter, and use the standard definitions of negligible functions, computational indistinguishability (with respect to nonuniform distinguishers), and pseudorandom generators. We use $[0..n]$ to denote the index set $\{0, \dots, n-1\}$, as well as $[0..n] = \{0, \dots, n\}$ and $[n] = \{1, \dots, n\}$.

Vectors, outer sum, outer product, and matrices. We use column vectors by default. For a vector $\vec{u}$, let i -th element of it be denoted by $\vec{u}[i]$. For two vectors $\vec{u}, \vec{v} \in \mathcal{R}^t$, for some ring $\mathcal{R}$, we write $\vec{u} \boxplus \vec{v}$ to mean the *outer sum* given by the length t^2 vector $(\vec{u}[i] + \vec{v}[j])_{i \in [t], j \in [t]}$. Similarly, we define the flattened outer product (or tensor product) to be $\vec{u} \boxtimes \vec{v} = (\vec{u}[i] \cdot \vec{v}[j])_{i \in [t], j \in [t]}$, that is, the vector $(\vec{u}[1] \cdot \vec{v}, \dots, \vec{u}[t] \cdot \vec{v})$. We denote the inner product of two vectors by $\langle \vec{u}, \vec{v} \rangle$. For a ring $\mathcal{R}$ and a positive

integer n , we define the matrix space $\mathcal{M}_n(\mathcal{R}) := \{(a_{jk}) \mid a_{jk} \in \mathcal{R}, 1 \leq j, k \leq n\}$, the set of all $n \times n$ matrices over $\mathcal{R}$. The (j, k) -th entry of a matrix $\mathbf{M}$ is denoted by $\mathbf{M}[j, k]$.

2.2 Cryptographic Primitives

2.2.1 Function Secret Sharing and Distributed Point Functions

Function secret sharing (FSS) [BGI15, BGI16] allows a dealer to split up a secret function f into succinct shares $\text{key}_0, \text{key}_1$, such that each share individually hides the function f . The shares can be locally evaluated on an input x to get shares of $f(x)$.

Definition 2.1 (Function Secret Sharing). Let $\mathcal{F} = \{f : I \rightarrow \mathbb{G}\}$ be a class of function descriptions, where the description of each f specifies the input domain I and an Abelian group $(\mathbb{G}, +)$ as the output domain. A (2-party) *function secret sharing* (FSS) scheme for $\mathcal{F}$ is a pair of algorithms $\text{FSS} = (\text{FSS.Gen}, \text{FSS.Eval})$ with the following syntax:

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

The scheme should satisfy the following requirements:

Correctness. For any $f \in \mathcal{F}$ and $x \in I$, keys output by $\text{FSS.Gen}(1^\lambda, f)$ satisfy

$$\text{FSS.Eval}(0, K_0, x) + \text{FSS.Eval}(1, K_1, x) = f(x)$$

with probability one over key generation.

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

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

We say that FSS is secure if, for each $b \in \{0, 1\}$, there exists Sim such that, for every nonuniform polynomial-size adversary $\mathcal{A}$,

$$\left| \Pr[\mathcal{A}(\text{Real}_b(1^\lambda)) = 1] - \Pr[\mathcal{A}(\text{Ideal}_b(1^\lambda)) = 1] \right| \leq \text{negl}(\lambda).$$

In the constructions we use, the leakage function $\text{Leak} : \{0, 1\}^* \rightarrow \{0, 1\}^*$ is given by $\text{Leak}(f_\lambda) = (I, \mathbb{G})$; namely, it outputs a description of the input and output domains of f .

We also define a full-domain evaluation algorithm, $\text{FSS.FullEval}(b, K_b)$, which outputs a vector of $|I|$ group elements, corresponding to running Eval on every element x in the domain I . For the type of FSS we consider, FSS.FullEval is significantly faster than the generic solution of running $|I|$ instances of Eval .

Definition 2.2 (Pseudorandom full-domain evaluations). Let FSS be an FSS scheme for a class $\mathcal{F} = \{f : I \rightarrow \mathbb{G}\}$ with finite domain I and abelian output group $\mathbb{G}$, equipped with the full-domain evaluation algorithm above. We say that FSS has *pseudorandom full-domain evaluations* if, for each $b \in \{0, 1\}$ and every polynomial-size sequence $(f_\lambda)_\lambda$ with $f_\lambda \in \mathcal{F}$, the distributions

$$\left\{ (f_\lambda, \text{FSS.FullEval}(b, K_b)) \mid (K_0, K_1) \xleftarrow{\$} \text{FSS.Gen}(1^\lambda, f_\lambda) \right\}$$

and

$$\left\{ (f_\lambda, U) \mid U \xleftarrow{\$} \mathbb{G}^{|I|} \right\}$$

are computationally indistinguishable against nonuniform polynomial-time distinguishers.

Remark 2.3 (Independent composition). We will use the following standard hybrid consequence of FSS security. Polynomially many FSS instances may have arbitrarily correlated function descriptions, provided that their key pairs are generated with independent coins. Conditional on the descriptions, we may also carry auxiliary information that is independent of these key-generation coins. A hybrid that replaces the instances one at a time then shows that simulation security holds jointly for the corresponding keys. If the FSS scheme has pseudorandom full-domain evaluations, the same argument shows that the full-domain evaluations may be replaced jointly by independent uniform vectors. The nonuniform formulation above lets us condition on, or equivalently hardwire, the correlated descriptions and auxiliary information in each single-instance hybrid.

Definition 2.4 (Distributed Point Function (DPF)). Let N be a domain size, $\mathbb{G}$ an abelian group, $\alpha \in [N]$, and $\beta \in \mathbb{G}$. A *point function* is a function $f_{\alpha, \beta} : [N] \rightarrow \mathbb{G}$ that evaluates to 0 on all inputs $x \neq \alpha$, and $f_{\alpha, \beta}(x) = \beta$ if $x = \alpha$. An FSS scheme for the class of point functions of domain size N is known as a *distributed point function* [GI14].

The DPF construction of [BGI16] can use any pseudorandom generator (PRG) $G : \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda+2}$ and has the following efficiency features. For $m = \lceil \frac{\log |\mathbb{G}|}{\lambda+2} \rceil$, the key generation algorithm Gen invokes G at most $2(\lceil \log N \rceil + m)$ times, the evaluation algorithm Eval invokes G at most $\lceil \log N \rceil + m$ times, and the full-domain evaluation algorithm FullEval invokes G at most $N \cdot (1 + m)$ times. The size of each key is at most $\lceil \log N \rceil \cdot (\lambda + 2) + \lambda + \lceil \log_2 |\mathbb{G}| \rceil$ bits.

As in previous work [BCG⁺20b, BCCD23], it will be convenient for our constructions to use FSS for the class of functions that can be described as a sum of t point functions, which we define next.

Definition 2.5 (FSS for sum of points (SPFSS)). For $\vec{s} \in [N]^t$ and $\vec{y} \in \mathbb{G}^t$, define

$$f_{\vec{s}, \vec{y}}(x) := \sum_{\substack{i \in [t] \\ \vec{s}[i] = x}} \vec{y}[i].$$

Thus repeated locations are allowed, and their payloads are added in $\mathbb{G}$. An SPFSS scheme is an FSS scheme for this class of functions.

A simple realization of SPFSS is obtained by independently generating t DPF key pairs and adding their evaluations.

Definition 2.6 (FSS for sum of points over ordered tuple (oSPFSS)). For $\vec{s} \in [N_1]^t$, $\vec{r} \in [N_2]^t$, $\vec{v} \in [N_3]^t$, $\vec{z} \in [N_4]^t$ and $\vec{y} \in \mathbb{G}^t$, the *sum of point functions over ordered tuples*

$$f_{\vec{s}, \vec{r}, \vec{v}, \vec{z}, \vec{y}} : [N_1] \times [N_2] \times [N_3] \times [N_4] \longrightarrow \mathbb{G}$$

is defined by

$$f_{\vec{s}, \vec{r}, \vec{v}, \vec{z}, \vec{y}}(x_1, x_2, x_3, x_4) := \sum_{\substack{i \in [t] \\ (\vec{s}[i], \vec{r}[i], \vec{v}[i], \vec{z}[i]) = (x_1, x_2, x_3, x_4)}} \vec{y}[i].$$

Thus repeated tuple locations are allowed, and all payloads assigned to the same location are added in $\mathbb{G}$. An oSPFSS scheme is an FSS scheme for the class of sums of point functions over ordered tuples.

To simplify notation, we use the shorthands

$$\text{SPFSS.Gen}(1^\lambda, \vec{s}, \vec{y}) \quad \text{and} \quad \text{oSPFSS.Gen}(1^\lambda, \vec{s}, \vec{r}, \vec{v}, \vec{z}, \vec{y})$$

for key generation on the corresponding function descriptions. A simple realization of oSPFSS is obtained by summing t independent DPF instances over a domain of size $N_1 N_2 N_3 N_4$.

For the function classes above, pseudorandom full-domain shares follow from ordinary FSS security.

Lemma 2.7 (Pseudorandom full-domain shares from FSS security). Let $T = T(\lambda) \geq 1$ be polynomially bounded. Let FSS be a correct and simulation-secure FSS scheme for sums of exactly T point functions over a domain I of polynomial size and an efficiently samplable finite abelian group $\mathbb{G}$. Assume that T is public, zero payloads are allowed, and the leakage is the same for all functions in the class. Then FSS has pseudorandom full-domain evaluations in the sense of Definition 2.2.

Proof. For a function f and a party b , let

$$Y_b^f := \text{FSS.FullEval}(b, K_b), \quad (K_0, K_1) \xleftarrow{\$} \text{FSS.Gen}(1^\lambda, f).$$

Because every function in the class has the same leakage, simulation security and efficient postprocessing give

$$Y_b^f \stackrel{c}{\approx} Y_b^g \tag{2}$$

for any two functions f, g in the class.

Let 0 be the zero function and set $Y := Y_0^0$. Let h be any point function. Both 0 and h belong to the class: use zero payloads in all T slots for 0 , and use one slot for h and zero payloads in the remaining slots. Correctness gives

$$Y_0^h + Y_1^h = h, \quad Y_0^0 + Y_1^0 = 0.$$

Applying Equation (2) first to party 1 and then to party 0 yields

$$Y_0^h = h - Y_1^h \stackrel{c}{\approx} h - Y_1^0 = h + Y_0^0 \quad \text{and} \quad Y_0^h \stackrel{c}{\approx} Y_0^0.$$

Consequently,

$$Y \stackrel{c}{\approx} Y + h \tag{3}$$

for every point function h .

Every fixed vector $v \in \mathbb{G}^{|I|}$ is a sum of at most $|I|$ point functions. Since $|I|$ is polynomially bounded, a hybrid using Equation (3) shows that $Y \stackrel{c}{\approx} Y + v$ for every fixed $v \in \mathbb{G}^{|I|}$.

Suppose that a nonuniform polynomial-time distinguisher separates Y from a uniform $U \xleftarrow{\$} \mathbb{G}^{|I|}$ with non-negligible advantage. For an independent uniform $V \xleftarrow{\$} \mathbb{G}^{|I|}$, the random variable $Y + V$ is exactly uniform, and hence

$$\Pr[\mathcal{A}(U) = 1] = \mathbb{E}_{v \xleftarrow{\$} \mathbb{G}^{|I|}} [\Pr[\mathcal{A}(Y + v) = 1]].$$

There is therefore a fixed polynomial-size vector v for which the same distinguisher separates Y from $Y + v$ with non-negligible advantage. The vector v may be hardwired into a nonuniform distinguisher, contradicting the preceding hybrid. Thus Y_0^0 is pseudorandom. Correctness gives $Y_1^0 = -Y_0^0$, so Y_1^0 is pseudorandom as well. Finally, Equation (2) extends the claim to all functions f in the class. $\square$

In particular, independently composing T simulation-secure DPF instances gives simulation-secure SPFSS and oSPFSS schemes by Remark 2.3; Lemma 2.7 then gives their pseudorandom full-domain evaluations.

2.2.2 Pseudorandom Correlation Generators

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

Definition 2.8 (Reverse-samplable $\binom{n}{t}$ -Correlation Generator). A PPT algorithm $\mathcal{C}$ is called a *reverse samplable $\binom{n}{t}$ -correlation generator*, if on input 1^λ it outputs elements in $\{0,1\}^\ell \times \dots \times \{0,1\}^\ell \in (\{0,1\}^\ell)^n$ for some length parameter $\ell \in \text{poly}$, and there exists a PPT algorithm RSample such that for any $T \subset [n]$ of size $\leq t$, the correlation obtained via:

$$\left\{ \begin{array}{l} (R'_1, \dots, R'_n) \mid \begin{array}{l} (R_1, \dots, R_n) \xleftarrow{\$} \mathcal{C}(1^\lambda) \\ (R'_\sigma)_{\sigma \in T} := (R_\sigma)_{\sigma \in T} \\ (R'_j)_{j \in \bar{T}} \xleftarrow{\$} \text{RSample}(T, (R_\sigma)_{\sigma \in T}) \end{array} \end{array} \right\}$$

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

Definition 2.9 (Pseudorandom $\binom{n}{t}$ -Correlation Generator). Let $\mathcal{C}$ be a reverse-samplable $\binom{n}{t}$ -correlation generator. A *pseudorandom correlation generator (PCG)* for $\mathcal{C}$ consists of algorithms $(\text{PCG.Setup}, \text{PCG.Gen}, \text{PCG.Expand})$ with the following syntax:

- $\text{PCG.Setup}(1^\lambda)$ is a PPT algorithm that outputs public parameters pp .
- $\text{PCG.Gen}(1^\lambda; \text{pp})$ is a PPT algorithm that outputs seeds $(\text{key}_1, \dots, \text{key}_n)$.
- $\text{PCG.Expand}(\text{pp}, \sigma, \text{key}_\sigma)$ is a polynomial-time algorithm that outputs $R_\sigma \in \{0,1\}^\ell$ for $\sigma \in [n]$.

The setup is optional: when no public parameters are required, we set $\text{pp} = \perp$ and omit Setup and pp from the notation.

Correctness. Define $\text{Real}_{\text{corr}}(1^\lambda)$ by sampling

$$\begin{aligned} \text{pp} &\xleftarrow{\$} \text{PCG.Setup}(1^\lambda), \\ (\text{key}_1, \dots, \text{key}_n) &\xleftarrow{\$} \text{PCG.Gen}(1^\lambda; \text{pp}), \\ R_\sigma &\leftarrow \text{PCG.Expand}(\text{pp}, \sigma, \text{key}_\sigma) \quad (\sigma \in [n]), \end{aligned}$$

and outputting $(\text{pp}, R_1, \dots, R_n)$. Define $\text{Ideal}_{\text{corr}}(1^\lambda)$ by sampling

$$\text{pp} \xleftarrow{\$} \text{PCG.Setup}(1^\lambda), \quad (R_1, \dots, R_n) \xleftarrow{\$} \mathcal{C}(1^\lambda),$$

independently, and outputting the same tuple. Correctness requires $\text{Real}_{\text{corr}} \stackrel{c}{\approx} \text{Ideal}_{\text{corr}}$.

$\binom{n}{t}$ -Security. Fix $T \subseteq [n]$ with $|T| \leq t$. The experiment $\text{Real}_T(1^\lambda)$ samples

$$\begin{aligned} \text{pp} &\xleftarrow{\$} \text{PCG.Setup}(1^\lambda), \\ (\text{key}_1, \dots, \text{key}_n) &\xleftarrow{\$} \text{PCG.Gen}(1^\lambda; \text{pp}), \\ R_j &\leftarrow \text{PCG.Expand}(\text{pp}, j, \text{key}_j) \quad (j \in \bar{T}), \end{aligned}$$

and outputs $(\text{pp}, (\text{key}_i)_{i \in T}, (R_j)_{j \in \bar{T}})$. The experiment $\text{Ideal}_T(1^\lambda)$ samples

$$\begin{aligned} \text{pp} &\stackrel{\$}{\leftarrow} \text{PCG.Setup}(1^\lambda), \\ (\text{key}_1, \dots, \text{key}_n) &\stackrel{\$}{\leftarrow} \text{PCG.Gen}(1^\lambda; \text{pp}), \\ R_i &\leftarrow \text{PCG.Expand}(\text{pp}, i, \text{key}_i) \quad (i \in T), \\ (R_j)_{j \in \bar{T}} &\stackrel{\$}{\leftarrow} \text{RSample}(T, (R_i)_{i \in T}), \end{aligned}$$

and outputs the same tuple. Here RSample is the reverse-sampling algorithm for $\mathcal{C}$. Security requires $\text{Real}_T \stackrel{\epsilon}{\approx} \text{Ideal}_T$ for every such T .

Definition 2.10 (Batched matrix-triple correlation). For parameters q, n, N , the two-party correlation $\mathcal{C}_{\text{Mat}, n, N}$ samples

$$\mathbf{A}_0, \mathbf{A}_1, \mathbf{B}_0, \mathbf{B}_1, \mathbf{C}_0 \stackrel{\$}{\leftarrow} \mathcal{M}_n(\mathbb{F}_q)^N$$

independently and uniformly, and sets

$$\mathbf{C}_1 := (\mathbf{A}_0 + \mathbf{A}_1)(\mathbf{B}_0 + \mathbf{B}_1) - \mathbf{C}_0,$$

where multiplication is component-wise across the N matrix components. Party P_σ receives $(\mathbf{A}_\sigma, \mathbf{B}_\sigma, \mathbf{C}_\sigma)$.

The correlation is reverse sampleable. Given one party's output $(\mathbf{A}_\sigma, \mathbf{B}_\sigma, \mathbf{C}_\sigma)$, sample $\mathbf{A}_{1-\sigma}, \mathbf{B}_{1-\sigma} \stackrel{\$}{\leftarrow} \mathcal{M}_n(\mathbb{F}_q)^N$ independently and set

$$\mathbf{C}_{1-\sigma} := (\mathbf{A}_0 + \mathbf{A}_1)(\mathbf{B}_0 + \mathbf{B}_1) - \mathbf{C}_\sigma.$$

2.3 Group Algebras

We use ordinary group algebras to obtain a batching dimension and twisted group algebras to encode noncommutative matrix multiplication. For standard background on group algebras and their representation-theoretic decomposition, see [Ser77, Ch. 6, §§6.1–6.2, pp. 47–49]; for the convolution and Fourier-transform viewpoints, see [Ter99, Ch. 15, p. 245 and Ch. 10, Thm. 2, pp. 168–169].

Let $\mathbb{F}_q$ denote the finite field with q elements, and let G be a finite abelian group of cardinality n . The group algebra of G with coefficients in $\mathbb{F}_q$ is the set $\mathbb{F}_q[G]$ of formal linear combinations

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

with multiplication induced by the group law of G and extended bilinearly. The elements of G form a distinguished basis. This basis induces a canonical support and Hamming weight:

$$\text{supp}_G(x) := \{g \in G : x_g \neq 0\}, \quad \text{wt}_G(x) := |\text{supp}_G(x)|.$$

Lemma 2.11. Let $x, y \in \mathbb{F}_q[G]$. Then

$$\text{wt}_G(xy) \leq \text{wt}_G(x) \text{wt}_G(y).$$

In particular, the product of two elements of weight at most t has weight at most t^2 .

Proof. Write $x = \sum_{g \in S} x_g g$ and $y = \sum_{h \in T} y_h h$, where $S = \text{supp}_G(x)$ and $T = \text{supp}_G(y)$. Every basis element occurring in xy is indexed by a product gh with $(g, h) \in S \times T$. Collisions and cancellations can only decrease the support size. $\square$

Let $G = \mathbb{Z}/n\mathbb{Z}$ be the cyclic group with n elements. It is easy to see that the group algebra $\mathbb{F}_q[G]$ is nothing but the usual polynomial ring $\mathbb{F}_q[X]/(X^n - 1)$. The isomorphism is given by $k \mapsto X^k$ extended by linearity.

Remark 2.12 ([BCCD23, Remark 9]). For any finite abelian group G which is a product of cyclic groups $\mathbb{Z}/d_1\mathbb{Z} \times \dots \times \mathbb{Z}/d_r\mathbb{Z}$, the group algebra $\mathbb{F}_q[G]$ is isomorphic to a quotient of a multivariate polynomial ring, namely:

$$\mathbb{F}_q[G] = \mathbb{F}_q[\mathbb{Z}/d_1\mathbb{Z} \times \dots \times \mathbb{Z}/d_r\mathbb{Z}] \simeq \mathbb{F}_q[X_1, \dots, X_r] / \langle X_1^{d_1} - 1, \dots, X_r^{d_r} - 1 \rangle.$$

More generally, for batching, we use $\mathbb{F}_q[H]$ for a finite abelian group H whose group algebra splits completely over $\mathbb{F}_q$. Write

$$\widehat{H} := \text{Hom}(H, \mathbb{F}_q^\times)$$

for its group of $\mathbb{F}_q$ -valued characters.

Proposition 2.13. Let H be a finite abelian group, and let

$$m := \exp(H) = \text{lcm}\{\text{ord}(h) : h \in H\}.$$

If $m \mid (q - 1)$, then H has exactly $|H|$ distinct $\mathbb{F}_q$ -valued characters, and the Fourier transform gives an $\mathbb{F}_q$ -algebra isomorphism

$$\mathbb{F}_q[H] \cong \prod_{\chi \in \widehat{H}} \mathbb{F}_q \cong \mathbb{F}_q^{|H|}.$$

Explicitly, the isomorphism is

$$\sum_{h \in H} x_h h \longmapsto \left(\sum_{h \in H} x_h \chi(h) \right)_{\chi \in \widehat{H}}.$$

Proof. Since $m \mid (q - 1)$, the cyclic group $\mathbb{F}_q^\times$ contains every m -th root of unity [Ter99, Ch. 3, Thm. 3(2), p. 62]. Moreover, $\text{char}(\mathbb{F}_q) \nmid |H|$. Thus $\mathbb{F}_q$ is a splitting field for H , and $\mathbb{F}_q[H]$ is split semisimple. Since H is abelian, all irreducible $\mathbb{F}_q$ -representations of H are one-dimensional. Equivalently, H has exactly $|H|$ distinct $\mathbb{F}_q$ -valued characters. This finite-field splitting statement is also an immediate specialization of [Kar85, Ch. 3, §2, Thm. 2.9, pp. 84–85].

For completeness, character orthogonality over $\mathbb{F}_q$ gives

$$\sum_{h \in H} \chi(h) \psi(h)^{-1} = \begin{cases} |H|, & \chi = \psi, \\ 0, & \chi \neq \psi. \end{cases}$$

The element $|H|$ is invertible in $\mathbb{F}_q$, so the Fourier transform is invertible, with

$$x_h = \frac{1}{|H|} \sum_{\chi \in \widehat{H}} \widehat{x}(\chi) \chi(h)^{-1}.$$

Finally, every character is multiplicative, and therefore the Fourier transform maps multiplication in $\mathbb{F}_q[H]$ to component-wise multiplication. This proves the claimed algebra isomorphism. The corresponding complex Fourier-transform and orthogonality statements appear in [Ter99, Ch. 10, Thm. 2 and the orthogonality lemma, pp. 168–169]. $\square$

Remark 2.14. The condition $\exp(H) \mid (q - 1)$ is weaker than $|H| \mid (q - 1)$. For example, if $H = (\mathbb{Z}/2\mathbb{Z})^d$, then $\exp(H) = 2$, so $\mathbb{F}_q[H] \cong \mathbb{F}_q^{2^d}$ over every finite field $\mathbb{F}_q$ of odd characteristic.

The ordinary group algebra of an abelian group is commutative. They therefore provide a natural batching mechanism, but they cannot by themselves model non-commutative matrix multiplication. This motivates passing to twisted group algebras.

We prove the following lemma, which we will need later.

Lemma 2.15. Let $r \geq 1$ and $d_1, \dots, d_r \geq 1$ be integers, let $G = \mathbb{Z}/d_1\mathbb{Z} \times \dots \times \mathbb{Z}/d_r\mathbb{Z}$, and let

$$R_{\vec{d}} = \mathbb{F}_q[X_1, \dots, X_r] / \langle X_1^{d_1} - 1, \dots, X_r^{d_r} - 1 \rangle.$$

Under the canonical $\mathbb{F}_q$ -algebra isomorphism $\phi : \mathbb{F}_q[G] \longrightarrow R_{\vec{d}}$, $u_{(i_1, \dots, i_r)} \longmapsto X_1^{i_1} \cdots X_r^{i_r}$, the Hamming weight in the distinguished group basis is exactly the number of nonzero coefficients in the unique reduced polynomial representation $f = \sum_{0 \leq i_k < d_k} c_{i_1, \dots, i_r} X_1^{i_1} \cdots X_r^{i_r}$. In particular, defining $\text{wt}_{R_{\vec{d}}}(f) := |\{(i_1, \dots, i_r) : c_{i_1, \dots, i_r} \neq 0\}|$, we have $\text{wt}_{R_{\vec{d}}}(\phi(x)) = \text{wt}_G(x)$ for every $x \in \mathbb{F}_q[G]$. Consequently, for all $f, g \in R_{\vec{d}}$,

$$\text{wt}_{R_{\vec{d}}}(fg) \leq \text{wt}_{R_{\vec{d}}}(f) \text{wt}_{R_{\vec{d}}}(g).$$

Proof. The map ϕ sends the distinguished basis $\{u_{(i_1, \dots, i_r)}\}_{(i_1, \dots, i_r) \in G}$ bijectively to the reduced monomial basis $\{X_1^{i_1} \cdots X_r^{i_r} : 0 \leq i_k < d_k\}$ of $R_{\vec{d}}$, hence it preserves Hamming weight. For $f, g \in R_{\vec{d}}$, let $x = \phi^{-1}(f)$ and $y = \phi^{-1}(g)$. Since ϕ is an algebra isomorphism, $\text{wt}_{R_{\vec{d}}}(fg) = \text{wt}_G(xy) \leq \text{wt}_G(x) \text{wt}_G(y) = \text{wt}_{R_{\vec{d}}}(f) \text{wt}_{R_{\vec{d}}}(g)$, where the inequality follows from Lemma 2.11. $\square$

2.3.1 Twisted group algebras

We recall the twisted group algebra definitions used below. They follow [Con64, pp. 152–153] and [Kar85, Ch. 2, §1, pp. 31–32 and Ch. 3, §2, pp. 77–80].

Definition 2.16. Let G be a finite group. A *normalized 2-cocycle* is a map

$$\alpha : G \times G \rightarrow \mathbb{F}_q^\times$$

such that, for all $g, h, k \in G$,

$$\alpha(g, 1) = \alpha(1, g) = 1$$

and

$$\alpha(g, h) \alpha(gh, k) = \alpha(h, k) \alpha(g, hk).$$

Definition 2.17 (Twisted group algebra). Let $\alpha : G \times G \rightarrow \mathbb{F}_q^\times$ be a normalized 2-cocycle. The twisted group algebra $\mathbb{F}_q^\alpha[G]$ is the $\mathbb{F}_q$ -vector space with distinguished basis

$$\{u_g : g \in G\}$$

and multiplication

$$u_g u_h := \alpha(g, h) u_{gh},$$

extended bilinearly. The cocycle identity is precisely the condition that makes this multiplication associative [Kar85, Ch. 3, §2, Lem. 2.1, p. 77].

The distinguished basis again defines a canonical support and weight. Since the product of two basis elements is a nonzero scalar multiple of a single basis element, the proof of Lemma 2.11 applies without change:

$$\text{wt}_G(xy) \leq \text{wt}_G(x) \text{wt}_G(y) \quad \text{for all } x, y \in \mathbb{F}_q^\alpha[G].$$

2.3.2 The finite Weyl–Pauli twisted algebra

Our starting point is the finite-dimensional *clock-and-shift* construction: a pair of operators $\mathbf{P}, \mathbf{Q}$ on an n -dimensional space satisfying $\mathbf{PQ} = \varepsilon \mathbf{QP}$ for a primitive n -th root of unity ε , whose powers $\mathbf{P}^k \mathbf{Q}^\ell$ span the entire matrix algebra. Because $\mathbf{P}$ and $\mathbf{Q}$ commute only up to the scalar ε , the assignment $(k, \ell) \mapsto \mathbf{P}^k \mathbf{Q}^\ell$ is a projective representation of $\mathbb{Z}_n \times \mathbb{Z}_n$: its multiplication is twisted by a root-of-unity cocycle. It is precisely this twisted-multiplication skeleton, and none of its analytic trappings, that we transport to a finite field.

This construction goes back to Weyl’s projective, or “ray,” representations [Wey50, Ch. III, §16, Eqs. (16.1)–(16.3), p. 181]. Schwinger later gave explicit finite-dimensional clock-and-shift operators and proved that their products form a complete operator basis [Sch60, pp. 573–576]; see also the generalized Pauli basis of Patera and Zassenhaus [PZ88, §V, Eqs. (5.1)–(5.15), pp. 670–671].

We use only the algebraic part of this construction. We replace the complex phase $e^{2\pi i/n}$ by a primitive n -th root of unity $\omega \in \mathbb{F}_q$; no Hilbert-space structure, adjoint operation, or notion of unitarity is used. The condition $n \mid (q - 1)$ guarantees that ω exists and that the Fourier argument below remains valid over $\mathbb{F}_q$.

In more detail, assume that $n \mid (q - 1)$, and fix a primitive n -th root of unity $\omega \in \mathbb{F}_q$. Let $(\vec{e}_r)_{r \in \mathbb{Z}/n\mathbb{Z}}$ be the standard basis of $\mathbb{F}_q^n$. Define the clock and shift matrices $\mathbf{P}, \mathbf{Q} \in \mathcal{M}_n(\mathbb{F}_q)$ by

$$\mathbf{P}\vec{e}_r := \omega^r \vec{e}_r, \quad \mathbf{Q}\vec{e}_r := \vec{e}_{r+1}.$$

Equivalently,

$$\mathbf{P} = \begin{pmatrix} 1 & & & \\ & \omega & & \\ & & \ddots & \\ & & & \omega^{n-1} \end{pmatrix}, \quad \mathbf{Q} = \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}.$$

Then

$$\mathbf{P}^n = \mathbf{Q}^n = \mathbf{I}_n, \quad \mathbf{PQ} = \omega \mathbf{QP}.$$

Lemma 2.18 (Weyl multiplication rule). For all $i, j, k, \ell \in \mathbb{Z}/n\mathbb{Z}$,

$$(\mathbf{P}^i \mathbf{Q}^j)(\mathbf{P}^k \mathbf{Q}^\ell) = \omega^{-jk} \mathbf{P}^{i+k} \mathbf{Q}^{j+\ell}.$$

Proof. The relation $\mathbf{PQ} = \omega \mathbf{QP}$ implies

$$\mathbf{Q}^j \mathbf{P}^k = \omega^{-jk} \mathbf{P}^k \mathbf{Q}^j.$$

This is because for any $r \in \mathbb{Z}/n\mathbb{Z}$,

$$\mathbf{Q}^j \mathbf{P}^k \vec{e}_r = \mathbf{Q}^j (\omega^{rk} \vec{e}_r) = \omega^{rk} \vec{e}_{r+j}, \quad \mathbf{P}^k \mathbf{Q}^j \vec{e}_r = \mathbf{P}^k \vec{e}_{r+j} = \omega^{(r+j)k} \vec{e}_{r+j}.$$

The second exceeds the first by the factor ω^{jk} , so $\mathbf{Q}^j \mathbf{P}^k = \omega^{-jk} \mathbf{P}^k \mathbf{Q}^j$. Therefore

$$(\mathbf{P}^i \mathbf{Q}^j)(\mathbf{P}^k \mathbf{Q}^\ell) = \mathbf{P}^i (\mathbf{Q}^j \mathbf{P}^k) \mathbf{Q}^\ell = \omega^{-jk} \mathbf{P}^{i+k} \mathbf{Q}^{j+\ell}.$$

□

Let $G_0 := (\mathbb{Z}/n\mathbb{Z})^2$ and define $\alpha_0 : G_0 \times G_0 \mapsto \mathbb{F}_q^\times$ as $\alpha_0((i, j), (k, \ell)) := \omega^{-jk}$. For all $x = (i, j)$, $y = (k, \ell)$, and $z = (r, s)$, we have

$$\begin{aligned} \alpha_0(x, y) \alpha_0(x + y, z) &= \omega^{-jk - (j+\ell)r} = \alpha_0(y, z) \alpha_0(x, y + z) \\ \alpha_0(x, (0, 0)) &= \alpha_0((0, 0), x) = \omega^0 = 1 \end{aligned}$$

thereby implying that α_0 is a normalized 2-cocycle. We next give the explicit finite-field realization that we need.

Proposition 2.19 (Weyl realization of the full matrix algebra). Let $\mathcal{A}_n := \mathbb{F}_q^{\alpha_0}[G_0]$ for $G_0 := (\mathbb{Z}/n\mathbb{Z})^2$. Then the linear map

$$\Phi_n : \mathcal{A}_n \rightarrow \mathcal{M}_n(\mathbb{F}_q), \quad u_{i,j} \mapsto \mathbf{P}^i \mathbf{Q}^j,$$

is an $\mathbb{F}_q$ -algebra isomorphism.

Proof. The Lemma 2.18 shows that Φ_n respects multiplication. It remains to prove that the n^2 matrices $\{\mathbf{P}^i \mathbf{Q}^j : i, j \in \mathbb{Z}/n\mathbb{Z}\}$ are linearly independent.

For fixed j , the matrices $\mathbf{P}^i \mathbf{Q}^j$ are supported on the same cyclic diagonal. Since

$$\mathbf{P}^i \mathbf{Q}^j \vec{e}_r = \omega^{i(r+j)} \vec{e}_{r+j},$$

their entries along that diagonal are given, up to a permutation of coordinates, by the rows of the Fourier matrix $(\omega^{ir})_{i,r \in \mathbb{Z}/n\mathbb{Z}}$. This matrix is invertible because $1, \omega, \dots, \omega^{n-1}$ are distinct, so its determinant is a nonzero Vandermonde determinant. Distinct values of j correspond to disjoint cyclic diagonals. Hence all n^2 Weyl matrices are linearly independent. Since $\dim_{\mathbb{F}_q}(\mathcal{M}_n(\mathbb{F}_q)) = n^2$, they form a basis, and Φ_n is an isomorphism. $\square$

Thus every matrix has a unique coefficient representation in the Weyl basis:

$$\mathbf{X} = \sum_{i,j \in \mathbb{Z}/n\mathbb{Z}} x_{i,j} \mathbf{P}^i \mathbf{Q}^j.$$

The sparsity in the twisted-group basis is therefore precisely sparsity of the Weyl coefficient array $(x_{i,j})$.

2.3.3 Batching matrix algebras

Let H be a finite abelian group. Define

$$G := G_0 \times H = (\mathbb{Z}/n\mathbb{Z})^2 \times H$$

and extend the cocycle trivially along H :

$$\alpha((i, j, h), (k, \ell, h')) := \omega^{-jk}.$$

Let

$$\mathcal{A}_{n,H} := \mathbb{F}_q^\alpha[G].$$

The map $u_{i,j,h} \mapsto u_{i,j} \otimes h$ gives

$$\mathcal{A}_{n,H} \cong \mathcal{A}_n \otimes_{\mathbb{F}_q} \mathbb{F}_q[H] \cong \mathcal{M}_n(\mathbb{F}_q) \otimes_{\mathbb{F}_q} \mathbb{F}_q[H].$$

Combining this identity with Proposition 2.13 yields a product of matrix algebras.

Corollary 2.20 (Batched matrix algebra). If $\exp(H) \mid (q-1)$, then

$$\mathcal{A}_{n,H} \cong \mathcal{M}_n(\mathbb{F}_q)^{|H|}.$$

Consequently, taking $N := |H|$, one algebra element represents a batch of N matrices, and algebra multiplication corresponds to component-wise matrix multiplication across the batch.

More explicitly, if

$$x = \sum_{i,j,h} x_{i,j,h} u_{i,j,h},$$

then the component indexed by $\chi \in \hat{H}$ is

$$\mathbf{x}_\chi = \sum_{i,j \in \mathbb{Z}/n\mathbb{Z}} \left(\sum_{h \in H} x_{i,j,h} \chi(h) \right) \mathbf{P}^i \mathbf{Q}^j.$$

This formula also gives an explicit conversion algorithm. First, for each pair (i, j) , perform a Fourier transform over H on the coefficients $(x_{i,j,h})_{h \in H}$. Then, for every $\chi \in \hat{H}$ and every j , recover the entries on the corresponding cyclic diagonal by a length- n Fourier transform over i .

To state the conversion cost precisely, let $T_H(N)$ be the cost of the Fourier transform on H and let $T_n(n)$ be the cost of the length- n cyclic transform. The total cost is

$$O(n^2 T_H(N) + Nn T_n(n)).$$

Hence, whenever

$$T_H(N) = O(N \log N) \quad \text{and} \quad T_n(n) = O(n \log n),$$

the conversion costs

$$O(Nn^2(\log N + \log n))$$

field operations, which is quasilinear in the output size Nn^2 up to logarithmic factors. For the radix-two cyclic FFT bound, see [Ter99, Ch. 9, Thm. 1, p. 153]; the standard composite-length factorization is discussed in [Ter99, Ch. 9, p. 155]. In particular, choosing $H = (\mathbb{Z}/2\mathbb{Z})^d$ makes the H -transform a Walsh–Hadamard transform with cost $O(N \log N)$. The displayed quasilinear bound is therefore conditional on the availability of fast transforms for the selected H and n ; it is not asserted uniformly for every finite abelian group.

2.3.4 From QA-SD to a twisted analogue

The Quasi-Abelian Syndrome Decoding (QA-SD) assumption was introduced by Bombar, Couteau, Couvreur, and Ducros as a decoding assumption over quasi-abelian codes [BCCD23].

Let G be a finite abelian group and $\mathbb{F}_q$ a finite field with q elements. Given an integer $t \in \mathbb{N}$, we denote by $\mathcal{D}_t(\mathbb{F}_q[G])$ a noise distribution over $\mathbb{F}_q[G]$ such that $\mathbb{E}[\text{wt}(x)] = t$ when $x \leftarrow \mathcal{D}_t$, and $\mathcal{D}_{t,n}(\mathbb{F}_q[G]) := \mathcal{D}_t(\mathbb{F}_q[G])^{\otimes n}$ will denote its n -fold tensorization, i.e., $\vec{e} \leftarrow \mathcal{D}_{t,n}(\mathbb{F}_q[G])$ is $\vec{e} \in \mathbb{F}_q[G]^n$ and its coordinates are drawn independently according to $\mathcal{D}_t(\mathbb{F}_q[G])$. A random quasi- G code of index 2, in systematic form, will be a quasi- G code whose parity-check matrix $\mathbf{H} \in (\mathbb{F}_q[G])^2$ is of the form $\mathbf{H} = (1 \mid a)$, where a is uniformly distributed over $\mathbb{F}_q[G]$. Equivalently, it is the dual of the code generated by H . The Quasi-Abelian Syndrome Decoding assumption is defined as follows.

Assumption 2.21 (QA-SD Assumption [BCCD23]). Given a target weight t , the goal of the QA-SD problem is to distinguish, with a non-negligible advantage, between the distributions

$$(a, ae + f) \text{ and } (a, u)$$

where $a, u \xleftarrow{\$} \mathbb{F}_q[G]$ and $e, f \xleftarrow{\$} \mathcal{D}_t(\mathbb{F}_q[G])$. The above generalizes immediately to the case of parity-check matrices with more columns and/or rows of blocks. When $\mathbf{H} = (1 \mid a_1 \mid \dots \mid a_{c-1})$, for some parameter c , this corresponds to what has been called module-LPN in the literature. This corresponds to the hardness of syndrome decoding for a quasi-abelian code of larger rate $(c-1)/c$. We call QA-SD($c, \mathcal{R}$) this natural generalization of QA-SD. The QA-SD assumption states that the above problem should be hard (for appropriate parameters).

The twisted algebra suggests the following noncommutative analogue.

Definition 2.22 (Sparse distribution in the twisted basis). Let $\mathcal{A}_{n,H} = \mathbb{F}_q^\alpha[G]$ have distinguished basis $\{u_g : g \in G\}$. The exact-weight distribution $\mathcal{D}_t(\mathcal{A}_{n,H})$ samples a uniformly random subset $S \subseteq G$ of size t , samples independent coefficients $c_g \xleftarrow{\$} \mathbb{F}_q^\times$ for $g \in S$, and outputs

$$e := \sum_{g \in S} c_g u_g.$$

Assumption 2.23 (Candidate left twisted QA-SD). For appropriate parameters, the distributions

$$(a, ae + f) \quad \text{and} \quad (a, \tilde{u})$$

are computationally indistinguishable, where

$$a, \tilde{u} \xleftarrow{\$} \mathcal{A}_{n,H}, \quad e, f \xleftarrow{\$} \mathcal{D}_t(\mathcal{A}_{n,H}).$$

Assumption 2.24 (Candidate right twisted QA-SD). For appropriate parameters, the distributions

$$(a, ea + f) \quad \text{and} \quad (a, \tilde{u})$$

are computationally indistinguishable, with the same sampling conventions.

When $n = 1$, the group G_0 and the cocycle α_0 are trivial, so $\mathcal{A}_{1,H} \cong \mathbb{F}_q[H]$. Both assumptions then specialize exactly to the two-block ordinary QA-SD assumption. For $n > 1$, however, they are new assumptions over a noncommutative algebra. The specialization at $n = 1$ does not give a reduction from ordinary QA-SD to either twisted variant. In particular, the isomorphism

$$\mathcal{A}_{n,H} \cong \mathcal{M}_n(\mathbb{F}_q)^{|H|}$$

lets an adversary inspect every matrix component, while sparsity is defined before this isomorphism in the twisted-group basis. The resulting component matrices are correlated through the Fourier transform over H . These additional projections and correlations must therefore be analyzed directly; security evidence for ordinary QA-SD does not automatically transfer to the twisted setting.

Although the algebra is noncommutative, the left and right variants above are equivalent in the present construction.

Proposition 2.25 (Equivalence of the left and right variants). Assumption 2.23 holds if and only if Assumption 2.24 holds, for the same parameters.

Proof. Under the identification

$$\mathcal{A}_{n,H} \cong \mathcal{M}_n(\mathbb{F}_q) \otimes_{\mathbb{F}_q} \mathbb{F}_q[H],$$

define the $\mathbb{F}_q$ -linear map

$$\tau(X \otimes r) := X^T \otimes r.$$

Since $\mathbb{F}_q[H]$ is commutative, τ is an anti-automorphism:

$$\tau(xy) = \tau(y)\tau(x).$$

On the distinguished basis, it acts as

$$\tau(u_{i,j,h}) = \omega^{ij} u_{i,-j,h},$$

because

$$(\mathbf{P}^i \mathbf{Q}^j)^\top = \mathbf{Q}^{-j} \mathbf{P}^i = \omega^{ij} \mathbf{P}^i \mathbf{Q}^{-j}.$$

Thus τ permutes the distinguished basis up to multiplication by nonzero scalars. It consequently preserves both the uniform distribution on $\mathcal{A}_{n,H}$ and the exact-weight distribution $\mathcal{D}_t(\mathcal{A}_{n,H})$.

Applying τ component-wise maps

$$(a, ae + f) \quad \text{to} \quad (\tau(a), \tau(e)\tau(a) + \tau(f)),$$

which has the right-sample distribution. It also maps $(a, \tilde{u})$ to a uniform pair. Since τ is efficiently computable and bijective, any distinguisher for one variant yields a distinguisher with the same advantage for the other. $\square$

2.4 Encoding and Decoding of Matrices

In this section, we recall the encoding and decoding procedures of matrices and the concept of the trace operation from [GL26]. They define map γ from $\mathbb{F}_q[X, Y, W] / \langle X^n - 1, Y^n - 1, W^N - 1 \rangle$ to $(\mathcal{M}_n(\mathbb{F}_q))^N$:

$$\gamma(m) = \{\mathbf{M}[j, k]^{(\ell)} = m(\zeta_j, \zeta_k, \eta_\ell) : j, k \in [0, n), \ell \in [0, N)\},$$

$\zeta \in \mathbb{F}_q$ is a fixed primitive n -th root of unity, $\eta \in \mathbb{F}_q$ a fixed primitive N -th root of unity, and $\zeta_j := \zeta^j$, $\eta_\ell := \eta^\ell$ (we require $n, N \mid q - 1$, so that $\mathbb{F}_q$ contains the required roots of unity).

Lemma 2.26. Let $n \mid q - 1$ and $N \mid q - 1$. Then the γ mapping described above is bijective.

Proof. The condition $n \mid q - 1$ and $N \mid q - 1$ implies that $\text{char}(\mathbb{F}_q) \nmid n$ and $\text{char}(\mathbb{F}_q) \nmid N$. Hence, the polynomials $X^n - 1, Y^n - 1, W^N - 1$ split completely into distinct linear factors over $\mathbb{F}_q$. It is immediate from the definition that γ is $\mathbb{F}_q$ -linear. To prove injectivity, it suffices to prove that

$$m(X, Y, W) = \sum_{a=0}^{n-1} \sum_{b=0}^{n-1} \sum_{c=0}^{N-1} m_{abc} X^a Y^b W^c$$

belong to $\ker(\gamma)$ i.e., $m(\zeta_j, \zeta_k, \eta_\ell) = 0$, for every triple (j, k, ℓ) . Ordering the coefficients m_{abc} lexicographically into a column vector $\mathbf{m}$, the above system can be written as

$$(\mathbf{V}_n \otimes \mathbf{V}_n \otimes \mathbf{V}_N) \vec{m} = 0,$$

where $\mathbf{V}_n = (\zeta_j^a)_{0 \leq j, a < n}$, $\mathbf{V}_N = (\eta_\ell^c)_{0 \leq \ell, c < N}$ are Vandermonde matrices.

Since the roots of unity are distinct,

$$\det(\mathbf{V}_n) = \prod_{0 \leq i < j < n} (\zeta_j - \zeta_i) \neq 0,$$

and similarly, $\det(\mathbf{V}_N) \neq 0$. Hence both $\mathbf{V}_n$ and $\mathbf{V}_N$ are invertible. Now we have

$$\det(\mathbf{V}_n \otimes \mathbf{V}_n \otimes \mathbf{V}_N) = \det(\mathbf{V}_n)^{2nN} \det(\mathbf{V}_N)^{n^2} \neq 0.$$

Therefore, $\mathbf{V}_n \otimes \mathbf{V}_n \otimes \mathbf{V}_N$ is invertible, which implies $\vec{m} = 0$. Hence all coefficients m_{abc} vanish, implying $\ker(\gamma) = \{0\}$, further implying γ is injective.

Next, to prove surjectivity, it suffices to notice that

$$\dim_{\mathbb{F}_q} \left(\frac{\mathbb{F}_q[X, Y, W]}{\langle X^n - 1, Y^n - 1, W^N - 1 \rangle} \right) = n^2 N = \dim_{\mathbb{F}_q} ((\mathcal{M}_n(\mathbb{F}_q))^N).$$

Therefore, γ is surjective. $\square$

Definition 2.27 (Trace operation). The trace over Z , denoted $\text{Tr}_Z(a)$, is the coefficient of Z^0 in the unique representative of a of degree less than n in Z . So, reduction modulo $Z^n - 1$ is implicit before taking the constant coefficient; the result may still depend on the other variables.

We will need the following two theorems.

Theorem 2.28 (Polynomial view of matrix multiplication [GL26]). By introducing an auxiliary variable Z , let $c(X, Y, W) = \text{Tr}_Z(a(X, Z, W) \cdot b(Z, Y, W))$, where $a(\zeta_j, \zeta_k, \eta_\ell) = \mathbf{A}[j, k]^{(\ell)}$ and $b(\zeta_j, \zeta_k, \eta_\ell) = \mathbf{B}[j, k]^{(\ell)}$ for all $j, k \in [0, n), \ell \in [0, N)$, then, $c(\zeta_j, \zeta_k, \eta_\ell) = \frac{\mathbf{C}[j, k]^{(\ell)}}{n}$ for $\mathbf{C}^{(\ell)} = \mathbf{A}^{(\ell)} \mathbf{B}^{(\ell)}$.

Theorem 2.29 (Trace as matrix form [GL26]). For polynomials $a, b \in \mathcal{R}$, write $\tilde{a}(X, Y, W) = \sum_{j, k, \ell} \tilde{a}_{j, k, \ell} X^j Y^k W^\ell$ and $\tilde{b}(X, Y, W) = \sum_{j, k, \ell} \tilde{b}_{j, k, \ell} X^j Y^k W^\ell$. Define matrices $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ in $\mathcal{M}_n(\mathcal{R}^N)$ by $\hat{\mathbf{A}}[j, k] = \sum_{\ell} \tilde{a}_{j, k, \ell} W^\ell$ and $\hat{\mathbf{B}}[j, k] = \sum_{\ell} \tilde{b}_{j, k, \ell} W^\ell$. Then

$$c(X, Y, W) = \text{Tr}_Z\left(\tilde{a}(X, Z, W)\tilde{b}(Z^{-1}, Y, W)\right) = \sum_{j, k} \hat{\mathbf{C}}[j, k] X^j Y^k,$$

where $\hat{\mathbf{C}} = \hat{\mathbf{A}}\hat{\mathbf{B}}$ and $\mathcal{R} \cong \mathcal{R}^N[X, Y] / \langle X^n - 1, Y^n - 1 \rangle$. Here Z^{-1} is interpreted modulo $Z^n - 1$, so $Z^{-1} = Z^{n-1}$.

Theorems 2.28 and 2.29 use two different representations. Theorem 2.29 works in the coefficient representation, where the inverse in the second factor is necessary: modulo $Z^n - 1$, the constant coefficient of $Z^r Z^s$ pairs $r + s \equiv 0 \pmod{n}$, whereas that of $Z^r Z^{-s}$ pairs $r = s$. By contrast, Section 3 uses the evaluation encoding γ throughout, for which Theorem 2.28 gives $\text{Tr}_Z(a(X, Z, W)b(Z, Y, W))$ with no inverse. The same evaluation encoding is used in Section 6; on the Z -evaluation grid, Tr_Z is the normalized sum over the n grid points, so the factor n appearing in the QA-SD construction turns this into the ordinary sum.

We therefore use

$$a \otimes b := \text{Tr}_Z(a(X, Z, W)b(Z, Y, W))$$

for the trace operation in the evaluation encoding used by the QA-SD construction.

3 PCGs for Matrix Multiplication from QA-SD

In this section, we present the construction of PCG for matrix multiplication from the QA-SD assumption. In particular, we will construct it from the following assumption, which is nothing but module-LPN as in Assumption 2.21, restated with the exact-weight noise distribution.

Assumption 3.1 (Multivariate module-LPN). Let $n, N \geq 1, c \geq 2$ be all integers, let q be a prime power, and set

$$\mathcal{R} = \mathbb{F}_q[X, Y, W] / \langle X^n - 1, Y^n - 1, W^N - 1 \rangle \cong \mathbb{F}_q[G], \quad G = \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z}.$$

Write $\mathcal{I} = [0..n) \times [0..n) \times [0..N)$ for the index set of the monomial basis $\{X^i Y^j W^k\}_{(i, j, k) \in \mathcal{I}}$ of $\mathcal{R}$, and $\text{wt}(e)$ for the number of nonzero coefficients of e in that basis.

For $t \in \mathbb{N}$, the *exact-weight distribution* $\mathcal{D}_t(\mathcal{R})$ samples a uniformly random subset $S \subseteq \mathcal{I}$ of size t , samples independent coefficients $v_s \leftarrow \mathbb{F}_q^\times$ for $s \in S$, and outputs

$$e(X, Y, W) := \sum_{(i, j, k) \in S} v_{(i, j, k)} X^i Y^j W^k,$$

so that $\text{wt}(e) = t$ with probability 1. We write $\mathcal{D}_t^{\otimes m}(\mathcal{R})$ for the m -fold tensorization, i.e. m independent coordinates.

For $\mathcal{R} = \mathcal{R}(\lambda)$ and $t = t(\lambda)$, we say the module-LPN problem $\mathcal{R}^c\text{-LPN}_{\mathcal{R},t}$ is hard if for every nonuniform polynomial-time distinguisher $\mathcal{A}$,

$$|\Pr[\mathcal{A}(\vec{a}, \langle \vec{a}, \vec{e} \rangle + f) = 1] - \Pr[\mathcal{A}(\vec{a}, u) = 1]| \leq \text{negl}(\lambda),$$

where the probabilities are taken over $\vec{a} \leftarrow \mathcal{R}^{c-1}$, $u \leftarrow \mathcal{R}$, $\vec{e} \leftarrow \mathcal{D}_t^{\otimes(c-1)}(\mathcal{R})$ and $f \leftarrow \mathcal{D}_t(\mathcal{R})$.

3.1 Construction

The construction of PCG $\text{PCG} = (\text{Gen}, \text{Expand})$ for the matrix triples is presented in Figure 1.

Figure 1: PCG for matrix triples from Assumption 3.1

PARAMETERS: Security parameter λ , noise weight $t = t(\lambda)$, field size $q = q(\lambda)$ and ring $\mathcal{R} = \mathbb{F}_q[X, Y, W] / \langle X^n - 1, Y^n - 1, W^N - 1 \rangle$ and $\mathcal{R}' = \mathbb{F}_q[X, Y, Z, W] / \langle X^n - 1, Y^n - 1, Z^n - 1, W^N - 1 \rangle$

PUBLIC INPUT: random polynomials $a_1, \dots, a_{c-1} \in \mathcal{R}$. These are sampled in $\text{Setup}(1^\lambda)$ and outputted as pp^a .

CORRELATION: After expansion, party σ holds $(\mathbf{A}_\sigma, \mathbf{B}_\sigma, \mathbf{Z}_\sigma)$ each in $(\mathcal{M}_n(\mathbb{F}_q))^N$ such that $\mathbf{Z}_0 + \mathbf{Z}_1 = (\mathbf{A}_0 + \mathbf{A}_1) \times (\mathbf{B}_0 + \mathbf{B}_1)$.

$\text{Gen}(1^\lambda)$:

1. For $\sigma \in \{0, 1\}$, $i \in [0..c)$, $\tau \in \{A, B\}$, sample a fresh uniformly random subset $S_\sigma^{i,\tau} \subseteq [0..n) \times [0..n) \times [0..N)$ of size exactly t , list its elements in an arbitrary but fixed order as $(\alpha_k, \beta_k, \tilde{\gamma}_k)_{k \in [0..t)}$, and set

$$\vec{a}_\sigma^{i,\tau}[k] := \alpha_k, \quad \vec{b}_\sigma^{i,\tau}[k] := \beta_k, \quad \vec{c}_\sigma^{i,\tau}[k] := \tilde{\gamma}_k \quad \text{for } k \in [0..t).$$

Sample $\vec{b}_\sigma^{i,\tau} \leftarrow (\mathbb{F}_q^\times)^t$.

2. For $i, j \in [0..c)$,
 - (a) Sample FSS keys $(K_0^{i,j,1}, K_1^{i,j,1}) \leftarrow \text{oSPFSS.Gen}(1^\lambda, \vec{a}_0^{i,A} \boxplus \vec{0}, \vec{0} \boxplus \vec{b}_1^{j,B}, \vec{b}_0^{i,A} \boxplus \vec{a}_1^{j,B}, \vec{c}_1^{i,A} \boxplus \vec{c}_0^{j,B}, \vec{b}_1^{i,A} \otimes \vec{b}_0^{j,B})$. These keys will be used to get the shares of $\mathbf{A}_0 \mathbf{B}_1$.
 - (b) Sample FSS keys $(K_0^{i,j,2}, K_1^{i,j,2}) \leftarrow \text{oSPFSS.Gen}(1^\lambda, \vec{a}_1^{i,A} \boxplus \vec{0}, \vec{0} \boxplus \vec{b}_0^{j,B}, \vec{b}_1^{i,A} \boxplus \vec{a}_0^{j,B}, \vec{c}_0^{i,A} \boxplus \vec{c}_1^{j,B}, \vec{b}_0^{i,A} \otimes \vec{b}_1^{j,B})$. These keys will be used to get the shares of $\mathbf{A}_1 \mathbf{B}_0$.
3. Set $k_\sigma = (\{K_\sigma^{i,j,1}, K_\sigma^{i,j,2}\}_{i,j \in [0..c)}, \{\vec{a}_\sigma^{i,\tau}, \vec{b}_\sigma^{i,\tau}, \vec{c}_\sigma^{i,\tau}, \vec{b}_\sigma^{i,\tau}\}_{i \in [0..c), \tau \in \{A,B\}})$. Output (k_0, k_1) .

$\text{Expand}(\sigma, k_\sigma)$:

1. Parse $k_\sigma = (\{K_\sigma^{i,j,1}, K_\sigma^{i,j,2}\}_{i,j \in [0..c)}, \{\vec{a}_\sigma^{i,\tau}, \vec{b}_\sigma^{i,\tau}, \vec{c}_\sigma^{i,\tau}, \vec{b}_\sigma^{i,\tau}\}_{i \in [0..c), \tau \in \{A,B\}})$.
2. For $i \in [0..c)$, $\tau \in \{A, B\}$, define the polynomial

$$e_\sigma^{i,\tau}(X, Y, W) = \sum_{k \in [0..t)} \vec{b}_\sigma^{i,\tau}[k] X^{\vec{a}_\sigma^{i,\tau}[k]} Y^{\vec{b}_\sigma^{i,\tau}[k]} W^{\vec{c}_\sigma^{i,\tau}[k]}$$

3. For $\tau \in \{A, B\}$, compute $x_\sigma^\tau = \langle \vec{a}, \vec{e}_\sigma^\tau \rangle$ where $\vec{a} = ((a_0 = 1), a_1, \dots, a_{c-1})$ and $\vec{e}_\sigma^\tau = (e_\sigma^{0,\tau}, \dots, e_\sigma^{c-1,\tau})$. Set $\mathbf{A}_\sigma = \gamma(x_\sigma^A)$ and $\mathbf{B}_\sigma = \gamma(x_\sigma^B)$. Compute $\mathbf{D}_\sigma = \mathbf{A}_\sigma \times \mathbf{B}_\sigma$.
4. For $i, j \in [0..c)$, compute
 - (a) $u_{\sigma,i,j}^1 \leftarrow \text{oSPFSS.FullEval}(\sigma, K_\sigma^{i,j,1})$.
 - (b) $u_{\sigma,i,j}^2 \leftarrow \text{oSPFSS.FullEval}(\sigma, K_\sigma^{i,j,2})$.
5. Compute $z_\sigma^1 = \sum_{i,j \in [0..c)} \text{Tr}_Z[(a_i(X, Z, W) a_j(Z, Y, W) u_{\sigma,i,j}^1)]$ and $z_\sigma^2 = \sum_{i,j \in [0..c)} \text{Tr}_Z[(a_i(X, Z, W) a_j(Z, Y, W) u_{\sigma,i,j}^2)]$.
6. Compute $\tilde{\mathbf{Z}}_\sigma^1 = n \cdot \gamma(z_\sigma^1)$, $\tilde{\mathbf{Z}}_\sigma^2 = n \cdot \gamma(z_\sigma^2)$ and $\mathbf{Z}_\sigma = \mathbf{D}_\sigma + \tilde{\mathbf{Z}}_\sigma^1 + \tilde{\mathbf{Z}}_\sigma^2$.
7. Output $(\mathbf{A}_\sigma, \mathbf{B}_\sigma, \mathbf{Z}_\sigma)$.

^aWe implicitly assume that Gen and Expand take pp as input.

Parameter Constraints.

- $n \mid q - 1$ and $N \mid q - 1$ (For bijection property of γ and for the trace identity)
- n, N, t and c are polynomially bounded in λ and $\log q = \text{poly}(\lambda)$.

Correctness. To prove correctness, we prove the following theorem.

Theorem 3.2. Let the assumption $\mathcal{R}^c\text{-LPN}_{\mathcal{R},t}$ hold (Assumption 3.1) and oSPFSS be a correct and simulation secure FSS scheme for the sum of exactly t^2 points over an ordered tuple (Definition 2.6). Then the construction as in Figure 1 is a correct PCG for Matrix multiplication over $(\mathcal{M}_n(\mathbb{F}_q))^N$.

Proof. To argue correctness, we first prove that $\mathbf{Z}_0 + \mathbf{Z}_1 = (\mathbf{A}_0 + \mathbf{A}_1) \times (\mathbf{B}_0 + \mathbf{B}_1)$ and then we prove that the correlation obtained from PCG as in Figure 1 is computationally indistinguishable from matrix multiplication correlation.

Let $i, j \in [0..c)$. For polynomials $e_0^{i,A}, e_1^{j,B}, e_1^{i,A}, e_0^{j,B}$, we have,

$$e_0^{i,A}(X, Z, W) \cdot e_1^{j,B}(Z, Y, W) = \sum_{k, \ell \in [0..t)} \vec{b}_0^{i,A}[k] \cdot \vec{b}_1^{j,B}[\ell] \cdot X^{\vec{a}_0^{i,A}[k]} Y^{\vec{b}_1^{j,B}[\ell]} Z^{\vec{b}_0^{i,A}[k] + \vec{a}_1^{j,B}[\ell]} W^{\vec{c}_0^{i,A}[k] + \vec{c}_1^{j,B}[\ell]}$$

$$e_1^{i,A}(X, Z, W) \cdot e_0^{j,B}(Z, Y, W) = \sum_{k, \ell \in [0..t)} \vec{b}_1^{i,A}[k] \cdot \vec{b}_0^{j,B}[\ell] \cdot X^{\vec{a}_1^{i,A}[k]} Y^{\vec{b}_0^{j,B}[\ell]} Z^{\vec{b}_1^{i,A}[k] + \vec{a}_0^{j,B}[\ell]} W^{\vec{c}_1^{i,A}[k] + \vec{c}_0^{j,B}[\ell]}$$

Hence, we can obtain the coefficients of this product by evaluating the sum of the point functions over the ordered tuples used in the (i, j) -th instance of oSPFSS. This implies

$$\begin{aligned} u_{0,i,j}^1 + u_{1,i,j}^1 &= e_0^{i,A}(X, Z, W) \cdot e_1^{j,B}(Z, Y, W) \\ u_{0,i,j}^2 + u_{1,i,j}^2 &= e_1^{i,A}(X, Z, W) \cdot e_0^{j,B}(Z, Y, W) \end{aligned} \tag{4}$$

Now,

$$\begin{aligned}
z_0^1 + z_1^1 &= \sum_{i,j \in [0..c)} \text{Tr}_Z[(a_i(X, Z, W)a_j(Z, Y, W)u_{0,i,j}^1)] + \sum_{i,j \in [0..c)} \text{Tr}_Z[(a_i(X, Z, W)a_j(Z, Y, W)u_{1,i,j}^1)] \\
&= \sum_{i,j \in [0..c)} \text{Tr}_Z[(a_i(X, Z, W)a_j(Z, Y, W) \cdot (u_{0,i,j}^1 + u_{1,i,j}^1))] \\
&\quad \text{(Since, Tr operation is linear and ring polynomials are distributive)} \\
&= \sum_{i,j \in [0..c)} \text{Tr}_Z[(a_i(X, Z, W)a_j(Z, Y, W) \cdot e_0^{i,A}(X, Z, W)e_1^{j,B}(Z, Y, W))] \quad \text{(From equation 4)} \\
&= \sum_{i,j \in [0..c)} \text{Tr}_Z[(a_i(X, Z, W)e_0^{i,A}(X, Z, W) \cdot a_j(Z, Y, W)e_1^{j,B}(Z, Y, W))] \\
&= \text{Tr}_Z\left[\left(\sum_{i \in [0..c)} a_i(X, Z, W)e_0^{i,A}(X, Z, W) \cdot \sum_{j \in [0..c)} a_j(Z, Y, W)e_1^{j,B}(Z, Y, W)\right)\right] \\
&= \text{Tr}_Z[x_0^A(X, Z, W) \cdot x_1^B(Z, Y, W)] \\
&= x_0^A \otimes x_1^B
\end{aligned}$$

Similarly, we can prove $z_0^2 + z_1^2 = x_1^A \otimes x_0^B$.

Now,

$$\begin{aligned}
\mathbf{Z}_0 + \mathbf{Z}_1 &= \mathbf{D}_0 + \tilde{\mathbf{Z}}_0^1 + \tilde{\mathbf{Z}}_0^2 + \mathbf{D}_1 + \tilde{\mathbf{Z}}_1^1 + \tilde{\mathbf{Z}}_1^2 \\
&= \mathbf{A}_0\mathbf{B}_0 + \mathbf{A}_1\mathbf{B}_1 + (\tilde{\mathbf{Z}}_0^1 + \tilde{\mathbf{Z}}_1^1) + (\tilde{\mathbf{Z}}_0^2 + \tilde{\mathbf{Z}}_1^2) \\
&= \mathbf{A}_0\mathbf{B}_0 + \mathbf{A}_1\mathbf{B}_1 + n\gamma(z_0^1 + z_1^1) + n\gamma(z_0^2 + z_1^2) \\
&= \mathbf{A}_0\mathbf{B}_0 + \mathbf{A}_1\mathbf{B}_1 + n\gamma(x_0^A \otimes x_1^B) + n\gamma(x_1^A \otimes x_0^B) \\
&= \mathbf{A}_0\mathbf{B}_0 + \mathbf{A}_1\mathbf{B}_1 + \mathbf{A}_0\mathbf{B}_1 + \mathbf{A}_1\mathbf{B}_0 \\
&= (\mathbf{A}_0 + \mathbf{A}_1) \times (\mathbf{B}_0 + \mathbf{B}_1)
\end{aligned} \tag{5}$$

where the second last inequality follows from Theorem 2.28. Now, we use hybrid arguments to prove indistinguishability from batched matrix triple correlation, defined as:

$$\left\{ \left(\begin{array}{l} (\mathbf{A}_0, \mathbf{B}_0, \mathbf{Z}_0), \\ (\mathbf{A}_1, \mathbf{B}_1, \mathbf{Z}_1) \end{array} \right) \middle| \begin{array}{l} \mathbf{A}_0, \mathbf{B}_0, \mathbf{A}_1, \mathbf{B}_1, \mathbf{Z}_0 \leftarrow (\mathcal{M}_n(\mathbb{F}_q))^N, \\ \mathbf{Z}_1 = (\mathbf{A}_0 + \mathbf{A}_1) \times (\mathbf{B}_0 + \mathbf{B}_1) - \mathbf{Z}_0 \end{array} \right\}.$$

Note that the output in the above distribution also contains pp but has been removed here to keep the distributions clean. Indistinguishability arguments also hold with its presence.

Hyb₀: This is the same as in the real world, i.e., for $\sigma \in \{0, 1\}$, $\mathbf{A}_\sigma, \mathbf{B}_\sigma, \mathbf{Z}_\sigma$ is being computed as in Figure 1.

Hyb₁: This is the same as Hyb₀ except that in this hybrid we compute $\mathbf{Z}_1 = (\mathbf{A}_0 + \mathbf{A}_1) \times (\mathbf{B}_0 + \mathbf{B}_1) - \mathbf{Z}_0$. We note that $\text{Hyb}_0 \equiv \text{Hyb}_1$ from equation (5).

Hyb₂: This is the same as Hyb₁ except that in this hybrid instead of computing $u_{0,i,j}^1, u_{0,i,j}^2$ for each i, j from oSPFSS.FullEval, we sample $u_{0,i,j}^1, u_{0,i,j}^2$ uniformly at random from $\mathbb{F}_q^{|\mathcal{I}'|}$ where $\mathcal{I}' = [0..n]^2 \times [0..2n-1] \times [0..2N-1]$. Each oSPFSS instance shares a sum of t^2 point functions of polynomial size, with public t^2 and leakage $(\mathcal{I}', \mathbb{F}_q)$; by Lemma 2.7 it has pseudorandom full-domain evaluations, and by Remark 2.3 the $2c^2$ instances may be replaced jointly. Hence $\text{Hyb}_1 \approx \text{Hyb}_2$.

Hyb₃: This is the same as Hyb₂ except that in this hybrid instead of computing $\mathbf{Z}_0$ as in Figure 1, we sample $\mathbf{Z}_0 \leftarrow (\mathcal{M}_n(\mathbb{F}_q))^N$. To prove indistinguishability from the previous hybrid, we have the following arguments. First note that,

$$\begin{aligned}\mathbf{Z}_0 &= \mathbf{D}_0 + n \cdot \gamma(z_0^1) + n \cdot \gamma(z_0^2) \\ &= \mathbf{D}_0 + n \cdot \gamma\left(\sum_{i,j \in [0..c)} \text{Tr}_Z[a_i(X, Z, W)a_j(Z, Y, W)u_{0,i,j}^1]\right) \\ &\quad + \sum_{i,j \in [0..c)} \text{Tr}_Z[a_i(X, Z, W)a_j(Z, Y, W)u_{0,i,j}^2]\end{aligned}$$

Let $i = j = 0$. In this case, we have $a_i(X, Z, W)a_j(Z, Y, W)u_{0,0,0}^1 = u_{0,0,0}^1$. Therefore, to prove $\mathbf{Z}_0$ is pseudorandom, it suffices to prove that $n \cdot \gamma(\text{Tr}_Z[u_{0,0,0}^1])$ is pseudorandom since anything added to random is still random. Note that $u_{0,0,0}^1$ is not used anywhere else. Since $u_{0,0,0}^1$ is randomly sampled, γ is bijective and $\text{Tr}_Z : R' \rightarrow R$ is surjective $\mathbb{F}_q$ -linear map (by definition), we have $n \cdot \gamma(\text{Tr}_Z[u_{0,0,0}^1])$ indistinguishable from random over $(\mathcal{M}_n(\mathbb{F}_q))^N$.

Hyb₄: This is the same as Hyb₃ except that in this hybrid for $\sigma \in \{0, 1\}, \tau \in \{A, B\}$, instead of computing $x_\sigma^\tau = \langle \vec{a}, \vec{e}_\sigma^\tau \rangle$, we sample $x_\sigma^\tau \leftarrow \mathcal{R}$. Hence, from a standard hybrid argument using $\mathcal{R}^c$ -LPN $_{\mathcal{R},t}$ assumption with fixed $\vec{a}$, we can prove that $(x_0^A, x_0^B, x_1^A, x_1^B)$ is jointly pseudorandom. Since, γ is a bijective, deterministic map, this implies $(\mathbf{A}_0, \mathbf{B}_0, \mathbf{A}_1, \mathbf{B}_1)$ is jointly pseudorandom.

This concludes the proof. $\square$

3.2 Security Proof

In this section, we prove the security of PCG as in Figure 1.

Theorem 3.3. Let the assumption $\mathcal{R}^c$ -LPN $_{\mathcal{R},t}$ hold (Assumption 3.1) and oSPFSS be a correct and simulation secure FSS scheme for the sum of exactly t^2 points over an ordered tuple (Definition 2.6). Then the construction as in Figure 1 is a secure PCG for Matrix multiplication over $(\mathcal{M}_n(\mathbb{F}_q))^N$.

Proof. To prove this, we fix $\sigma = 0$, i.e., corrupting party 1, as the case for $\sigma = 1$ follows symmetrically. We need to prove the indistinguishability of the following two distributions.

$$\begin{aligned}&\left((k_1, \mathbf{A}_0, \mathbf{B}_0, \mathbf{Z}_0) \left| \begin{array}{l} (k_0, k_1) \leftarrow \text{Gen}(1^\lambda) \\ (\mathbf{A}_\sigma, \mathbf{B}_\sigma, \mathbf{Z}_\sigma) \leftarrow \text{Expand}(\sigma, k_\sigma) \\ \forall \sigma \in \{0, 1\} \end{array} \right. \right) \\ &\approx \left((k_1, \mathbf{A}_0, \mathbf{B}_0, \mathbf{Z}_0) \left| \begin{array}{l} (k_0, k_1) \leftarrow \text{Gen}(1^\lambda) \\ \mathbf{A}_0, \mathbf{B}_0 \leftarrow (\mathcal{M}_n(\mathbb{F}_q))^N \\ \mathbf{Z}_0 = (\mathbf{A}_0 + \mathbf{A}_1)(\mathbf{B}_0 + \mathbf{B}_1) - \mathbf{Z}_1 \end{array} \right. \right)\end{aligned}\tag{6}$$

where in the second distribution $(\mathbf{A}_1, \mathbf{B}_1, \mathbf{Z}_1) \leftarrow \text{Expand}(1, k_1)$. Note that the output in the above distribution also contains pp but has been removed here to keep the distributions clean. Indistinguishability arguments also hold with its presence. We prove the indistinguishability via a sequence of hybrids Hyb₀, ..., Hyb₄.

Hyb₀: This is the same as the first distribution in equation 6.

Hyb₁: This is the same as Hyb₀ except that in this hybrid we compute $\mathbf{Z}_0 = (\mathbf{A}_0 + \mathbf{A}_1) \times (\mathbf{B}_0 + \mathbf{B}_1) - \mathbf{Z}_1$. We note that Hyb₀ $\equiv$ Hyb₁ from the correctness of PCG, which in turn is implied by the correctness of FSS scheme.

Hyb₂: This is the same as Hyb₁ except that in this hybrid instead of sampling $K_1^{i,j,1}, K_1^{i,j,2}$ for each $i, j \in [0..c)$ from oSPFSS.Gen, we simulate it using Sim for all $i, j \in [0..c)$. The indistinguishability from previous hybrid follows from oSPFSS security.

Hyb₃: This is the same as Hyb₂ except that in this hybrid instead of computing $x_0^\tau = \langle \vec{a}, \vec{e}_0^\tau \rangle$ and setting $\mathbf{A}_0 = \gamma(x_0^A), \mathbf{B}_0 = \gamma(x_0^B)$, we sample $x_0^\tau \leftarrow \mathcal{R}$ for $\tau \in \{A, B\}$ and set $\mathbf{A}_0 = \gamma(x_0^A), \mathbf{B}_0 = \gamma(x_0^B)$. Hence, from a standard hybrid argument using $\mathcal{R}^c$ -LPN $_{\mathcal{R},t}$ assumption, we can prove that (x_0^A, x_0^B) is jointly pseudorandom. Since γ is a bijective, deterministic map, this implies $(\mathbf{A}_0, \mathbf{B}_0)$ is jointly pseudorandom.

Hyb₄: This hybrid is same as Hyb₃ except that in here we switch back to sampling valid oSPFSS keys, $K_1^{i,j,1}, K_1^{i,j,2}$ using oSPFSS.Gen for all $i, j \in [0..c)$. The indistinguishability from the previous hybrid follows from the security of oSPFSS. This is the second distribution in equation 6.

This concludes the proof. $\square$

Remark 3.4 (General index groups). Section 3 is written with cyclic factors only for ease of understanding. For finite abelian A, H with $\exp(A) \mid q-1$ and $\exp(H) \mid q-1$, writing A and H as products of cyclic groups gives the same monomial presentation of $\mathbb{F}_q[A \times A \times H]$ in several variables. The Assumption 3.1, Figure 1 and Theorems 3.2, 3.3 use only that the group algebra splits completely (Proposition 2.13), that products of sparse elements are sparse with supports given by the group law (Lemma 2.15), and the character orthogonality $\sum_{z \in A} \chi(z) = |A| \cdot [\chi = 1]$ underlying Theorem 2.28. They therefore hold over $\mathbb{F}_q[A \times A \times H]$, with $X^i Y^j W^k$ replaced by the basis element $u_{(g_1, g_2, h)}$ and γ by the character transform. Section 6.2 provide concrete efficiency by instantiating A, H .

3.3 Efficiency analysis of the QA-SD construction

In this section, we provide the asymptotic analysis of the Expand algorithm and seed size.

If we instantiate oSPFSS using PRG $G : \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda+2}$ via PRG-based DPF construction from [BGI16] then we have the following features. For $m = \lceil \frac{\log |\mathbb{G}|}{\lambda+2} \rceil$, the key generation algorithm Gen invokes G at most $2(\lceil \log N \rceil + m)$ times, the evaluation algorithm Eval invokes G at most $\lceil \log N \rceil + m$ times, and the full-domain evaluation algorithm FullEval invokes G at most $N \cdot (1 + m)$ times. The size of each key is at most $\lceil \log N \rceil \cdot (\lambda + 2) + \lambda + \lceil \log_2 |\mathbb{G}| \rceil$ bits.

At first note that since each $e_\sigma^{i,A}, e_\sigma^{i,B}$ is t -sparse, the product $e_0^{i,A} \cdot e_1^{j,B}$ and $e_1^{i,A} \cdot e_0^{j,B}$ is at most t^2 -sparse from Lemma 2.15.

Asymptotic Seed Size. Now for each $\sigma \in \{0, 1\}$, for the construction as in Figure 1, we have,

$$\begin{aligned} |k_\sigma| &= |\{K_\sigma^{i,j,1}, K_\sigma^{i,j,2}\}_{i,j \in [0..c)}| + |\{\vec{a}_\sigma^{i,\tau}, \vec{b}_\sigma^{i,\tau}, \vec{c}_\sigma^{i,\tau}, \vec{b}_\sigma^{i,\tau}\}_{i \in [0..c), \tau \in \{A, B\}}| \\ &= 2c^2 |K_\sigma^{i,j,1}| + 2c |\{\vec{a}_\sigma^{i,\tau}, \vec{b}_\sigma^{i,\tau}, \vec{c}_\sigma^{i,\tau}, \vec{b}_\sigma^{i,\tau}\}| \\ &\leq 2c^2 t^2 (\lceil \log(4n^3 N) \rceil \cdot (\lambda + 2) + \lambda + \lceil \log q \rceil) + 2ct (\log n + \log n + \log N + \log q) \\ &= 2(ct)^2 (\lceil \log(4n^3 N) \rceil \cdot (\lambda + 2) + \lambda + \lceil \log q \rceil) + 2ct(2 \log n + \log N + \log q) \end{aligned}$$

where second equality follows from fact that $|K_\sigma^{i,j,1}| = |K_\sigma^{i,j,2}|$, third inequality follows from the fact that domain size of oSPFSS is $|\mathcal{I}'|$ where $\mathcal{I}' = [0..n)^2 \times [0..2n-1) \times [0..2N-1)$ and its range is $\mathbb{F}_q$. In third inequality, the t^2 factor in the first part is coming from the fact that oSPFSS can be realized with t^2 independent instances of DPF. Similarly, the t factor in second part is coming from the fact that each of $\vec{a}_\sigma^{i,\tau}, \vec{b}_\sigma^{i,\tau}, \vec{c}_\sigma^{i,\tau}$ and $\vec{b}_\sigma^{i,\tau}$ is of length t .

Asymptotic Expansion Cost.

- To compute x_σ^A, x_σ^B , we need $O(cn^2N(\log(n^2N)))$ operations in $\mathbb{F}_q$.
- The γ operation needs $O(n^2N(\log(n^2N)))$ operations in $\mathbb{F}_q$.
- To compute $\mathbf{D}_\sigma$, we would need $O(n^{\hat{c}}N)$ operations in $\mathbb{F}_q$. In here $\hat{c} = 2.8$ if we use Strassen's Algorithm and $\hat{c} = 3$ if we use the trivial multiplication algorithm.
- For oSPFSS.FullEval , we need at most $2(ct)^2(4n^3N)(1 + \lceil \frac{\log q}{\lambda+2} \rceil)$ PRG operations
- To compute z_σ^1, z_σ^2 , we would need $O(c^2n^3N \log(n^3N))$ operations in $\mathbb{F}_q$.
- To compute $\mathbf{Z}_\sigma$, we would need $O(n^2N)$ operations in $\mathbb{F}_q$ because it is just matrix addition.

Hence overall, the computation of Expand can be done with at most $2(ct)^2(4n^3N)(1 + \lceil \frac{\log q}{\lambda+2} \rceil)$ PRG operations and $O(c^2n^3N \log(n^3N))$ operations in $\mathbb{F}_q$.

If we instantiate oSPFSS using PRG $G : \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda+2}$ via PRG-based DPF construction from [BGI16], with the early termination of [BBC⁺24a]: writing $\mu := \lfloor \lambda / \lceil \log_2 q \rceil \rfloor$ for the number of field elements that fit in one λ -bit leaf block, the DPF tree is stopped $\log_2 \mu$ levels early and the last correction word is replaced by a $\mu \lceil \log_2 q \rceil$ -bit one. The only change with respect to the previous asymptotic costs is:

- $|k_\sigma| \leq 2(ct)^2 \left(\lambda + \lceil \log_2 \left(\frac{4n^3N}{\mu} \right) \rceil \cdot (\lambda + 2) + \mu \lceil \log_2 q \rceil \right) + 2ct(2 \log n + \log N + \log q)$
- For oSPFSS.FullEval , we need at most $2(ct)^2(4n^3N) / \mu$ PRG operations.

In Section 6, we provide concrete estimates.

4 PCGs for Matrix Multiplication from Twisted QA-SD

This section gives a twisted analogue of the QA-SD-based construction in Section 3. The two constructions follow the same general strategy where the parties sample sparse errors, use SPFSS to share their pairwise products, and apply public linear operations during expansion. There are two changes that are necessary. First, the twisted group algebra is noncommutative, so the two pseudorandom operands must use opposite orientations. Second, the algebra isomorphism $\Psi : \mathcal{A} \rightarrow \mathcal{M}_n(\mathbb{F}_q)^N$ converts multiplication in $\mathcal{A}$ directly into N component-wise matrix multiplications. We therefore do not use the polynomial encoding and trace map from the preceding QA-SD construction.³

The opposite orientations are what keep the secret sparse factors adjacent. For a uniform $a \in \mathcal{A}$, and $e_L, e_R, f_L, f_R \leftarrow \mathcal{D}_t(\mathcal{A})$, define

$$x_L = ae_L + f_L, \quad x_R = e_Ra + f_R.$$

Then

$$\begin{aligned} x_L x_R &= (ae_L + f_L)(e_Ra + f_R) \\ &= a(e_L e_R)a + a(e_L f_R) + (f_L e_R)a + f_L f_R. \end{aligned} \tag{7}$$

No factors have been commuted. Each secret product in the last line has weight at most t^2 in the twisted-group basis. Four SPFSS instances can therefore share these products, after which the parties apply the public left and right multiplications by a locally.

³Color convention. We use for left-oriented expressions, for right-oriented expressions, for the forward invocation, for the reverse invocation

4.1 Construction

The construction of the PCG $\text{PCG} = (\text{Gen}, \text{Expand})$ for producing N additively shared $n \times n$ matrix triples is presented in Figure 3. We first describe the ordered product subroutine used by its generation and expansion algorithms.

Fix the parameter ensemble from the preliminaries. Thus $q = q(\lambda)$ is a prime power, $n = n(\lambda)$ divides $q - 1$, and $H = H(\lambda)$ is a finite abelian group of order $N = N(\lambda)$ whose exponent divides $q - 1$. Write

$$G := (\mathbb{Z}/n\mathbb{Z})^2 \times H, \quad \mathcal{A} := \mathbb{F}_q^\alpha[G], \quad L := |G| = n^2 N.$$

We assume that $L \log q = \text{poly}(\lambda)$ and that the descriptions of the groups, the cocycle, and the algorithms used below have polynomial size. Let $\mathcal{D}_t(\mathcal{A})$ be the exact-weight distribution in the twisted basis defined in the preliminaries, where $1 \leq t = t(\lambda) \leq L$.

The construction and proofs extend directly to any finite abelian group A of order n and exponent dividing $q - 1$ in place of $\mathbb{Z}/n\mathbb{Z}$, using the corresponding product Weyl cocycle, and to module rank $c > 2$ by taking $x_L = \sum_i a_i e_L^i$ and $x_R = \sum_i e_R^i a_i$. We use both extensions in Section 6.

By Corollary 2.20, there is an efficiently computable algebra isomorphism

$$\Psi : \mathcal{A} \longrightarrow \mathcal{M}_n(\mathbb{F}_q)^N.$$

In particular, $\Psi(xy) = \Psi(x)\Psi(y)$. Multiplication on the right is performed separately in each matrix component. The parameter ensemble fixes the descriptions of $\mathcal{A}$, $\mathcal{D}_t(\mathcal{A})$, Ψ , Ψ^{-1} , and the SPFSS scheme. These descriptions are available to the algorithms and are not sampled separately for each seed pair.

The target is the batched matrix-triple correlation $\mathcal{C}_{\text{Mat},n,N}$ from Definition 2.10. For the proof, it is convenient to use the following equivalent algebra-level description. Sample

$$A_0, A_1, B_0, B_1, C_0 \xleftarrow{\$} \mathcal{A}$$

independently and uniformly, set

$$C_1 := (A_0 + A_1)(B_0 + B_1) - C_0, \tag{8}$$

and give

$$\mathbf{R}_\sigma := (\Psi(A_\sigma), \Psi(B_\sigma), \Psi(C_\sigma))$$

to P_σ for $\sigma \in \{0, 1\}$. Since Ψ is a linear bijection and an algebra isomorphism, this has exactly the distribution of $\mathcal{C}_{\text{Mat},n,N}$.

We next describe how to share the sparse products in Equation (7).

Lemma 4.1 (Sharing an ordered sparse product). Let

$$r = \sum_{i=1}^t r_i u_{g_i}, \quad s = \sum_{j=1}^t s_j u_{h_j}$$

be elements of $\mathcal{A}$ of weight exactly t . Define $F_{r,s} : G \rightarrow \mathbb{F}_q$ by

$$F_{r,s}(x) := \sum_{\substack{i,j \in [t] \\ g_i + h_j = x}} r_i s_j \alpha(g_i, h_j).$$

Then

$$rs = \sum_{x \in G} F_{r,s}(x) u_x,$$

and $F_{r,s}$ is represented as a sum of exactly t^2 point functions, allowing repeated locations. Consequently, one SPFSS instance over domain G gives additive shares $[rs]_0, [rs]_1 \in \mathcal{A}$ satisfying

$$[rs]_0 + [rs]_1 = rs.$$

Proof. The multiplication rule in $\mathcal{A}$ gives

$$rs = \sum_{i,j \in [t]} r_i s_j \alpha(g_i, h_j) u_{g_i + h_j}.$$

For each $x \in G$, collecting all terms whose basis index satisfies $g_i + h_j = x$ gives

$$F_{r,s}(x) = \sum_{\substack{i,j \in [t] \\ g_i + h_j = x}} r_i s_j \alpha(g_i, h_j).$$

Consequently,

$$rs = \sum_{x \in G} F_{r,s}(x) u_x.$$

For FSS key generation, we retain the uncollected multiset of t^2 point functions. Collisions and cancellations therefore reveal neither a smaller support nor a smaller number of summands through the key format.

Generate SPFSS keys for $F_{r,s}$, and let $y_0, y_1 : G \rightarrow \mathbb{F}_q$ be their full-domain evaluations. By FSS correctness,

$$y_0 + y_1 = F_{r,s}.$$

Set

$$[rs]_0 := \sum_{x \in G} y_0(x) u_x, \quad [rs]_1 := \sum_{x \in G} y_1(x) u_x.$$

It follows that

$$[rs]_0 + [rs]_1 = \sum_{x \in G} F_{r,s}(x) u_x = rs.$$

□

The order in (r, s) is important because the cocycle factors $\alpha(g, h)$ and $\alpha(h, g)$ may differ, even though the underlying group is abelian.

By Lemma 2.7, correctness and simulation security of SPFSS already imply the pseudorandom full-domain shares used in the output-distribution argument below.

We first define the oriented-product subroutine used by the final generator. It is parameterized by a common multiplier $a \in \mathcal{A}$ because the matrix-triple generator invokes it twice, with opposite party roles. The construction appears in Figure 2.

Figure 2: The oriented-product subroutine

PARAMETERS: Security parameter λ , algebra $\mathcal{A}$, exact noise weight t , and the selected SPFSS scheme.

COMMON INPUT: A multiplier $a \in \mathcal{A}$. In the final PCG, this multiplier is sampled by the outer generator and included in both parties' seeds; it is not a public input.

CORRELATION: The left and right roles receive outputs (x_L, z_L) and (x_R, z_R) , respectively, satisfying

$$z_L + z_R = x_L x_R,$$

where

$$x_L = ae_L + f_L, \quad x_R = e_R a + f_R.$$

Gen_{Ord}($1^\lambda; a$):

1. Sample independently

$$e_L, f_L, e_R, f_R \xleftarrow{\$} \mathcal{D}_t(\mathcal{A}).$$

2. For each ordered pair

$$(r, s) \in \{(e_L, e_R), (e_L, f_R), (f_L, e_R), (f_L, f_R)\},$$

write

$$r = \sum_{i=1}^t r_i u_{g_i}, \quad s = \sum_{j=1}^t s_j u_{h_j},$$

and use fresh independent coins to generate

$$(K_L^{r,s}, K_R^{r,s}) \xleftarrow{\$} \text{SPFSS.Gen} \left(1^\lambda, (g_i + h_j)_{(i,j) \in [t]^2}, (r_i s_j \alpha(g_i, h_j))_{(i,j) \in [t]^2} \right).$$

The left role receives $K_L^{r,s}$ and the right role receives $K_R^{r,s}$.

3. Set

$$k_L := (e_L, f_L, (K_L^{r,s})_{r,s})$$

and

$$k_R := (e_R, f_R, (K_R^{r,s})_{r,s}).$$

Output (k_L, k_R) .

Expand_{Ord}(a, ρ, k_ρ):

1. Parse k_ρ according to the role $\rho \in \{L, R\}$.

2. Compute

$$x_\rho := \begin{cases} a e_L + f_L, & \rho = L, \\ e_R a + f_R, & \rho = R. \end{cases}$$

3. Fully evaluate the four SPFSS keys and decode the resulting vectors as the algebra shares

$$[e_L e_R]_\rho, [e_L f_R]_\rho, [f_L e_R]_\rho, [f_L f_R]_\rho$$

from Lemma 4.1.

4. Compute

$$z_\rho := a [e_L e_R]_\rho a + a [e_L f_R]_\rho + [f_L e_R]_\rho a + [f_L f_R]_\rho. \quad (9)$$

5. Output (x_ρ, z_ρ) .

The product shares reconstruct correctly because

$$\begin{aligned} z_L + z_R &= a e_L e_R a + a e_L f_R + f_L e_R a + f_L f_R \\ &= (a e_L + f_L)(e_R a + f_R) \\ &= x_L x_R. \end{aligned}$$

The matrix-triple generator invokes the subroutine once in each orientation. In the forward instance, P_0 has the left role and P_1 has the right role, producing shares of A_0B_1 . In the reverse instance, P_1 has the left role and P_0 has the right role, producing shares of A_1B_0 . Figure 3 gives the resulting PCG.

Figure 3: PCG for matrix triples from twisted QA-SD

PARAMETERS: Security parameter λ , algebra $\mathcal{A} = \mathbb{F}_q^\alpha[(\mathbb{Z}/n\mathbb{Z})^2 \times H]$, exact noise weight t , algebra isomorphism $\Psi : \mathcal{A} \rightarrow \mathcal{M}_n(\mathbb{F}_q)^N$, and the selected SPFSS scheme.

PUBLIC INPUT: None. The uniform multipliers $a^\rightarrow, a^\leftarrow$ are sampled by Gen and included in both parties' seeds.

CORRELATION: After expansion, P_σ receives

$$R_\sigma := (\Psi(A_\sigma), \Psi(B_\sigma), \Psi(C_\sigma)),$$

where $A_\sigma, B_\sigma, C_\sigma \in \mathcal{A}$ and

$$C_0 + C_1 = (A_0 + A_1)(B_0 + B_1).$$

Equivalently, the parties receive additive shares of N component-wise $n \times n$ matrix triples.

Gen(1^λ):

1. Sample

$$a^\rightarrow, a^\leftarrow \xleftarrow{\$} \mathcal{A}$$

independently.

2. With independent randomness, run

$$(k_L^\rightarrow, k_R^\rightarrow) \xleftarrow{\$} \text{Gen}_{\text{Ord}}(1^\lambda; a^\rightarrow),$$

$$(k_L^\leftarrow, k_R^\leftarrow) \xleftarrow{\$} \text{Gen}_{\text{Ord}}(1^\lambda; a^\leftarrow).$$

3. Set

$$k_0 := (a^\rightarrow, a^\leftarrow, k_L^\rightarrow, k_R^\leftarrow),$$

$$k_1 := (a^\rightarrow, a^\leftarrow, k_R^\rightarrow, k_L^\leftarrow).$$

Output (k_0, k_1) .

Expand(σ, k_σ):

1. Parse k_σ as above.

2. If $\sigma = 0$, compute

$$(A_0, U_0) := \text{Expand}_{\text{Ord}}(a^\rightarrow, L, k_L^\rightarrow),$$

$$(B_0, V_0) := \text{Expand}_{\text{Ord}}(a^\leftarrow, R, k_R^\leftarrow).$$

If $\sigma = 1$, compute

$$(B_1, U_1) := \text{Expand}_{\text{Ord}}(a^\rightarrow, R, k_R^\rightarrow),$$

$$(A_1, V_1) := \text{Expand}_{\text{Ord}}(a^\leftarrow, L, k_L^\leftarrow).$$

3. Set

$$C_\sigma := A_\sigma B_\sigma + U_\sigma + V_\sigma.$$

4. Output

$$R_\sigma := (\Psi(A_\sigma), \Psi(B_\sigma), \Psi(C_\sigma)).$$

Correctness. We first verify that the two outputs of an oriented-product instance add to the ordered product. We then use the forward and reverse instances to prove that the outputs of Figure 3 satisfy the matrix-triple relation. The computational indistinguishability required by the PCG correctness definition is proved in Theorem 4.5.

Lemma 4.2 (Oriented-product relation). The outputs of $\text{Expand}_{\text{Ord}}$ satisfy

$$z_L + z_R = x_L x_R.$$

Proof. Lemma 4.1 gives $[rs]_L + [rs]_R = rs$ for each of the four ordered pairs. Summing Equation (9) over the parties therefore gives

$$a(e_L e_R) a + a(e_L f_R) + (f_L e_R) a + f_L f_R.$$

Equation (7) identifies this expression with $x_L x_R$. □

The forward instance in Figure 3 gives

$$U_0 + U_1 = A_0 B_1,$$

and the reverse instance gives

$$V_0 + V_1 = A_1 B_0.$$

Consequently,

$$\begin{aligned} C_0 + C_1 &= A_0 B_0 + A_1 B_1 + (U_0 + U_1) + (V_0 + V_1) \\ &= A_0 B_0 + A_0 B_1 + A_1 B_0 + A_1 B_1 \\ &= (A_0 + A_1)(B_0 + B_1). \end{aligned}$$

The order of the four terms is unchanged. Applying Ψ preserves the identity in every matrix component.

4.2 Security

The four function descriptions used in one oriented-product instance are correlated because they depend on the same sparse errors. Lemma 2.7 turns correctness and simulation security into pseudorandom full-domain evaluations for each instance. Since the four key pairs are generated with independent coins, Remark 2.3 permits their keys, or their full-domain evaluations, to be replaced jointly in the hybrids below.

We first establish the security of the oriented-product subroutine. Its ideal correlation samples

$$x_L, x_R, z_L \xleftarrow{\$} \mathcal{A}, \quad z_R := x_L x_R - z_L,$$

and gives (x_L, z_L) to the left party and (x_R, z_R) to the right party. Given the left output, the reverse sampler chooses $\tilde{x}_R \xleftarrow{\$} \mathcal{A}$ and returns

$$(\tilde{x}_R, x_L \tilde{x}_R - z_L).$$

Given the right output, it chooses $\tilde{x}_L \xleftarrow{\$} \mathcal{A}$ and returns

$$(\tilde{x}_L, \tilde{x}_L x_R - z_R).$$

The order in the last expression is part of the correlation.

Lemma 4.3 (Output distribution of the oriented-product subroutine). Assume that Assumption 2.23 holds for $(\mathcal{A}, \mathcal{D}_t)$ and that the selected SPFSS scheme is correct and simulation secure for sums of exactly t^2 point functions. Then, for $a \xleftarrow{\$} \mathcal{A}$, the joint distribution of a and the two expanded outputs of $\text{Gen}_{\text{Ord}}(1^\lambda; a)$ is computationally indistinguishable from a together with an independent sample from the ideal oriented-product correlation.

Proof. By Lemma 4.2, we may write the right product share exactly as

$$z_R = x_L x_R - z_L.$$

Thus, for the output-distribution argument, the right product share can be recomputed from the two operands and the left product share.

By Lemma 2.7 and Remark 2.3, replace the four left-role SPFSS full-domain evaluations jointly by independent uniform vectors. Under the coefficient-vector identification $\mathbb{F}_q^G \cong \mathcal{A}$, write the corresponding independent uniform algebra elements as

$$U_{e,e}, \quad U_{e,f}, \quad U_{f,e}, \quad U_{f,f} \xleftarrow{\$} \mathcal{A}.$$

The left product share is then replaced by

$$Z := aU_{e,e}a + aU_{e,f} + U_{f,e}a + U_{f,f}.$$

Conditioned on all other variables, $U_{f,f}$ is uniform and enters this expression additively with coefficient one. Consequently, Z is uniform in $\mathcal{A}$ independently of (a, x_L, x_R) . The resulting distribution is

$$(a, x_L, x_R, Z, x_L x_R - Z), \quad Z \xleftarrow{\$} \mathcal{A}.$$

Apply left twisted QA-SD to replace

$$x_L = ae_L + f_L$$

with a uniform algebra element. By Proposition 2.25, the right variant holds for the same parameters, so we may then replace

$$x_R = e_R a + f_R$$

with an independent uniform element. After each replacement, recompute the last share as

$$x_L x_R - Z.$$

This is efficient postprocessing of the corresponding twisted QA-SD challenge. The final distribution is the ideal oriented-product correlation, independently of a . $\square$

Lemma 4.4 (Seed security of the oriented-product subroutine). Suppose that the SPFSS scheme is correct and simulation secure for sums of exactly t^2 point functions and that Assumption 2.23 holds for $(\mathcal{A}, \mathcal{D}_t)$. Sample $a \xleftarrow{\$} \mathcal{A}$, run $\text{Gen}_{\text{Ord}}(1^\lambda; a)$, and include a in both parties' seeds. The resulting subroutine satisfies the PCG seed-security experiment for the ideal oriented-product correlation.

Proof. Suppose first that the left party is corrupted. The real view is

$$(a, k_L, x_R, z_R).$$

Replace z_R exactly by

$$x_L x_R - z_L,$$

where (x_L, z_L) is obtained by expanding the corrupted seed. Lemma 4.2 shows that this does not change the distribution.

Replace the four SPFSS keys in k_L one at a time with simulated keys. After every replacement, recompute z_L from the current corrupted seed and set

$$z_R := x_L x_R - z_L.$$

The simulation-security part of Remark 2.3, together with efficient postprocessing, justifies these hybrids. Once all four keys are simulated, the corrupted seed is independent of e_R, f_R .

Right twisted QA-SD now lets us replace

$$x_R = e_R a + f_R$$

with an independent $\tilde{x}_R \xleftarrow{\$} \mathcal{A}$. Set

$$\tilde{z}_R := x_L \tilde{x}_R - z_L.$$

To reverse the FSS hybrids, sample independent auxiliary sparse elements

$$e_R^*, f_R^* \xleftarrow{\$} \mathcal{D}_t(\mathcal{A}).$$

These elements are independent of $\tilde{x}_R$ and are used only to define the functions underlying the restored SPFSS keys.

Restore the four real corrupted keys in reverse order. After each restoration, recompute z_L from the current corrupted seed and reset

$$\tilde{z}_R := x_L \tilde{x}_R - z_L.$$

The simulation-security part of Remark 2.3 justifies each reverse replacement. At the end, k_L has its real distribution, because its restored keys use fresh errors distributed exactly as the honest errors in real seed generation. The other operand is uniform, and its product share is determined by the oriented-product equation. This is precisely the reverse-sampling experiment given the real corrupted seed. This reverse-key step is the same proof step used in [BCCD23, Theorem 22].

For a corrupted right party, start from

$$(a, k_R, x_L, z_L)$$

and replace z_L exactly by

$$x_L x_R - z_R,$$

where (x_R, z_R) is obtained by expanding the corrupted seed. Simulate the four keys in k_R , recomputing z_R and the displayed honest share after each replacement. The corrupted seed is then independent of e_L, f_L , so left twisted QA-SD lets us replace x_L by an independent $\tilde{x}_L \xleftarrow{\$} \mathcal{A}$. Set

$$\tilde{z}_L := \tilde{x}_L x_R - z_R.$$

Sample independent auxiliary sparse elements $e_L^*, f_L^* \xleftarrow{\$} \mathcal{D}_t(\mathcal{A})$ and use them only to restore the four real keys in reverse order. Recompute the two product shares after every restoration. The final corrupted seed has its real distribution, while the other operand is uniform and its share is given by the reverse sampler. This proves both corruption cases. $\square$

The two preceding lemmas use both orientations of twisted QA-SD. Proposition 2.25 proves their equivalence for the Weyl cocycle and exact-weight distribution fixed in the preliminaries, so the result requires only the left assumption. A different cocycle, basis, or noise distribution would require a separate proof or assumption for the right-oriented distribution.

Theorem 4.5 (Matrix-triple PCG from twisted QA-SD). Suppose that Assumption 2.23 holds for $(\mathcal{A}, \mathcal{D}_t)$ and that the selected SPFSS scheme is correct and simulation secure for sums of exactly t^2 point functions. Then the algorithms in Figure 3 form a secure two-party PCG for $\mathcal{C}_{\text{Mat},n,N}$. Their expanded outputs are computationally indistinguishable from N independent additively shared $n \times n$ matrix triples over $\mathbb{F}_q$.

Proof. Lemma 4.2 and the calculation following it prove the required algebraic relation. It remains to prove the output distribution and seed security.

Output distribution. Replace the forward and reverse oriented-product outputs, one at a time, with their ideal correlations. Lemma 4.3 justifies both hybrids. The four operands A_0, A_1, B_0, B_1 are then independent and uniform in $\mathcal{A}$. In the forward ideal correlation, U_0 is uniform independently of both operands and of the reverse instance. Hence

$$C_0 = A_0 B_0 + U_0 + V_0$$

is uniform even after fixing (A_0, A_1, B_0, B_1) . The algebraic relation determines C_1 . This is exactly the algebra-level target distribution in Equation (8). Since Ψ is a linear bijection and an algebra isomorphism, the matrix outputs are computationally indistinguishable from N independent ideal matrix triples.

Seed security for corrupt P_0 . At the algebra level, the real view is

$$(a^{\rightarrow}, a^{\leftarrow}, k_L^{\rightarrow}, k_R^{\leftarrow}, A_1, B_1, C_1).$$

Apply Lemma 4.4 to the forward instance with P_0 as the corrupted left party, carrying the independent reverse instance as auxiliary information. This retains the real $k_L^{\rightarrow}$, expands it to (A_0, U_0) , samples $B_1 \xleftarrow{\$} \mathcal{A}$, and sets

$$U_1 := A_0 B_1 - U_0.$$

After the replacement, recompute the honest party's share C_1 .

Next apply the same lemma to the reverse instance with P_0 as the corrupted right party. This retains the real $k_R^{\leftarrow}$, expands it to (B_0, V_0) , samples $A_1 \xleftarrow{\$} \mathcal{A}$, and sets

$$V_1 := A_1 B_0 - V_0.$$

The two newly sampled operands are independent. The honest party's third share is now

$$\begin{aligned} C_1 &= A_1 B_1 + U_1 + V_1 \\ &= A_1 B_1 + (A_0 B_1 - U_0) + (A_1 B_0 - V_0) \\ &= (A_0 + A_1)(B_0 + B_1) - (A_0 B_0 + U_0 + V_0) \\ &= (A_0 + A_1)(B_0 + B_1) - C_0. \end{aligned}$$

Thus the final hybrid retains P_0 's real combined seed and reverse-samples P_1 's output exactly from P_0 's expanded output.

Seed security for a corrupted P_1 . The real view is

$$(a^{\rightarrow}, a^{\leftarrow}, k_R^{\rightarrow}, k_L^{\leftarrow}, A_0, B_0, C_0).$$

Apply Lemma 4.4 first to the forward instance with P_1 as the corrupted right party. This retains $k_R^\rightarrow$, expands it to (B_1, U_1) , samples $A_0 \xleftarrow{\$} \mathcal{A}$, and sets

$$U_0 := A_0 B_1 - U_1.$$

Apply the lemma next to the reverse instance with P_1 as the corrupted left party. This retains $k_L^\leftarrow$, expands it to (A_1, V_1) , samples $B_0 \xleftarrow{\$} \mathcal{A}$, and sets

$$V_0 := A_1 B_0 - V_1.$$

After each replacement, recompute the honest share C_0 . The final hybrid has independent $A_0, B_0 \xleftarrow{\$} \mathcal{A}$ and

$$\begin{aligned} C_0 &= A_0 B_0 + U_0 + V_0 \\ &= A_0 B_0 + (A_0 B_1 - U_1) + (A_1 B_0 - V_1) \\ &= (A_0 + A_1)(B_0 + B_1) - (A_1 B_1 + U_1 + V_1) \\ &= (A_0 + A_1)(B_0 + B_1) - C_1. \end{aligned}$$

This is exactly the reverse sampler given P_1 's real combined seed. $\square$

4.3 Efficiency analysis of the twisted construction

Cost conventions. Seed lengths are measured in bits. Local expansion work is reported in calls to the underlying cryptographic primitives and, where useful, in operations over $\mathbb{F}_q$; the latter use the unit-cost field-operation model. Seed lengths and expansion costs are per party, whereas generation cost refers to the total work performed by Gen. Communication and round complexity are accounted for separately when Gen is instantiated by an interactive distributed setup. Section 6.1 gives better estimates for the optimized instantiations, and Section 6.5 gives concrete estimates.

Recall that

$$L = \dim_{\mathbb{F}_q}(\mathcal{A}) = n^2 N.$$

One execution of the construction produces N additively shared $n \times n$ matrix triples over $\mathbb{F}_q$. We use fixed-length coordinate-wise encodings and write

$$d_L := \lceil \log_2 L \rceil, \quad \ell_q := \lceil \log_2 q \rceil, \quad \ell_{q-1} := \lceil \log_2(q-1) \rceil.$$

Thus, one element of $\mathcal{A}$ occupies $L\ell_q$ bits.

An exact-weight- t algebra element can be encoded by its support and its t nonzero coefficients using

$$s_{\text{sp}}(L, t, q) := \left\lceil \log_2 \binom{L}{t} \right\rceil + t\ell_{q-1} \leq t(d_L + \ell_{q-1}) \quad (10)$$

bits. We will also use the sharper estimate

$$s_{\text{sp}}(L, t, q) \leq \left\lceil t \log_2 \left(\frac{eL}{t} \right) \right\rceil + t\ell_{q-1}.$$

Seed size. One invocation of Gen_{Ord} generates four SPFSS key pairs and four sparse algebra elements in total. Each role receives four SPFSS keys and two sparse algebra elements. Since the matrix-triple generator invokes the subroutine twice, each party receives eight SPFSS keys and four sparse algebra elements in total.

Let

$$s_{\text{SPFSS}}(\lambda, L, t^2, q)$$

denote the size of one party's key for an SPFSS instance that shares a sum of exactly t^2 point functions over domain G and range $(\mathbb{F}_q, +)$. Excluding the two uniform multipliers, the total inner contribution to either party's seed is

$$S_{\text{inner}} := 8s_{\text{SPFSS}}(\lambda, L, t^2, q) + 4s_{\text{sp}}(L, t, q) + O(1).$$

The two uniform multipliers $a^{\rightarrow}, a^{\leftarrow} \in \mathcal{A}$ use $2L\ell_q$ bits. Consequently, the complete seed length per party is

$$S_{\text{seed}} := 2L\ell_q + S_{\text{inner}}.$$

A conservative DPF instantiation. Section 6.3 gives optimized multi-point FSS instantiations that improve the full-domain expansion cost. For reference, consider the PRG-based DPF of [BGI18, Theorem 3.3]. This gives a concrete instantiation of the efficiency formulas whenever the selected DPF is correct and simulation secure; Lemma 2.7 then supplies pseudorandom full-domain evaluations. After embedding the domain G into $\{0, 1\}^{d_L}$, one DPF key for a point function with domain size L and range $\mathbb{F}_q$ has length at most

$$s_{\text{DPF}}(\lambda, L, q) := d_L(\lambda + 2) + \lambda + \ell_q.$$

Implementing one SPFSS key as the concatenation of t^2 independently generated DPF keys gives

$$s_{\text{SPFSS}}(\lambda, L, t^2, q) \leq t^2 s_{\text{DPF}}(\lambda, L, q).$$

Therefore,

$$S_{\text{inner}}^{\text{DPF}} \leq 8t^2 s_{\text{DPF}}(\lambda, L, q) + 4s_{\text{sp}}(L, t, q) + O(1).$$

In particular,

$$S_{\text{inner}}^{\text{DPF}} = O(t^2(\lambda \log L + \lambda + \log q) + t \log(L/t) + t \log q).$$

A sufficient condition for the per-instance contribution to be sublinear in one algebra element is

$$t^2(\lambda \log L + \lambda + \log q) = o(L\ell_q). \quad (11)$$

This condition is obtained from the conservative independent-DPF realization.

Output size and stretch. Each party obtains one share of each of the three algebra elements in the matrix triple. Equivalently, it obtains shares of N matrices in each of the three output coordinates. Hence its output contains $3L$ field elements and has length

$$S_{\text{out}} = 3Nn^2\ell_q = 3L\ell_q \quad \text{bits.}$$

Since one execution produces N matrix triples, the output length per matrix triple and per party is

$$\frac{S_{\text{out}}}{N} = 3n^2\ell_q \quad \text{bits.}$$

The per-party stretch is

$$\text{stretch}_1 := \frac{S_{\text{out}}}{S_{\text{seed}}} = \frac{3L\ell_q}{2L\ell_q + S_{\text{inner}}}.$$

Thus,

$$\text{stretch}_1 > 1 \iff S_{\text{inner}} < L\ell_q.$$

Under Equation (11),

$$\text{stretch}_1 \longrightarrow \frac{3}{2}.$$

The limiting factor $3/2$ results from the two uniform full-length multipliers, which already occupy $2L\ell_q$ bits in each party's seed.

Corollary 4.6 (Polynomially batched generation). Under the assumptions of Theorem 4.5, let $\mu = \mu(\lambda)$ be polynomially bounded. There is a two-party PCG for $\mathcal{C}_{\text{Mat},n,N}^\mu$ whose generator samples

$$a^{\rightarrow}, a^{\leftarrow} \xleftarrow{\$} \mathcal{A}$$

once and generates μ independent inner seed pairs using these multipliers. The multipliers are included once in each party's batched seed. The expanded outputs are computationally indistinguishable from μ independent samples of $\mathcal{C}_{\text{Mat},n,N}$, and the complete batched seeds satisfy the PCG seed-security definition.

Proof. We first observe that polynomially many twisted QA-SD samples may share one uniformly sampled multiplier. Consider

$$(a, ae_1 + f_1), \dots, (a, ae_\mu + f_\mu),$$

where $a \xleftarrow{\$} \mathcal{A}$ and the pairs (e_j, f_j) are sampled independently. Replace the samples by independent uniform algebra elements one at a time. A reduction for the j -th transition embeds its twisted QA-SD challenge at position j and, using the multiplier contained in the challenge, samples every other error pair itself. Since μ is polynomially bounded, the hybrid preserves computational indistinguishability. The same argument applies to the right-oriented distribution and to both multipliers.

For the output distribution, repeat the two oriented-product hybrids used in the proof of Theorem 4.5 for the μ inner instances one at a time. All other instances can be generated efficiently from the common multipliers and carried as auxiliary information.

For seed security, fix the corrupted party and repeat the single-instance seed-security proof for each inner seed pair. At each step, the reduction retains the common multipliers, embeds its challenge in the selected inner instance, and samples every other inner instance with fresh randomness. The correlated-key hybrids remain valid by the simulation-security part of Remark 2.3, since the SPFSS key-generation coins are independent conditional on their function descriptions. The final distribution retains the corrupted party's complete batched seed and reverse-samples the other party's μ outputs component-wise. $\square$

Corollary 4.6 permits the multipliers to be reused within any polynomially bounded batch. It does not treat them as permanent public parameters that may be reused across an unbounded number of independent executions.

Amortized seed size under polynomial batching. The construction of Corollary 4.6 produces μN matrix triples. Its complete seed length and output length per party are

$$\begin{aligned} S_{\text{seed}}^{(\mu)} &= 2L\ell_q + \mu S_{\text{inner}}, \\ S_{\text{out}}^{(\mu)} &= 3\mu L\ell_q. \end{aligned}$$

The corresponding stretch is

$$\text{stretch}_\mu = \frac{3\mu L\ell_q}{2L\ell_q + \mu S_{\text{inner}}}.$$

The amortized seed length per matrix triple and per party is

$$\frac{S_{\text{seed}}^{(\mu)}}{\mu N} = \frac{S_{\text{inner}}}{N} + \frac{2n^2\ell_q}{\mu}.$$

Equivalently,

$$\text{stretch}_\mu = \frac{3L\ell_q}{S_{\text{inner}} + 2L\ell_q/\mu}.$$

For every admissible batch size μ ,

$$\text{stretch}_\mu > 1 \iff S_{\text{inner}} < \left(3 - \frac{2}{\mu}\right) L\ell_q.$$

As μ grows within the polynomially bounded regime permitted by Corollary 4.6, the stretch approaches

$$\frac{3L\ell_q}{S_{\text{inner}}}.$$

Generation cost. Let

$$C_{\text{SPFSS}}^{\text{Gen}}(\lambda, L, t^2, q)$$

denote the cost of generating one SPFSS key pair. Let $C_{\text{unif}}(L, q)$ denote the cost of sampling one uniform element of $\mathcal{A}$, and let $C_{\text{sp-samp}}(L, t, q)$ denote the cost of sampling one element from $\mathcal{D}_t(\mathcal{A})$.

One execution of Gen performs eight SPFSS key-pair generations. It also samples two uniform algebra elements and eight sparse algebra elements, and constructs eight lists of t^2 point-function locations and payloads. Hence

$$C_{\text{Gen}} = 8C_{\text{SPFSS}}^{\text{Gen}}(\lambda, L, t^2, q) + 2C_{\text{unif}}(L, q) + 8C_{\text{sp-samp}}(L, t, q) + O(t^2).$$

Here the $O(t^2)$ term accounts for constructing the eight lists of point-function descriptions, assuming constant-time group, field, and cocycle operations.

With standard coordinate-wise sampling,

$$C_{\text{unif}}(L, q) = O(L), \quad C_{\text{sp-samp}}(L, t, q) = O(t),$$

and therefore

$$C_{\text{Gen}} = 8C_{\text{SPFSS}}^{\text{Gen}}(\lambda, L, t^2, q) + O(L + t^2).$$

Since one execution generates N matrix triples, the generation cost per matrix triple is C_{Gen}/N .

For the conservative independent-DPF realization, let $C_{\text{DPF}}^{\text{Gen}}(\lambda, L, q)$ denote the cost of generating one DPF key pair. Then

$$C_{\text{SPFSS}}^{\text{Gen}}(\lambda, L, t^2, q) \leq t^2 C_{\text{DPF}}^{\text{Gen}}(\lambda, L, q) + O(t^2),$$

and therefore

$$C_{\text{Gen}}^{\text{DPF}} \leq 8t^2 C_{\text{DPF}}^{\text{Gen}}(\lambda, L, q) + O(L + t^2).$$

For a polynomial batch of size μ , the two uniform multipliers are sampled once, whereas all sparse elements, SPFSS keys, and point-function descriptions are generated independently. Thus

$$C_{\text{Gen}}^{(\mu)} = 8\mu C_{\text{SPFSS}}^{\text{Gen}}(\lambda, L, t^2, q) + 2C_{\text{unif}}(L, q) + 8\mu C_{\text{sp-samp}}(L, t, q) + O(\mu t^2).$$

The amortized generation cost per matrix triple is $C_{\text{Gen}}^{(\mu)} / (\mu N)$.

Local expansion cost. Let

- $C_{\text{SPFSS}}^{\text{FullEval}}(\lambda, L, t^2, q)$ denote the cost of one full-domain SPFSS evaluation;
- $M_{\mathcal{A}}(L)$ denote the cost of one dense algebra multiplication;
- $M_{\mathcal{A},t}(L)$ denote an upper bound on the cost of multiplying a dense algebra element by a weight- t algebra element, in either order;
- $C_{\Psi}(n, N, q)$ denote the cost of one application of Ψ ; and
- $C_{\Psi^{-1}}(n, N, q)$ denote the cost of one application of Ψ^{-1} .

Each party performs eight full-domain SPFSS evaluations. Using the factored form of Equation (9), one oriented-product share can be computed as

$$a \left([e_L e_R]_{\rho} a + [e_L f_R]_{\rho} \right) + [f_L e_R]_{\rho} a + [f_L f_R]_{\rho}.$$

This requires three dense algebra multiplications. The two oriented-product invocations therefore require six dense algebra multiplications in total. The local product $A_{\sigma} B_{\sigma}$ requires one further dense multiplication. Computing the two local operands requires two dense-by-sparse multiplications. Finally, the party applies Ψ to A_{σ} , B_{σ} , and C_{σ} . Therefore,

$$C_{\text{Expand}} = 8C_{\text{SPFSS}}^{\text{FullEval}}(\lambda, L, t^2, q) + 7M_{\mathcal{A}}(L) + 2M_{\mathcal{A},t}(L) + 3C_{\Psi}(n, N, q) + O(L).$$

The $O(L)$ term accounts for additions, subtractions, and the assembly of algebra elements from the full-domain evaluation vectors.

In the twisted-group coefficient representation, multiplication by a weight- t element can be implemented using t shifted and rescaled copies of a length- L coefficient vector. Hence

$$M_{\mathcal{A},t}(L) = O(tL)$$

field operations. This term cannot be absorbed into $O(L)$ when t grows.

Since one expansion produces N matrix triples, the local expansion cost per matrix triple is

$$\frac{C_{\text{Expand}}}{N}.$$

For the independent-DPF realization, let $C_{\text{DPF}}^{\text{FullEval}}(\lambda, L, q)$ denote the cost of one full-domain DPF evaluation. Evaluating and adding t^2 independent DPF outputs gives

$$C_{\text{SPFSS}}^{\text{FullEval}}(\lambda, L, t^2, q) \leq t^2 C_{\text{DPF}}^{\text{FullEval}}(\lambda, L, q) + O(t^2 L).$$

Consequently,

$$\begin{aligned} C_{\text{Expand}}^{\text{DPF}} &\leq 8t^2 C_{\text{DPF}}^{\text{FullEval}}(\lambda, L, q) + O(t^2 L) \\ &\quad + 7M_{\mathcal{A}}(L) + 2M_{\mathcal{A},t}(L) \\ &\quad + 3C_{\Psi}(n, N, q) + O(L). \end{aligned}$$

Dense algebra multiplication and basis conversion. Let $\text{MM}_q(n)$ denote the cost of one $n \times n$ matrix multiplication over $\mathbb{F}_q$. Since

$$\Psi : \mathcal{A} \longrightarrow \mathcal{M}_n(\mathbb{F}_q)^N$$

is an algebra isomorphism, one dense algebra multiplication can be implemented by converting both operands, multiplying the N matrix components, and converting the result back. Thus,

$$M_{\mathcal{A}}(L) \leq 2C_{\Psi}(n, N, q) + C_{\Psi^{-1}}(n, N, q) + N\text{MM}_q(n) + O(L). \quad (12)$$

The conversion algorithm described in the preliminaries gives

$$C_{\Psi}(n, N, q) = O(n^2 T_H(N) + Nn T_n(n)),$$

and the same asymptotic bound holds for Ψ^{-1} . Whenever the selected groups admit transforms satisfying

$$T_H(N) = O(N \log N), \quad T_n(n) = O(n \log n),$$

we obtain

$$C_{\Psi}(n, N, q), C_{\Psi^{-1}}(n, N, q) = O(L(\log N + \log n)).$$

Equation (12) treats each dense algebra multiplication independently. A concrete implementation may retain intermediate elements in the Ψ -representation and reuse transformed copies of $a^{\rightarrow}$ and $a^{\leftarrow}$, thereby reducing the number of forward and inverse transforms.

For a polynomial batch of size μ , the abstract expansion cost is

$$C_{\text{Expand}}^{(\mu)} = \mu C_{\text{Expand}}.$$

This count does not include possible savings from precomputing the transforms of the reused multipliers. The batch produces μN matrix triples, so its local expansion cost per matrix triple is $C_{\text{Expand}}^{(\mu)} / (\mu N)$.

5 Attacks by Folding

Generic decoding algorithms do not directly exploit the group algebra structure of quasi-abelian codes. The main structural attack in this setting is known as folding [CT19, BCCD23, BBC⁺24a]. The core idea is to project the original decoding problem to a smaller decoding instance. In this section, we review this technique for usual group algebras, and extend it to the twisted case.

5.1 Folding Group Algebras

let G be an Abelian group, and $H \subset G$ a subgroup. We denote by $(u_g)_{g \in G}$ the distinguished basis of $\mathbb{F}_q[G]$, with

$$u_g u_h = u_{g+h}.$$

Let $(v_{\bar{g}})_{\bar{g} \in G/H}$ be a basis of $\mathbb{F}_q[G/H]$. The quotient map $G \rightarrow G/H$ extends linearly to a surjective morphism of algebras

$$\pi_H : \begin{cases} \mathbb{F}_q[G] & \longrightarrow & \mathbb{F}_q[G/H] \\ \sum_{g \in G} x_g u_g & \longmapsto & \sum_{\bar{g} \in G/H} \left(\sum_{h \in H} x_{g+h} \right) v_{\bar{g}}. \end{cases}$$

In other words, a group algebra element represented as a vector of length $|G|$ is *folded* to a vector of length $|G|/|H|$ by summing components indexed by elements of the same H -coset.

For $\mathbf{y} = (y_1, \dots, y_c) \in \mathbb{F}_q[G]^c$, we then define the folding operator with respect to H as

$$\text{Fold}_H(\mathbf{y}) := (\pi_H(y_1), \dots, \pi_H(y_c)).$$

Now, let $(\mathbf{a}, \langle \mathbf{a}, \mathbf{e} \rangle + e_0)$ be an instance of QA-SD. Applying π_H on the syndrome, and using the fact that it is an algebra homomorphism, yields

$$\pi_H \left(e_0 + \sum_{i=1}^{c-1} a_i e_i \right) = \pi_H(e_0) + \sum_{i=1}^{c-1} \pi_H(a_i) \pi_H(e_i) \in \mathbb{F}_q[G/H] = \text{Fold}_H(1, \mathbf{a}) \text{Fold}_H(\mathbf{e})^\top.$$

In other words, starting from the original QA-SD instance which is a decoding instance with code-length $c|G|$ and dimension $(c-1)|G|$, folding with respect to a subgroup H reduces to a decoding instance with code-length $c|G|/|H|$ and dimension $(c-1)|G|/|H|$. In particular, the code-rate is untouched by folding and remains equal to $1 - 1/c$. On the other hand, the Hamming weight of the folded error $\text{Fold}_H(\mathbf{e})$ cannot increase, and may even be smaller due to potential collisions of elements belonging to the same H -coset. The resulted folding weight-distribution has been studied in detail in [BBC⁺24b, Section 6.3].

5.2 Obstructions for Twisted Group Algebras

Let G be an Abelian group, let $H \subset G$ a subgroup, and let α be a normalized 2-cocycle defining the twisted group algebra $\mathbb{F}_q^\alpha[G]$. Let $(u_g)_{g \in G}$ be the chosen distinguished basis, with multiplication

$$u_g u_h = \alpha(g, h) u_{g+h}. \quad (13)$$

Similarly to the untwisted case, one may try to find a suitable morphism of algebras

$$\pi: \mathbb{F}_q^\alpha[G] \rightarrow \mathbb{F}_q^\beta[G/H],$$

where $\beta: G/H \times G/H \rightarrow \mathbb{F}_q^\times$ is a normalized 2-cocycle of G/H . In order to yield an easier decoding instance, though, π must also preserve sparsity of group algebra elements. In particular, it should map unitary elements to unitary elements. In other words, if $(v_{\bar{g}})$ denotes the distinguished basis of $\mathbb{F}_q^\beta[G/H]$, there must exist some scalar coefficients $\lambda(g) \in \mathbb{F}_q^\times$ and a global permutation σ of G/H such that

$$\pi(u_g) = \lambda(g) v_{\sigma(\bar{g})}.$$

Remark 5.1. Since α and β are normalized, it holds that $\lambda(0) = 1$.

In the following, we show that such a morphism of algebras does not exist in general.

Applying π on Equation (13), it must hold that for any $g, h \in G$,

$$\pi(u_g) \pi(u_h) = \alpha(g, h) \pi(u_{g+h}).$$

By expanding this equality, and identifying basis elements, we see that the left hand side is supported on $v_{\sigma(\bar{g}) + \sigma(\bar{h})}$ while the right hand side is supported on $v_{\sigma(\bar{g} + \bar{h})}$. In particular, σ must be a group automorphism of G/H . In particular, it simply is a relabelling of the elements of G/H . For $x \in G/H$, define

$$w_x := v_{\sigma(x)}.$$

It is clear that (w_x) forms a new $\mathbb{F}_q$ -basis of $\mathbb{F}_q[G/H]$. Moreover, for $x, y \in G/H$,

$$w_x w_y = \beta(\sigma(x), \sigma(y)) v_{\sigma(x) + \sigma(y)} = \beta(\sigma(x), \sigma(y)) w_{x+y}$$

since σ is a group homomorphism. In particular, the same underlying algebra, expressed in the basis (w_x) , is nothing but the twisted group algebra associated with $\sigma^*\beta$ defined by

$$\sigma^*\beta(x, y) := \beta(\sigma(x), \sigma(y)).$$

It is readily seen that $\sigma^*\beta$ is also a normalized 2-cocycle of G/H . After replacing β by its pullback $\sigma^*\beta$ if necessary, we may assume without loss of generality that π takes the simpler form

$$\pi(u_g) = \lambda(g)v_{\bar{g}}$$

for some application $\lambda: G \rightarrow \mathbb{F}_q^\times$.

Now, using again the multiplicativity of the putative morphism π , we must have for $g, h \in G$

$$\alpha(g, h)\lambda(g+h)v_{\overline{g+h}} = \pi(u_g u_h) = \pi(u_g)\pi(u_h) = \lambda(g)\lambda(h)\beta(\bar{g}, \bar{h})v_{\overline{g+h}},$$

i.e.,

$$\alpha^\lambda(g, h) := \alpha(g, h) \frac{\lambda(g+h)}{\lambda(g)\lambda(h)} = \beta(\bar{g}, \bar{h}), \quad (14)$$

must only depend on the H -cosets of its arguments.

Remark 5.2. For any function $\lambda: G \mapsto \mathbb{F}_q^\times$ such that $\lambda(0) = 1$, it is readily shown that α^λ is again a normalized 2-cocycle of G , which defines an isomorphic twisted group algebra $\mathbb{F}_q^{\alpha^\lambda}[G]$. We will give a generic argument in Section 5.3.

However, in general, there is no rescaling λ which makes the cocycle α^λ constant on H -cosets. In order to see that, let us define the ratio

$$\rho_\alpha(g, h) := \frac{\alpha(g, h)}{\alpha(h, g)} \in \mathbb{F}_q^\times.$$

It measures the failure of commutativity in the twisted group algebra. Indeed, for $g, h \in G$,

$$u_g u_h = \rho_\alpha(g, h) u_h u_g.$$

Furthermore, the 2-cocycle identity implies that ρ_α is multiplicative in each variable:

$$\rho_\alpha(g+h, k) = \rho_\alpha(g, k)\rho_\alpha(h, k)$$

and

$$\rho_\alpha(g, h+k) = \rho_\alpha(g, h)\rho_\alpha(g, k).$$

Finally, $\rho_\alpha(g, g) = 1$ and $\rho_\alpha(g, h) = \rho_\alpha(h, g)^{-1}$. We say that ρ_α is the *alternating bicharacter* associated with the cocycle α .

The important observation, is that it is invariant under rescaling:

$$\rho_{\alpha^\lambda} = \rho_\alpha.$$

Assume now that $g \in G$ and $h \in H$. Since α^λ is a normalized 2-cocycle which depends only on the class of its entries modulo H , it holds that

$$\alpha^\lambda(g, h) = \alpha^\lambda(g, 0) = 1$$

and similarly

$$\alpha^\lambda(h, g) = 1.$$

In particular, for $g \in G$ and $h \in H$, we have

$$\rho_{\alpha^\lambda}(g, h) = 1.$$

Let

$$\text{Reg}(\alpha) := \{h \in G \mid \alpha(g, h) = \alpha(h, g), \quad \forall g \in G\}.$$

Elements of $\text{Reg}(\alpha)$ are called α -regular (as defined for example in [DFM24]), and invariance after rescaling by λ implies that all elements of H must be regular in G . We therefore have the following necessary condition

Proposition 5.3. Let G be an Abelian group, and let $H \subset G$ be a subgroup. Let α be a normalized 2-cocycle on G . If there exists a function

$$\lambda: G \mapsto \mathbb{F}_q^\times \quad \text{with } \lambda(0) = 1$$

and a normalized cocycle on G/H such that

$$\alpha(g, h) \frac{\lambda(g+h)}{\lambda(g)\lambda(h)} = \beta(\bar{g}, \bar{h}),$$

then $H \subset \text{Reg}(\alpha)$.

In particular, if there exist $h \in H$ and $g \in G$ such that

$$\alpha(g, h) \neq \alpha(h, g)$$

then no such rescaling may exist.

Example 5.4. Let q be a prime power, and n a positive integer such that $n \mid q-1$ so that $\mathbb{F}_q$ contains a primitive n -th root of unity denoted by ω . Let $G = (\mathbb{Z}/n\mathbb{Z})^2$ and consider the Weyl twisted algebra $\mathbb{F}_q^\alpha[G]$ with cocycle

$$\alpha((i, j), (k, \ell)) = \omega^{-jk},$$

and basis $\{P^i Q^j\}_{(i,j) \in G}$ as defined in Section 2.3.2. The alternating bicharacter associated to α is therefore

$$\rho_\alpha((i, j), (k, \ell)) := \omega^{\ell i - j k}.$$

If (k, ℓ) satisfies $\rho_\alpha((i, j), (k, \ell)) = 1$ for all (i, j) , then since ω is a primitive n -th root of unity,

- choosing $(i, j) = (0, 1)$ gives $\omega^{-k} = 1$, i.e., $k = 0$.
- choosing $(i, j) = (1, 0)$ gives $\omega^\ell = 1$, i.e., $\ell = 0$.

In particular, $\text{Reg}(\alpha) = \{0\}$, i.e., the bicharacter is actually nondegenerate. Hence, no non-trivial subgroup H can be regular, and folding via a morphism of algebras cannot exist, for any subgroup $H \subset G$.

5.3 Characterization of the Obstruction via Group Cohomology

Theorem 5.3 only provides a necessary condition for the possibility of folding. It is however not sufficient, as shown in Theorem 5.5 below. The main reason is that ρ_α does not see symmetries within α .

Example 5.5. Let $q = 5$ and $G = (\mathbb{Z}/2\mathbb{Z})^2$, and consider the subgroup $H = \mathbb{Z}/2\mathbb{Z} \times \{0\}$. Define

$$\alpha((i, j), (k, \ell)) := 2^{ik}.$$

One can readily see that α is indeed a 2-cocycle on G , which only depends on the first coordinates of its entries, and by symmetry, it is clear that every element of G is regular. In particular, Theorem 5.3 does not rule out the possibility of folding.

Nevertheless, suppose there exists a normalized cocycle β on G/H , and a rescaling function $\lambda: G \rightarrow \mathbb{F}_5^\times$ with $\lambda(0) = 1$ such that

$$\alpha(g, h) \frac{\lambda(g+h)}{\lambda(g)\lambda(h)} = \beta(\bar{g}, \bar{h}), \quad \forall g, h \in G. \quad (15)$$

Let $a = (1, 0) \in H$. By applying Equation (15) on the pair (a, a) , it holds that

$$\alpha(a, a) \frac{\lambda(0)}{\lambda(a)^2} = \beta(0, 0).$$

Since $\alpha(a, a) = \alpha((1, 0), (1, 0)) = 2$, we get

$$2 \frac{1}{\lambda(a)^2} = 1$$

and $2 = \lambda(a)^2$ must be a square in $\mathbb{F}_5^\times$, which is a contradiction.

This examples shows that the alternating bicharacter does not capture the full obstruction. Group cohomology provides a convenient language for describing exactly the rescaling appearing in Equation (15).

Definition 5.6 (2-coboundary). Let $\lambda: G \rightarrow \mathbb{F}_q^\times$ be any function such that $\lambda(0) = 1$. Such functions are also called 1-cochains of G . We can associate to λ the following map

$$(\delta\lambda)(g, h) := \frac{\lambda(g)\lambda(h)}{\lambda(g+h)},$$

called the 2-coboundary of λ , and it can be easily verified that $\delta\lambda$ is a 2-cocycle on G . We denote by $B^2(G)$ the set of all 2-coboundaries on G .

Let $Z^2(G) := Z^2(G, \mathbb{F}_q^\times)$ be the set of all normalized 2-cocycles on G . It forms an Abelian group with pointwise multiplication, of which $B^2(G)$ is a subgroup. The quotient $H^2(G, \mathbb{F}_q^\times) := Z^2(G)/B^2(G)$ is known as the second cohomology group of G , with coefficients in $\mathbb{F}_q^\times$. Equation (14) precisely expresses that the existence of folding is not merely a property of a specific cocycle α , but a property of its *cohomology class*. In fact, we have the following proposition

Proposition 5.7 ([Mil26, Proposition 1.6]). Let G be a group, and α, β two normalized cocycles. Denote the distinguished bases of $\mathbb{F}_q^\alpha[G]$ and $\mathbb{F}_q^\beta[G]$ respectively by (u_g) and (v_g) . Then, there exists an $\mathbb{F}_q$ -algebra isomorphism $\mathbb{F}_q^\alpha[G] \simeq \mathbb{F}_q^\beta[G]$ such that for $g \in G$

$$u_g \mapsto \lambda(g)v_g, \quad \text{for some cochain } \lambda,$$

if and only if α and β belong to the same cohomology class. Moreover, such isomorphism preserves Hamming weights with respect to the distinguished bases.

In particular, if α is actually a coboundary, then its cohomology class is that of the trivial cocycle, and in particular $\mathbb{F}_q^\alpha[G]$ is nothing but the usual group algebra $\mathbb{F}_q[G]$, up to rescaling basis elements. Therefore, folding is defined for $\mathbb{F}_q^\alpha[G]$ exactly as in Section 5.1. More generally, from Equation (14), cocycles which enable folding with respect to a subgroup H are exactly those which arise from a cocycle on the quotient group G/H .

Definition 5.8. Let $\pi_H: G \rightarrow G/H$ denote the canonical projection, and let β be a normalized 2-cocycle on G/H . One can define its lift to G as the cocycle

$$\beta^\uparrow(g, h) := \beta(\bar{g}, \bar{h}).$$

Lifting maps coboundaries to coboundaries and therefore induces a group morphism

$$\text{Lift}_H: H^2(G/H, \mathbb{F}_q^\times) \rightarrow H^2(G, \mathbb{F}_q^\times).$$

We are now ready to state the exact characterization of the possibility to fold, via a morphism of algebras. The proof follows from the above discussion.

Theorem 5.9. Let G be a finite Abelian group, and let $H \subset G$ be a subgroup. Let $\alpha \in Z^2(G, \mathbb{F}_q^\times)$.

1. There exist a normalized 2-cocycle β on G/H and a morphism of algebras

$$\pi: \mathbb{F}_q^\alpha[G] \rightarrow \mathbb{F}_q^\beta[G/H]$$

which maps the distinguished basis of $\mathbb{F}_q^\alpha[G]$ to a rescaling of the distinguished basis of $\mathbb{F}_q^\beta[G/H]$.

2. The cohomology class of α belongs to the image of the lifting morphism:

$$[\alpha] \in \text{Im}(\text{Lift}_H).$$

5.4 Folding as a Projector

Section 5.2 and Section 5.3 explain why the direct adaptation of folding to twisted group algebras is in general not possible. Nevertheless, it turns out that asking for a morphism of algebras is too restrictive, and is not actually necessary to define folding attacks. In fact, it admits a weaker formulation in terms of projectors, which we will then adapt to twisted group algebras in Section 5.5.

Let G be an Abelian group, let $H \subset G$ be a subgroup and let $\pi_H: \mathbb{F}_q[G] \rightarrow \mathbb{F}_q[G/H]$ be the quotient folding map. Assume $|H| \in \mathbb{F}_q^\times$.⁴ The defining property of π_H is that it identifies all basis elements whose indices belong to the same H -coset:

$$\pi_H(u_{g+h}) = \pi_H(u_g), \quad \text{for all } g \in G, h \in H. \quad (16)$$

Our goal here is to look for a similar description, but within $\mathbb{F}_q[G]$ itself. Our approach rests on standard properties of finite dimensional algebras. We refer the reader to [DK94, Sections 1.6, and 1.7] for more details on the topic. More specifically, there is a canonical element of $\mathbb{F}_q[G]$ mimicking Equation (16), namely the average over the subgroup H :

$$p_H := \frac{1}{|H|} \sum_{h \in H} u_h.$$

⁴This condition is automatically satisfied in all our instantiations.

A direct computation shows that $p_H^2 = p_H$, i.e., p_H is an *idempotent* element of the group algebra, or a projector, following the terminology of [DK94], and that it is constant on H -cosets. Consider the left $\mathbb{F}_q[G]$ -module generated by p_H , namely

$$N_H := \mathbb{F}_q[G]p_H,$$

and the map

$$\omega_H := \begin{cases} \mathbb{F}_q[G] & \rightarrow N_H \\ x & \mapsto xp_H. \end{cases}$$

By associativity, it is clear that ω_H is a morphism of left $\mathbb{F}_q[G]$ -modules, and idempotence of p_H shows that $\omega_H(y) = y$ for every $y \in N_H$. Hence by [DK94, Proposition 1.6.2], there is a direct sum decomposition of $\mathbb{F}_q[G]$, of which $N_H = \mathbb{F}_q[G]p_H$ is a direct summand, and ω_H is the projector onto this summand. More precisely, we have the decomposition

$$\mathbb{F}_q[G] = \mathbb{F}_q[G]p_H \oplus \mathbb{F}_q[G](1 - p_H).$$

The component $\mathbb{F}_q[G]p_H$ is naturally a copy of $\mathbb{F}_q[G/H]$ inside $\mathbb{F}_q[G]$. Indeed, since $\mathbb{F}_q[G]$ is commutative, it is readily seen that the left-module N_H is actually an $\mathbb{F}_q[G]$ -algebra, with identity p_H .⁵ Moreover, ω_H is constant on H -cosets, and therefore the following map Θ_H is well defined

$$\Theta_H := \begin{cases} \mathbb{F}_q[G/H] & \mapsto \mathbb{F}_q[G]p_H \\ v_{\bar{g}} & \mapsto u_g p_H. \end{cases}$$

A simple computation shows that Θ_H is an isomorphism of algebras. Furthermore, if $\Omega \subset G$ is a set of representatives of the quotient G/H , it is clear that $(u_g p_H)_{g \in \Omega}$ forms an $\mathbb{F}_q$ -basis of $\mathbb{F}_q[G]p_H$. Now, for $x = \sum_{g \in G} x_g u_g \in \mathbb{F}_q[G]$ we have

$$\begin{aligned} xp_H &= \sum_{g \in \Omega} \sum_{h \in H} x_{g+h} \underbrace{u_{g+h} p_H}_{=u_g p_H} \\ &= \sum_{g \in \Omega} \left(\sum_{h \in H} x_{g+h} \right) u_g p_H. \end{aligned}$$

Hence, in the basis $(u_g p_H)_{g \in \Omega}$, the coefficient indexed by the coset $g + H$ is precisely $\sum_{h \in H} x_{g+h}$, which is exactly the folded coefficient from Section 5.1. In particular, sparsity is also preserved.

Finally, if $(\mathbf{a}, \langle \mathbf{a}, \mathbf{e} \rangle + e_0)$ is an instance of QA-SD, right multiplication by p_H yields

$$\left(e_0 + \sum_{i=1}^{c-1} a_i e_i \right) p_H = e_0 p_H + \sum_{i=1}^{c-1} a_i (e_i p_H).$$

At first glance, the public elements a_i do not appear to be folded in the expression above. However, observe that their action is now restricted to the lower-dimensional $\mathbb{F}_q[G]$ module generated by p_H . More precisely, let $L_a : x \mapsto ax$ denote the left multiplication by a on $\mathbb{F}_q[G]$, and let $L_a^{(H)}$ denote its restriction to $N_H = \mathbb{F}_q[G]p_H$. Then, the following diagram commutes:

$$\begin{array}{ccc} \mathbb{F}_q[G] & \xrightarrow{L_a} & \mathbb{F}_q[G] \\ \omega_H \downarrow & & \downarrow \omega_H \\ N_H & \xrightarrow{L_a^{(H)}} & N_H \end{array} \quad ,$$

⁵It is not a subalgebra of $\mathbb{F}_q[G]$ though, since their identity elements are not equal.

i.e., $\omega_H \circ L_a = L_a^{(H)} \circ \omega_H$. In particular, the public multiplication operator L_a in the original QA-SD instance is replaced by its induced operator $L_a^{(H)}$ on N_H . Under the algebra isomorphism $\Theta_H: \mathbb{F}_q[G/H] \rightarrow \mathbb{F}_q[G]N_H$, this induced operator corresponds exactly to multiplication by the folded element $\pi_H(a)$, namely $\Theta_H^{-1} \circ L_a^{(H)} \circ \Theta_H$ is exactly the multiplication-by- $\pi_H(a)$ operator on $\mathbb{F}_q[G/H]$. In other words, starting from a QA-SD instance, the description of folding using the algebra morphism π_H described in Section 5.1 coincides with the above description using the multiplication-by- p_H map.

Remark 5.10. When folding ordinary Abelian group algebras, this distinction is immaterial, since $\mathbb{F}_q[G]p_H$ is naturally an algebra isomorphic to $\mathbb{F}_q[G/H]$. In fact, we have $\Theta_H \circ \pi_H = \omega_H$, and the action of $\mathbb{F}_q[G]$ on $\mathbb{F}_q[G]p_H$ factors through the quotient group algebra $\mathbb{F}_q[G/H]$. Nevertheless, the existence of the quotient algebra $\mathbb{F}_q[G/H]$ is not necessary for the attack. It suffices that the action of $\mathbb{F}_q[G]$ by left-multiplication on itself descends to an action on a lower-dimensional module, through a projection which cannot increase Hamming weights in the corresponding bases. This distinction will become essential in the next section when generalizing to the twisted setting.

5.5 Extending Folding to Twisted Group Algebras

5.5.1 General Framework

Let $A := \mathbb{F}_q^\alpha[G]$ be a twisted group algebra, with twisting cocycle α . We showed in Section 5.2 that folding through algebra morphisms did not exist in general. Nevertheless, it turns out that the formulation of Section 5.4 actually extends to this setting. Indeed, assume there exists an idempotent $p \in A$, and consider the left module $N_p := Ap$ generated by p . Then, define the map

$$\omega_p := \begin{cases} A & \rightarrow N_p \\ x & \mapsto xp \end{cases}$$

Similarly as in the previous section, ω_p is a morphism of left A -modules such that $\omega_p(y) = y$ for every $y \in N_p$. Therefore, using again [DK94, Proposition 1.6.2], the left ideal N_p is a direct summand of A and as before, we have

$$A = Ap \oplus A(1 - p).$$

Furthermore, the left-module property shows that the decoding equation survives projection onto N_p . More specifically, if $(\mathbf{a}, \langle \mathbf{a}, \mathbf{e} \rangle + e_0)$ is an instance of QA-SD, then multiplying the syndrome by p yields

$$\left(e_0 + \sum_{i=1}^{c-1} a_i e_i \right) p = e_0 p + \sum_{i=1}^{c-1} a_i (e_i p).$$

However, this algebraic property is not sufficient: sparse errors in the original twisted QA-SD instance must remain sparse after projection. This motivates the following definition.

Definition 5.11 (Folding-Compatible Idempotent). Let $p \in A := \mathbb{F}_q^\alpha[G]$ be a non-zero idempotent, and let $(u_g)_{g \in G}$ be the corresponding distinguished basis. We say that p is folding-compatible if Ap admits an $\mathbb{F}_q$ -basis $(v_k)_{k \in \Omega}$ such that for every $g \in G$ there exist $\mu_g \in \mathbb{F}_q^\times$ and $\varphi(g) \in \Omega$ such that

$$u_g p = \mu_g v_{\varphi(g)}.$$

In particular, if p is folding-compatible, then for every $x \in A$, the Hamming weight of xp with respect to the basis $(v_k)_{k \in \Omega}$ is no larger than that of x .

In general, idempotents of A may not be folding-compatible. However, it turns out that the idempotent built from cyclic subgroups in [DFM24] actually are folding-compatible, with our definition. In particular, this shows that folding is always possible for twisted group algebras. Using the characterization of idempotents in [DFM24], we could also identify the optimal folding-compatible idempotents, but in order to simplify the exposition we now restrict ourselves to the Weyl twisted group algebra.

5.5.2 Folding the Weyl Twisted Group Algebra

From now on, n is a positive integer such that $n|q-1$, so that $\mathbb{F}_q$ contains ω , a primitive n -th root of unity, and we set $G_0 := (\mathbb{Z}/n\mathbb{Z})^2$. Recall from Section 2.3.2 that the Weyl twisted algebra is $A := \mathbb{F}_q^{\alpha_0}[G_0]$ with cocycle

$$\alpha_0((i, j), (k, \ell)) = \omega^{-jk}.$$

We extensively use the isomorphism $\Phi: \mathbb{F}_q^{\alpha_0}[G_0] \simeq \mathcal{M}_n(\mathbb{F}_q)$ with distinguished basis $\{P^i Q^j\}_{(i,j) \in G}$, where P and Q are the clock and shift matrices from Section 2.3.2. In particular, we will silently identify an element of $\mathbb{F}_q^{\alpha_0}[G_0]$ with the corresponding matrix.

Proposition 5.12 (Characteristics of the folded space). Let $p \in \mathbb{F}_q^{\alpha_0}[G_0]$ be a folding-compatible idempotent, and let r be its rank when seen as an element of $\mathcal{M}_n(\mathbb{F}_q)$. Then,

- (i) the dimension of the left ideal spanned by p is

$$\dim_{\mathbb{F}_q}(\mathbb{F}_q^{\alpha_0}[G_0]p) = nr.$$

- (ii) the n^2 Weyl coordinates are partitionned into exactly nr blocks of equal size n/r .

Proof. (i) After a change of basis of $\mathbb{F}_q^n$, an idempotent of rank r is represented by the matrix

$$\hat{p} := \begin{pmatrix} I_r & 0 \\ 0 & 0 \end{pmatrix}.$$

Then, $\mathcal{M}_n(\mathbb{F}_q)p$ is exactly the space of matrices whose last $n-r$ columns vanish and the statement regarding the dimension follows immediately.

- (ii) Consider the binary relation on G_0 defined by

$$g \sim g' \iff u_g p \in \mathbb{F}_q^\times u_{g'} p.$$

A simple computation shows that this defines an equivalence relation which is invariant by translations of G_0 . In particular, G_0 acts transitively on the equivalence classes, and all classes have the same size. Furthermore, the family $(u_g p)$ spans $Ap := \mathbb{F}_q^{\alpha_0}[G]p$. Since p is folding-compatible, each of these elements is a scalar multiple of some fixed basis of Ap , and by definition, two elements of G_0 are equivalent if and only if their image through p are colinear. In particular, there are $\dim_{\mathbb{F}_q}(Ap) = nr$ equivalence classes, each of them having size $\frac{n^2}{nr} = \frac{n}{r}$. □

Remark 5.13. The description of folding by folding-compatible idempotents can also be extended to the full batched algebra

$$\mathcal{A}_{n,H} = \mathbb{F}_q^{\alpha_0}[G] \otimes \mathbb{F}_q[H] \simeq \mathcal{M}_n(\mathbb{F}_q) \otimes \mathbb{F}_q[H].$$

Indeed, folding-compatible idempotents on $\mathbb{F}_q^{\alpha_0}[G]$ may be composed with the projectors representing folding along any subgroup H' of H . A rank r projector, together with a subgroup H' gives folded dimension $nr|H/H'|$ and block size $(n/r)|H'|$.

A consequence of Theorem 5.12 is that an instance of twisted-QA-SD may only be folded to a space of dimension at least n , which is obtained by rank-one idempotents. Furthermore, every folding-compatible projector of rank r will group the n^2 coordinates into nr blocks each of size $\frac{n}{r}$. In particular, the error distribution after folding will only depend on the rank of the folding-compatible projector, and all folding-compatible projectors of same rank r will yield similar instances of the decoding problem.

Therefore, it suffices to analyze the folding with respect to a folding-compatible projector of least rank. It turns out that the cyclic idempotents introduced in [DFM24] are folding-compatible, and are the direct analogues of the average operator over a subgroup recalled in Section 5.4. We will recall the construction in the case of the Weyl group algebra.

Let $H = \langle (0, 1) \rangle$ be the subgroup of $G_0 = (\mathbb{Z}/n\mathbb{Z})^2$ supported on the second coordinate. Through the isomorphism Φ , the corresponding group algebra identifies as the n -dimensional algebra generated by Q in $\mathcal{M}_n(\mathbb{F}_q)$. Observe that for every $j, \ell \in \mathbb{Z}/n\mathbb{Z}$, we have $\alpha_0((0, j), (0, \ell)) = 1$, and the cocycle α_0 becomes trivial when restricted to $H \times H$. For $s \in \mathbb{Z}/n\mathbb{Z}$, let p_s be defined as

$$p_s := \frac{1}{n} \sum_{j \in \mathbb{Z}/n\mathbb{Z}} \omega^{-js} u_{(0,j)} \simeq \frac{1}{n} \sum_{j \in \mathbb{Z}/n\mathbb{Z}} \omega^{-js} Q^j,$$

where the right-hand side is the corresponding object in $\mathcal{M}_n(\mathbb{F}_q)$. It is exactly the subgroup idempotent of $\mathbb{F}_q^{\alpha_0}[G_0]$ as defined in [DFM24, Proposition 3.2], attached to H and the root ω^s of $X^n - 1$. More precisely, we have the following proposition

Proposition 5.14 (Adapted from [DFM24, Proposition 3.2]). For every $s \in \mathbb{Z}/n\mathbb{Z}$, the element p_s defined by

$$p_s := \frac{1}{n} \sum_{j=0}^{n-1} \omega^{-js} Q^j$$

is an idempotent of $\mathbb{F}_q^{\alpha}[G_0]$, and for $s, s' \in \mathbb{Z}/n\mathbb{Z}$, then $p_s p_{s'} = \mathbb{1}_{s=s'} p_s$.

Proof. [DFM24, Proposition 3.2] shows that p_s is an idempotent of the twisted group algebra $\mathbb{F}_q^{\alpha_0}[H]$. Since it is a subalgebra of $\mathbb{F}_q^{\alpha_0}[G_0]$, the same property holds for the full Weyl twisted algebra. $\square$

Remark 5.15. When the cocycle α does not become trivial on the chosen subgroup, it may be necessary to extend the scalars to an extension of $\mathbb{F}_q$ to define these idempotents.

Proposition 5.16. For every $s \in \mathbb{Z}/n\mathbb{Z}$, the element p_s is a rank-one, folding-compatible idempotent. The family $(P^i p_s)_{i \in \mathbb{Z}/n\mathbb{Z}}$ is a basis of $\mathbb{F}_q^{\alpha_0}[G_0] p_s$, and for

$$x = \sum_{i,j} x_{i,j} P^i Q^j$$

written in the Weyl basis, one has

$$x p_s = \sum_{i \in \mathbb{Z}/n\mathbb{Z}} \left(\sum_{j \in \mathbb{Z}/n\mathbb{Z}} x_{i,j} \omega^{sj} \right) P^i p_s.$$

In particular, the projection induced by p_s reduces an element of length n^2 in the Weyl basis to an element of length n without increasing the Hamming weight.

Proof. In this proof, we extensively use the isomorphism $\mathbb{F}_q^{\alpha_0}[G_0] \simeq \mathcal{M}_n(\mathbb{F}_q)$ and we silently identify elements of the Weyl group algebra to the corresponding matrices.

We already know from Theorem 5.14 that p_s is an idempotent of $\mathbb{F}_q^{\alpha_0}[G_0]$. Let $(e_r)_{r \in \mathbb{Z}/n\mathbb{Z}}$ be the canonical basis of $\mathbb{F}_q^n$. Recall that P and Q act on e_r by

$$P e_r = \omega^r e_r, \quad \text{and} \quad Q e_r = e_{r+1},$$

and for $s \in \mathbb{Z}/n\mathbb{Z}$, define the vector

$$v_s := \frac{1}{n} \sum_{k \in \mathbb{Z}/n\mathbb{Z}} \omega^{-sk} e_k.$$

Then,

$$p_s e_r = \frac{1}{n} \sum_{j \in \mathbb{Z}/n\mathbb{Z}} \omega^{-js} Q^j e_r = \frac{1}{n} \sum_{j \in \mathbb{Z}/n\mathbb{Z}} \omega^{-js} e_{r+j} = \frac{\omega^{rs}}{n} \sum_{k \in \mathbb{Z}/n\mathbb{Z}} \omega^{-ks} e_k = \omega^{rs} v_s.$$

Hence, the image of p_s is the line spanned by v_s and therefore p_s has rank one. More precisely, consider the linear form ℓ_s defined by the row vector

$$\ell_s := \sum_{k \in \mathbb{Z}/n\mathbb{Z}} \omega^{ks} e_k^\top,$$

so that $\ell_s(v_t) = \mathbb{1}_{s=t}$. In other words, ℓ_s is the dual of v_s , and our idempotent admits the simple form $p_s = v_s \ell_s$. Therefore, for $X \in \mathcal{M}_n(\mathbb{F}_q)$, we have $X p_s = (X v_s) \ell_s$ and the map

$$\Psi_s: \begin{cases} \mathcal{M}_n(\mathbb{F}_q) p_s & \rightarrow \mathbb{F}_q^n \\ X p_s & \mapsto X v_s \end{cases}$$

is an $\mathbb{F}_q$ -linear isomorphism. Under this map, for $i \in \mathbb{Z}/n\mathbb{Z}$, we have $\Psi_s(P^i p_s) = P^i v_s$. On the other hand, observe that the r -th coordinate of $P^i v_s$ is $\frac{1}{n} \omega^{(i-s)r}$, and hence the vectors $(P^i v_s)_i$ form the column of a Vandermonde matrix (up to rescaling). In particular, they are linearly independent, and consequently $(P^i p_s)$ forms a basis of the left ideal $\mathcal{M}_n(\mathbb{F}_q) p_s \simeq \mathbb{F}_q^{\alpha_0}[G_0] p_s$.

Finally, a simple computation shows that

$$P^i Q^j p_s = \omega^{sj} P^i p_s, \quad \text{for every } i, j \in \mathbb{Z}/n\mathbb{Z}.$$

Therefore, for every element $x := \sum_{i,j \in \mathbb{Z}/n\mathbb{Z}} x_{i,j} P^i Q^j$ of $\mathbb{F}_q^{\alpha_0}[G_0]$, we have

$$x p_s = \sum_{i,j \in \mathbb{Z}/n\mathbb{Z}} x_{i,j} \omega^{sj} P^i p_s = \sum_{i \in \mathbb{Z}/n\mathbb{Z}} \left(\sum_{j \in \mathbb{Z}/n\mathbb{Z}} x_{i,j} \omega^{sj} \right) P^i p_s.$$

□

Matrix Interpretation The above description of folding the Weyl twisted algebra may seem at first glance very abstract. It turns out that through the isomorphism with $\mathcal{M}_n(\mathbb{F}_q)$, it has a simple interpretation. Indeed, for a matrix

$$X = \begin{pmatrix} | & & | \\ X_0 & \cdots & X_{n-1} \\ | & & | \end{pmatrix},$$

where X_j denotes the j -th column, we can see from the proof that $X p_s = (X v_s) \ell_s$ where

$$X v_s = \frac{1}{n} \sum_{c=0}^{n-1} \omega^{-sc} X_c,$$

and $X p_s$ is the matrix defined by

$$X p_s = \begin{pmatrix} | & | & & | \\ X v_s & \omega^s X v_s & \cdots & \omega^{s(n-1)} X_{n-1} v_s \\ | & | & & | \end{pmatrix}.$$

Finally, to get the folded coordinates of X , we need to express $X p_s$ in the basis $(P^i p_s)$ of $\mathcal{M}_n(\mathbb{F}_q) p_s$, i.e., we need to apply the inverse Ψ_s^{-1} . Let

$$B_s := \begin{pmatrix} | & | & & | \\ v_s & P v_s & \cdots & P^{n-1} v_s \\ | & | & & | \end{pmatrix} \in \mathcal{M}_n(\mathbb{F}_q)$$

be the matrix representing Ψ . Then,

$$\text{Fold}_s(X) := B_s^{-1} X v_s = B_s^{-1} \left(\frac{1}{n} \sum_{c=0}^{n-1} \omega^{-sc} X_c \right).$$

From the proof, B_s is the Vandermonde-like matrix defined by $(B_s)_{r,i} = \frac{1}{n} \omega^{(i-s)r}$ and therefore $(B_s^{-1})_{i,r} = \omega^{-(i-s)r}$. In other words, given the matrix X representing an element $x \in \mathbb{F}_q^{\alpha_0}[G_0]$, the i -th component of its folded vector is

$$\left(\text{Fold}_s(X) \right)_i = \frac{1}{n} \sum_{r \in \mathbb{Z}/n\mathbb{Z}} \omega^{-(i-s)r} \left(\sum_{c \in \mathbb{Z}/n\mathbb{Z}} X_{r,c} \omega^{-sc} \right). \quad (17)$$

Example 5.17. Take

$$n = 3, \quad q = 7, \quad \omega = 2 \in \mathbb{F}_7.$$

Then, the Weyl basis matrices are

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{pmatrix}, \quad Q = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}.$$

and consider the previous folding with $s = 0$. Then, p_0 is the projector defined by the matrix

$$p_0 = 3^{-1}(I + Q + Q^2) = 5 \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \in \mathcal{M}_3(\mathbb{F}_7),$$

and the coordinates of the folding of a matrix $X \in \mathcal{M}_3(\mathbb{F}_7)$ are

$$\left(\text{Fold}_0(X) \right)_i = 3^{-1} \sum_{r,c \in \mathbb{Z}/3\mathbb{Z}} X_{r,c} \omega^{-ir} = 5 \cdot \sum_r 2^{-ir} \left(\sum_c X_{r,c} \right), \quad \text{for } i \in \{0, 1, 2\}.$$

Consider the following explicit toy example with codeword c represented by the matrix

$$C = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 6 & 1 & 3 \end{pmatrix}$$

and let e be the sparse error $e = P + Q$ of Hamming weight 2 in the Weyl basis. It is represented by the matrix

$$E = P + Q = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 0 \\ 0 & 1 & 4 \end{pmatrix},$$

such that the noisy codeword $y = c + e$ is represented by the matrix

$$Y = C + E = \begin{pmatrix} 2 & 2 & 4 \\ 5 & 0 & 6 \\ 6 & 2 & 0 \end{pmatrix}.$$

Using Equation (17), we first sum every column of Y to get

$$Y_0 + Y_1 + Y_2 = \begin{pmatrix} 2 \\ 5 \\ 6 \end{pmatrix} + \begin{pmatrix} 2 \\ 0 \\ 2 \end{pmatrix} + \begin{pmatrix} 4 \\ 6 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 4 \\ 1 \end{pmatrix}.$$

Hence,

$$\begin{cases} \text{Fold}_0(Y)_0 = 5(1 + 4 + 1) & = 2 \\ \text{Fold}_0(Y)_1 = 5(1 + 4\omega^{-1} + \omega^{-2}) & = 4 \\ \text{Fold}_0(Y)_2 = 5(1 + 4\omega^{-2} + \omega^{-4}) & = 2 \end{cases},$$

i.e., $\text{Fold}_0(Y) = (2, 4, 2)$. We can also check that the folded error is also sparse since a similar computation gives $\text{Fold}_0(E) = (1, 1, 0)$ which has Hamming weight 2.

Remark 5.18. In this section, we considered folding along subgroup $H = \langle(0, 1)\rangle$, but we could have equivalently considered folding along subgroup $K := \langle(1, 0)\rangle$ which also gives an idempotent of rank 1. Theorem 5.12 shows that both approaches are essentially equivalent in terms of Hamming weight of the folded error.

5.5.3 Distribution of the Folded Error

In order to get a precise analysis of the behavior of decoding algorithm with respect to the folded instance, we need to analyze the weight distribution of the folded error. Let $A = \mathbb{F}_q^{\alpha_0}[G_0]$ with distinguished basis $(u_g)_{g \in G_0}$. Let p be a folding-compatible idempotent of rank r . From Theorem 5.12, the u_g are partitionned into nr blocks of equal size $\ell := \frac{n}{r}$. If $(v_k)_{k \in \Omega}$ is a basis of the left ideal Ap compatible with folding, let C_k denote the block of coordinates corresponding to v_k . By Theorem 5.11, for every $g \in C_k$, there exists $\mu_g \in \mathbb{F}_q^\times$ such that

$$u_g p = \mu_g v_k.$$

Therefore, if

$$e = \sum_{g \in S} e_g u_g \leftarrow \mathcal{D}_t(A)$$

is a uniform vector of weight t , then

$$ep = \sum_{k \in \Omega} \left(\sum_{g \in S \cap C_k} \mu_g c_g \right) v_k.$$

Since $\mu_g \neq 0$, for the fixed support S of e the coefficients $(\mu_g c_g)_{g \in S}$ are again independent and uniformly distributed over $\mathbb{F}_q^\times$. Therefore, the probability that the coefficient of v_k becomes zero depends only on $|S \cap C_k|$, and is exactly the same as for ordinary folding along a subgroup of size ℓ .

In particular, the Hamming weight of the folded error ep has *exactly* the same distribution as analyzed in [BBC⁺24a]. With our notations, we have

Proposition 5.19 (Adapted from [BBC⁺24a, Proposition 18, long version]). Let $p \in \mathbb{F}_q^{\alpha_0}[G_0]$ be a folding-compatible idempotent of rank r , and let $(v_k)_{k \in \Omega}$ be a compatible basis of the left ideal $\mathbb{F}_q^{\alpha_0}[G_0]p$. Set $\ell = \frac{n}{r}$. Let $0 \leq t \leq n^2$ and $0 \leq u \leq \min(t, nr)$. If x is uniformly distributed over the elements of $\mathbb{F}_q^{\alpha_0}[G_0]$ of Hamming weight t , then

$$\mathbb{P}_x[\text{wt}_{v_k}(xp) = u] = \frac{\binom{nr}{u} [X^t] (P_{\theta, \ell}(X)^u P_{\nu, \ell}(X)^{nr-u})}{\binom{n^2}{t} (q-1)^t}$$

where

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

and

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

are respectively the weight enumerators of the $[\ell, \ell-1]_q$ parity code $\mathcal{C}_\ell$, and its complement $\mathbb{F}_q^\ell \setminus \mathcal{C}_\ell$.

6 Optimizations and Efficiency Estimation

In Sections 3 and 4, we presented two constructions of PCGs for matrix multiplication. To keep these two sections readable, we left out the standard optimizations used in the PCG literature. These optimizations are well understood by now, and they apply almost identically to both constructions. In this section, we give an overview of these optimizations, and we apply them to our two constructions. We then select parameters for both constructions, based on the state-of-the-art cryptanalysis of QA-SD and on our new analysis of twisted QA-SD in Section 5. Finally, we report on the concrete efficiency of both constructions, using an implementation that includes all the optimizations. We consider two settings: a large 64-bit prime field, and the small field $q = 16$ used by the MAYO signature scheme, which we target in the application section.

This section does not contain any novel scientific content: it solely describes the routine optimizations applied to our constructions, selects parameters using the known attack framework and a script from a previous work [BBC⁺24a], implements the schemes, and reports on the resulting efficiency. As such, a more liberal use of AI has been made in this section. Concretely, the authors selected the optimizations from the literature and instructed an AI model (Claude Fable 5.1) to integrate all three optimizations to each of the two schemes, compute the resulting parameters, and produce a preliminary draft of the outcome. The authors verified afterwards that all optimizations were properly applied and that all calculations were correct, and thoroughly revised the draft.

The two constructions of Sections 3 and 4 follow one blueprint. Each party holds c sparse elements per operand, the pairwise products of the two parties' sparse elements are shared with SPFSS, and the shares are post-processed by public linear maps. The two constructions differ in the algebra in which the products live, in the size of the SPFSS domain, and in the post-processing: a trace contraction for the commutative construction, an algebra isomorphism Ψ for the twisted one. Everything else, in particular the choice of the underlying groups and the instantiation of SPFSS, is common to both. We therefore treat the optimizations once, in a common cost model (Section 6.1), and specialize only where the two constructions differ.

We consider three standard optimizations:

1. *Elementary abelian index groups* (Section 6.2). We replace the cyclic groups $\mathbb{Z}/n\mathbb{Z}$ and $\mathbb{Z}/N\mathbb{Z}$ by $\mathbb{Z}_2^d$ and $\mathbb{Z}_2^e$ (when q is odd), or by $\mathbb{Z}_3^d$ and $\mathbb{Z}_3^e$ (in characteristic 2). This turns every transform of the expansion into a Walsh–Hadamard transform, or its ternary version. This is the choice recently proposed for scalar PCGs by Li, Liu, Xing, Yao and Yuan [LLX⁺26], and used in the $\mathbb{F}_4$ setting of FOLEAGE [BBC⁺24a].
2. *Multi-point FSS with distributed generation* (Section 6.3). We replace the t^2 independent DPFs of each SPFSS instance by one cuckoo-hashed distributed multi-point function. This removes the factor t from the expansion cost. The fully distributed key generation of Agarwal, Raghuraman and Rindal [ARR25] keeps the setup free of a dealer.
3. *Early termination* of the DPF trees (Section 6.3). This halves the number of PRG calls over a 64-bit field, and saves even more over small fields.

We instantiate everything in two settings: the 64-bit prime field $q = 2^{64} - 2^{32} + 1$ with $n = 128$, and the field $q = 16$ of MAYO with $n = 81$, where A and H are 3-groups. Section 6.4 discusses which parameters these optimizations favor in each setting, and Section 6.5 gives concrete estimates for both constructions.

6.1 A common cost model

Groups and dimensions. Throughout, A is the *index group*: a finite abelian group of order n whose exponent divides $q - 1$. H is the *batch group*: a finite abelian group of order N whose exponent also

divides $q - 1$. The constructions as written use $A = \mathbb{Z}/n\mathbb{Z}$ and $H = \mathbb{Z}/N\mathbb{Z}$. Section 6.2 instead uses $A = \mathbb{Z}_2^d$ and $H = \mathbb{Z}_2^c$. We write

$$D := n^2 N, \quad E := n^3 N, \quad L := n^2 N,$$

D is the dimension of the sample ring $\mathbb{F}_q[A \times A \times H]$ used in Section 3. E is the dimension of the four-variable ring $\mathbb{F}_q[A \times A \times A \times H]$, over which the SPFSS instances of that construction live. L is the dimension of the twisted algebra $\mathcal{A} = \mathbb{F}_q^\alpha[A \times A \times H]$ used in Section 4, which is also its SPFSS domain. The two FSS domains differ by the factor n of the contracted index Z : this extra factor is the structural cost of working in the commutative construction.

The common shape of one expansion. Fix the module rank $c \geq 2$ (so there are $c - 1$ public multipliers besides $a_0 = 1$) and the exact noise weight t . In both constructions, computing a full matrix triple $(A_0 + A_1)(B_0 + B_1)$ needs the two cross products $A_0 B_1$ and $A_1 B_0$. Each cross product splits into c^2 products of one sparse element from each party, so each party holds

$$I := 2c^2$$

SPFSS keys, each sharing a sum of exactly t^2 point functions (Figures 1 and 3). The seed of a party therefore contains the I SPFSS keys, $2c$ descriptions of its own sparse elements, and the public multipliers (or a short seed from which they are expanded):

$$S_{\text{seed}} = I \cdot s_{\text{SPFSS}}(\lambda, \text{Dom}, t^2, q) + 2c \cdot s_{\text{sp}}(\text{Dom}', t, q) + O(\lambda), \quad \text{Dom} \in \{E, L\},$$

Here $s_{\text{sp}}(M, t, q) \leq t(\lceil \log_2 M \rceil + \lceil \log_2 q \rceil)$ is the length needed to describe a weight- t element in dimension M (Equation (10)). We have $\text{Dom}' = D$ for the commutative construction and $\text{Dom}' = L$ for the twisted one. Expansion performs I full-domain SPFSS evaluations, followed by construction-specific post-processing:

Commutative (Section 3). Each of the I shares is a coefficient vector over E . We transform it to the evaluation grid of $A \times A \times A \times H$, multiply it pointwise by the public product $a_i(X, Z, W)a_j(Z, Y, W)$, and accumulate the I results. The quantity $n \text{Tr}_Z$ is the sum over the Z -coordinate of the grid, and the decoding of Theorem 2.28 is the identity on grid values. The party also evaluates its own operands $\mathbf{A}_\sigma, \mathbf{B}_\sigma$ on the D -grid, and computes the local product $\mathbf{D}_\sigma = \mathbf{A}_\sigma \mathbf{B}_\sigma$ using N matrix products. This gives

$$\begin{aligned} \text{mul}_1 &\approx I(T^\times(E) + 2E) + 2(c - 1)D + N \text{MM}(n), \\ \text{add}_1 &\approx I(T^+(E) + E) + E + 2c T^+(D), \end{aligned} \tag{18}$$

where $T^\times(M)$ and $T^+(M)$ are the number of multiplications and additions of one transform of size M over the chosen groups, and $\text{MM}(n)$ is the cost of one $n \times n$ product.

Twisted (Section 4). Each of the I shares is a coefficient vector over L in the twisted basis. Since Ψ is an algebra isomorphism, we can do all the remaining arithmetic in the matrix domain: we convert each share once with Ψ , and the sandwich products by the public multipliers become N independent $n \times n$ products. Writing an oriented cross product as $\sum_j (\sum_i a_i [e_i e_j]) a_j$ with $a_0 = 1$, one oriented cross product costs $c(c - 1) + (c - 1) = c^2 - 1$ dense products, and the local product $A_\sigma B_\sigma$ costs one more. This gives

$$M_{\text{dense}} = 2(c^2 - 1) + 1 = 2c^2 - 1$$

dense algebra products per party. For $c = 2$ this gives back the seven products of Section 4.3. We also compute the party's own operands $x = \sum_i a_i e_i + f$ in the matrix domain: a monomial u_g of the twisted basis is a scaled permutation matrix in every slot, so computing $a_i e_i$ costs tL multiplications. This gives

$$\begin{aligned} \text{mul}_2 &\approx I T_\Psi^\times(L) + (2c^2 - 1) N \text{MM}(n) + 2(c - 1)tL, \\ \text{add}_2 &\approx I T_\Psi^+(L) + (2c^2 - 1) N \text{MM}^+(n), \end{aligned} \tag{19}$$

where $T_\Psi(L)$ is the cost of one conversion by Ψ , namely n^2 transforms of size N over H followed by Nn transforms of size n over A (Section 2.3.3), and $\text{MM}^+(n)$ is the number of additions of one $n \times n$ product that are not part of a multiply-accumulate (zero for the schoolbook algorithm). We never need the inverse conversion, since the outputs are used directly as matrices.

With the schoolbook algorithm, $\text{MM}(n) = n^3$ multiply-accumulates and $\text{MM}^+(n) = 0$. Strassen's algorithm [Str69] lowers the number of multiplications to $7^{\log_2 n-3} \cdot 8^3$ when we recurse down to 8×8 blocks. This costs $18(n/2)^2$ extra additions per recursion level. Strassen's algorithm only helps for the twisted construction, since the sandwich products dominate its cost there. In the counts of Section 6.5, we also fold the additions of the local product $\mathbf{D}_\sigma$ of the commutative construction into its multiply-accumulates.

6.2 Elementary abelian index and batch groups

Neither construction uses the cyclic structure of the groups. The commutative construction only needs that $\mathbb{F}_q[A \times A \times H]$ splits completely (Proposition 2.13), that positions add inside the group (Lemma 2.15), and the character orthogonality $\sum_{z \in A} \chi(z) = n \cdot [\chi = 1]$ behind the trace identity of Theorem 2.28. The twisted construction only needs that $\mathbb{F}_q^A[A \times A]$ is a full matrix algebra and that $\mathbb{F}_q[H]$ splits (Corollary 2.20). All of this holds for any finite abelian groups A and H of exponent dividing $q - 1$. So we are free to choose A and H , and this choice controls the cost of the transform step.

The commutative construction over $\mathbb{Z}_2^d$. Let q be odd, $n = 2^d$ and $N = 2^e$, and take $A = \mathbb{Z}_2^d$, $H = \mathbb{Z}_2^e$. The ring $\mathbb{F}_q[A \times A \times H]$ is then the multiquadratic ring $\mathbb{F}_q[X_1, \dots, X_d, Y_1, \dots, Y_d, W_1, \dots, W_e] / (X_i^2 - 1, Y_i^2 - 1, W_i^2 - 1)$, and the module-LPN assumption of Assumption 3.1 becomes the multiquadratic QA-SD assumption used for scalar PCGs in [LLX⁺26]. Rows, columns and the contracted index are all indexed by vectors of $\mathbb{Z}_2^d$. The contraction $\sum_l \mathbf{A}[i, l] \mathbf{B}[l, j]$ still runs over all n values of l , regardless of the group structure, so Theorem 2.28 and the correctness proof of Section 3 do not change, and the factor n in $n \cdot \gamma$ stays invertible. What we gain is a faster transform: the only square root of unity is -1 , every character takes values in $\{\pm 1\}$, and the four-dimensional transform over $A^3 \times H \cong \mathbb{Z}_2^{3d+e}$ becomes the Walsh–Hadamard transform,

$$T^\times(E) = 0, \quad T^+(E) = E \log_2 E,$$

computed in place by $\log_2 E$ sweeps of butterflies $(a, b) \mapsto (a + b, a - b)$. The same holds when the party evaluates its own operands on the D -grid. The pointwise stage still costs $2E$ multiplications per SPSS share, so the multiplication count of Equation (18) drops from $I(\frac{1}{2}E \log_2 E + 2E)$ to $2IE$, and $IE \log_2 E$ additions appear in its place instead. At $n = 128$, $N = 2^{16}$ and $c = 4$, this reduces the multiplication count from about $9 \cdot 10^{13}$ to about $9 \cdot 10^{12}$. The additions are cheap and vectorize well, but they become memory-bound at these sizes, so the streaming considerations of Section 6.5 matter more.

The twisted construction over $\mathbb{Z}_2^d$: the Pauli grading. For the twisted algebra, the group A enters through the cocycle, so changing A changes the algebra itself. Let q be odd and $n = 2^d$. We write $\mathcal{M}_n(\mathbb{F}_q) = \mathcal{M}_2(\mathbb{F}_q)^{\otimes d}$, and we let $\mathbf{P}_s, \mathbf{Q}_s$ be the clock and shift matrices of the s -th tensor factor: $\mathbf{P}_s = \text{diag}(1, -1)$ and $\mathbf{Q}_s = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, so that $\mathbf{P}_s \mathbf{Q}_s = -\mathbf{Q}_s \mathbf{P}_s$. For $x, y \in \mathbb{Z}_2^d$, set

$$\mathbf{B}_{x,y} := \bigotimes_{s=1}^d \mathbf{P}_s^{x_s} \mathbf{Q}_s^{y_s}.$$

These are the n^2 Pauli operators of d qubits (the case $\omega = -1$ of the Weyl basis, see [Got99]). They satisfy

$$\mathbf{B}_{x,y} \mathbf{B}_{x',y'} = (-1)^{\langle y, x' \rangle} \mathbf{B}_{x+x', y+y'}, \quad (\mathbf{B}_{x,y})_{i,j} = (-1)^{\langle x, i \rangle} [i = j + y], \quad (20)$$

with sums taken in $\mathbb{Z}_2^d$. So $\alpha_2((x, y), (x', y')) := (-1)^{\langle y, x' \rangle}$ is a normalized 2-cocycle on $G_0 := \mathbb{Z}_2^{2d}$, and $u_{x,y} \mapsto \mathbf{B}_{x,y}$ is an algebra isomorphism $\mathbb{F}_q^{\alpha_2}[G_0] \cong \mathcal{M}_n(\mathbb{F}_q)$. This is the d -fold tensor power of the $n = 2$ case of Proposition 2.19. Its commutation bicharacter $\rho_{\alpha_2}((x, y), (x', y')) = (-1)^{\langle y, x' \rangle + \langle x, y' \rangle}$ is nondegenerate, exactly as in Example 5.4. More generally, take $A = \prod_s \mathbb{Z}/n_s \mathbb{Z}$ with each $n_s \mid q - 1$. Then the cocycle $\alpha((x, y), (x', y')) = \prod_s \omega_{n_s}^{-y_s x'_s}$ realizes $\mathbb{F}_q^\alpha[A \times A] \cong \bigotimes_s \mathcal{M}_{n_s}(\mathbb{F}_q) \cong \mathcal{M}_n(\mathbb{F}_q)$. The Weyl algebra of Section 2.3.2 is the special case of a single cyclic factor, and the Pauli algebra is the case $n_s = 2$ for all s . If we extend the cocycle trivially along H , as in Section 2.3.3, we get $\mathcal{A} = \mathbb{F}_q^\alpha[A \times A \times H] \cong \mathcal{M}_n(\mathbb{F}_q)^N$ whenever $\exp(H) \mid q - 1$.

The construction, correctness, and security proofs of Section 4 only use the structure of the twisted basis, the sparsity bound $\text{wt}(xy) \leq \text{wt}(x)\text{wt}(y)$, and the isomorphism Ψ . So they apply without change to any A of this form, with the assumption of Section 2.3.4 stated over the matching algebra. Two things improve. First, every phase $\alpha(g, h)$ in the construction, and in particular the payload $r_i s_j \alpha(g_i, h_j)$ of Lemma 4.1, is now just a sign $(-1)^{\langle y_i, x'_j \rangle}$. This simplifies the distributed key generation of Section 6.3. Second, the conversion Ψ no longer needs any multiplication: by Equation (20), the entry $(i, i + y)$ of the slot $\chi \in \hat{H}$ of $\Psi(\sum_{x,y,h} c_{x,y,h} u_{x,y,h})$ equals $\sum_x (-1)^{\langle x, i \rangle} \hat{c}_{x,y}(\chi)$, where $\hat{c}_{x,y}(\chi) = \sum_h c_{x,y,h} \chi(h)$. So Ψ becomes n^2 Walsh–Hadamard transforms of size N , followed by Nn Walsh–Hadamard transforms of size n :

$$T_\Psi^\times(L) = 0, \quad T_\Psi^+(L) = L(\log_2 N + \log_2 n).$$

This does not change the sandwich products, which still dominate the cost of the twisted construction: the transform saving is only a few percent of its multiplications. So the main benefit of the Pauli grading for the twisted construction is not arithmetic. It is that the setup becomes Boolean-friendly for distributed generation, and it becomes consistent with the commutative construction.

Security of the elementary abelian instantiations. We now describe what changes for security, and point to Section 5 for the details.

- *Folding of the commutative instance.* Folding acts along subgroups of $A \times A \times H = \mathbb{Z}_2^{2d+e}$ (Section 5.1). The possible quotient sizes are $\{2^0, \dots, 2^{2d+e}\}$, exactly as for $\mathbb{Z}_n \times \mathbb{Z}_n \times \mathbb{Z}_N$ with $n = 2^d$, $N = 2^e$, and the folded weight distribution of Proposition 5.19 only depends on the size of the folded subgroup. So the fold-by-fold analysis of [BCCD23, BBC⁺24a], done there for $\mathbb{Z}_3^d$, applies here with the same conclusions at first order. What does change is the number of folds available at each size: $\mathbb{Z}_2^m$ has one subgroup per $\mathbb{F}_2$ -subspace, instead of one per divisor, and its automorphism group is $\text{GL}_m(\mathbb{F}_2)$ instead of a cyclic group. We are not aware of attacks that exploit this beyond the translation symmetry already taken into account by decoding-one-out-of-many [Sen11].
- *Folding of the twisted instance.* Proposition 5.12 and its proof only use that $\mathbb{F}_q^\alpha[G_0] \cong \mathcal{M}_n(\mathbb{F}_q)$, and that the equivalence relation $g \sim g'$ defined by $u_g p \in \mathbb{F}_q^\times u_{g'} p$ is translation-invariant. So they hold without change for the Pauli algebra. The rank-one folding-compatible idempotents of Section 5.5.2 become $p_\chi = \frac{1}{n} \sum_{y \in A} \chi(y)^{-1} u_{(0,y)}$ for characters $\chi \in \hat{A}$, and Proposition 5.19 does not change, since the block size n/r is the only parameter that matters. In other words, the folding analysis of the twisted part of the algebra does not depend at all on the group structure of A . Only the batch direction H is affected by the multiplicity remark above.
- *Interpolation and correlation attacks.* The attacks of Bouillaguet, Delaplace, Hamdad and Vergnaud [BDHV25], later refined by Joux [Jou26] using correlations, recover the sparse noise from evaluations of the syndrome at points where every public multiplier vanishes: if $a_i(P) = 0$ for all $i \geq 1$, then $y(P) = f(P)$. From many such values, [BDHV25] reconstructs f by sparse interpolation, and [Jou26] reconstructs it from the correlations between the values of f and the evaluations of monomials, all computed at once with a Walsh transform. Both attacks fail over a 64-bit field for the commutative construction. A

grid point isolates f with probability $q^{-(c-1)}$, so among the $D = 2^{30}$ points of the grid, the expected number of isolating points is $2^{30-64(c-1)} \ll 1$. The correlation bias scales as $(\sqrt{q}/(q-1))^t$ [Jou26, Thm. 1 and §10], so the number of samples needed grows as $((q-1)^2/q)^t$, which already exceeds the size of the grid for every $t \geq 1$ at $q \approx 2^{64}$. Joux states explicitly that the large-field instantiation of [LLX⁺26], which is the assumption used by our commutative construction, is not vulnerable to his attack. He leaves open whether the fact that the evaluation points have coordinates in a small multiplicative subgroup could be exploited some other way [Jou26, §10].

For the twisted construction, this attack has no starting point at all. The algebra $\mathcal{A} \cong \mathcal{M}_n(\mathbb{F}_q)^N$ admits no ring homomorphism to $\mathbb{F}_q$, since $\mathcal{M}_n(\mathbb{F}_q)$ is simple for $n \geq 2$: there are no scalar evaluations to exploit. Its only homomorphic images are the N slot maps to $\mathcal{M}_n(\mathbb{F}_q)$. The slot image $y_\chi = \sum_{i \geq 1} A_{i,\chi} E_{i,\chi} + F_\chi$ only reveals information about the noise if the uniform matrices $A_{i,\chi}$ share a left kernel vector, which happens with probability about $q^{-(c-1)}$ per slot: this happens for none of the $N = 2^{16}$ slots at 64 bits. Even when it does happen, what leaks is only a linear function of the Pauli-sparse matrix F_χ . This is a syndrome decoding instance, not an interpolation instance, and it is exactly the instance obtained by fully folding along H , which the folding estimator of Section 6.4 already accounts for.

We reach the same conclusion from the matrix side. Write $\mathcal{A} = \mathcal{M}_n(\mathbb{F}_q[H])$. Each entry of a weight- t element is a polynomial in the batch variables with about t/n terms, supported on at most t of the n diagonals. Column j of a sample is then a module-LPN sample over $\mathbb{F}_q[H]$ with n dense multipliers a_{ik} . An isolating evaluation at a character $\eta \in \hat{H}$ would require the matrix $(a_{ik}(\eta))_{i,k}$ to be singular, again with probability about $1/q$. So the twist removes the attack surface at a structural level, not just quantitatively. The one caveat over small fields, the case of singular slots, is quantified for $q = 16$ in the last paragraph of this section.

The small-field setting: $q = 16$ and ternary groups. When $q = 2^k$, the group $\mathbb{Z}_2^d$ is not available, since $2 \nmid q-1$. The natural choice is then 3. For $q = 16$, we have $q-1 = 15 = 3 \cdot 5$, so $\mathbb{F}_4 \subseteq \mathbb{F}_{16}$ contains a primitive cube root of unity ω . We take $A = \mathbb{Z}_3^4$, so $n = 81$, and $H = \mathbb{Z}_3^e$, so $N = 3^e$. This is the setting of threshold MAYO, whose matrices live over $\mathbb{F}_{16}$ and fit in 81×81 blocks.

For the commutative construction, the sample ring $\mathbb{F}_{16}[\mathbb{Z}_3^{8+e}]$ is the ring used by FOLEAGE [BBC⁺24a]. Its triadic transform computes the size-3 butterflies $(a, b, c) \mapsto (a+b+c, a+\omega b+\omega^2 c, a+\omega^2 b+\omega c)$. The multiplications by ω are fixed $\mathbb{F}_2$ -linear maps, so a transform of size M only costs $2M \log_3 M$ additions or ω -scalings, and no general multiplication: $T^\times(M) = 0$ and $T^+(M) = 2M \log_3 M$. On a vector unit, both operations are byte-table lookups.

For the twisted construction, $\alpha((x, y), (x', y')) = \omega^{-\langle y, x' \rangle}$, with the inner product taken modulo 3, realizes $\mathbb{F}_{16}^\alpha[\mathbb{Z}_3^8] \cong \mathcal{M}_3(\mathbb{F}_{16})^{\otimes 4} \cong \mathcal{M}_{81}(\mathbb{F}_{16})$. All the phases are cube roots of unity, and Ψ only consists of ternary transforms. Positions are shared additively over $\mathbb{Z}_3^m$ instead of XOR-shared, which the ternary distributed point functions of [BBC⁺24a] support directly (Section 6.3).

The biggest difference is early termination: a λ -bit leaf block holds $\mu = 27$ field elements of 4 bits when we stop the tree three levels early, against $\mu = 2$ over the 64-bit field. So the PRG cost of an expansion drops by another factor of 13. Field arithmetic is also much cheaper here: a multiplication by a fixed scalar of $\mathbb{F}_{16}$, applied to 16 elements at once, is a single table-lookup instruction. So the $n \times n$ products of the twisted construction cost only 0.1 ns per multiply-accumulate on one core (Table 3).

This gain comes at a cost, but only for the commutative construction. Over a small field, the isolating evaluation points needed by the interpolation and correlation attacks do exist: a grid point isolates the noise with probability $16^{-(c-1)}$, so $2^{24.5}$ of the $3^{18} \approx 2^{28.5}$ points do so at $c = 2$, and $2^{20.5}$ do so at $c = 3$. Section 6.4 works out the consequences with the estimator: at $c = 2$, a fraction 2^{-28} of the seeds can be broken in about 2^{31} operations. From $c = 3$ on, no attack of this family applies for any t , because the complete interpolation lines needed to split the noise polynomial no longer exist, and the noise itself has

far too many terms for a direct attack.

The twisted construction has no scalar evaluation points at any field size. Its only small-field weakness is the singular slot: over $\mathbb{F}_{16}$, about 6.7% of the N slots have a singular multiplier when $c = 2$, and each such slot leaks n linear equations about the Pauli-sparse image of the noise in that slot. For $c \geq 3$, two independent uniform matrices share a left kernel vector with probability about 16^{-n} , so nothing leaks. Recovering a t -sparse vector of length n^2 from n equations per leaking slot costs about $(15n)^t$ by information-set decoding, i.e. 2^{10t} at $n = 81$, far above 2^{128} at the weights of Section 6.4. The kernel vectors of each slot are also uniform, so they scramble the character structure that correlation attacks need. We therefore consider the twisted construction at $q = 16$ to be as secure as at 64 bits, under the same assumptions: $c = 2$ is available with this small residual leakage, and $c \geq 3$ has none.

6.3 Instantiating the sum-of-points FSS

Both constructions treat SPFSS as a black box. The proofs of Sections 3 and 4 only need correctness and simulation security for sums of t^2 point functions. Lemma 2.7 turns this into pseudorandom full-domain shares, and Remark 2.3 lets us replace the $I = 2c^2$ instances one at a time. Any scheme with these two properties can be used, and the choice of SPFSS instantiation decides most of the expansion cost. We consider three instantiations, from the least to the most efficient, and we write $\text{Dom} \in \{E, L\}$ for the FSS domain of the construction at hand.

(a) Sum of t^2 independent DPFs. This is the instantiation assumed in Section 4.3: one tree-based DPF [BGI16] per point, using a length-doubling PRG. A DPF over a domain of size M has a key of $\lceil \log_2 M \rceil (\lambda + 2) + \lambda + \lceil \log_2 q \rceil$ bits, and a full-domain evaluation costs about M PRG calls, one per internal node. *Early termination* [BBC⁺24a] stops the tree $\log_2 \mu$ levels early, where $\mu := \lfloor \lambda / \lceil \log_2 q \rceil \rfloor$ is the number of field elements that fit in one λ -bit leaf block, and replaces the last correction word by a $\mu \lceil \log_2 q \rceil$ -bit one. With early termination,

$$s_{\text{DPF}}(M) = \lambda + \left\lceil \log_2 \frac{M}{\mu} \right\rceil (\lambda + 2) + \mu \lceil \log_2 q \rceil, \quad C_{\text{DPF}}^{\text{FullEval}}(M) \approx \frac{M}{\mu} \text{ PRG calls},$$

and one SPFSS instance costs t^2 times this. Over the 64-bit field, $\mu = 2$, so early termination halves the number of PRG calls and shortens each key by 66 bits. Over $\mathbb{F}_{16}$ with ternary groups, the tree is ternary, as in FOLEAGE [BBC⁺24a]: one PRG call expands a node as $\{0, 1\}^\lambda \rightarrow \{0, 1\}^{3(\lambda+1)}$, i.e. three AES blocks, there are $(M/\mu - 1)/2$ internal nodes for M/μ leaves, and each level carries three correction words of $\lambda + 1$ bits. So

$$\begin{aligned} s_{\text{DPF}}^{(3)}(M) &= \lambda + \left\lceil \log_3 \frac{M}{\mu} \right\rceil \cdot 3(\lambda + 1) + \mu \lceil \log_2 q \rceil, \\ C_{\text{DPF},3}^{\text{FullEval}}(M) &\approx \frac{3}{4} \cdot \frac{M}{\mu} \text{ length-doubling equivalents}, \end{aligned} \tag{21}$$

with $\mu = 27$ when we stop the tree three levels early. The keys are about 1.5 times longer per level than binary ones, but this is dominated by the 27-fold gain from early termination. A further, standard saving needs *regular* noise: suppose each sparse element has exactly one nonzero coefficient in each of the t cosets of a fixed index- t subgroup of H (so $t \mid N$, and t is a power of two when $H = \mathbb{Z}_2^e$). Then the product of the k -th monomial of one element with the ℓ -th monomial of the other lies in the publicly known coset indexed by $k + \ell$, of size Dom/t . Each point function can then be shared by a DPF over that coset, and a full evaluation costs $t^2 \cdot \text{Dom}/(t\mu) = t \text{Dom}/\mu$ PRG calls, a factor of t less. This is how all implemented QA-SD and ring-LPN PCGs work [BCG⁺20b, BBC⁺24a, LLX⁺26]. The cost is that we need the regular-noise variant of the assumptions, which is the variant targeted by the algebraic attacks of [BØ23]. The constructions of Sections 3 and 4 are written for exact-weight uniform noise, and do not use this saving.

(b) Multi-point FSS with a dealer. A distributed multi-point function (DMPF) shares a sum of t^2 point functions directly. With cuckoo hashing [SGRR19, BGH⁺25], we place the t^2 points into m bins using κ public hash functions, so that each bin gets at most one point. Bin b then holds a DPF over the set of domain positions that hash to b , of size about $\kappa \text{Dom}/m$. We evaluate the function at x as the sum of the κ bin-DPFs at x , so a full-domain evaluation touches every position κ times:

$$s_{\text{DMPF}} \approx m \cdot s_{\text{DPF}} \left(\frac{\kappa \text{Dom}}{m} \right), \quad C_{\text{DMPF}}^{\text{FullEval}} \approx \frac{\kappa \text{Dom}}{\mu} \text{ PRG calls},$$

independently of t . Typical parameters are $\kappa = 3$ with $m \approx 1.3 t^2$, or $\kappa = 2$ with $m \approx 2.4 t^2$ [ARR25]. The dealer variant of FOLEAGE uses $\kappa = 2$ and $m = 2t^2$ [BBC⁺24a, App. D]. The hash functions must be chosen independently of the points, and a placement failure (probability $2^{-\lambda_{\text{stat}}}$, using a small stash or a slight over-provisioning) is handled as in [BGH⁺25]. The dealer knows all t^2 points and can hash them directly, so we do not need regularity here: we can keep the exact-weight uniform noise of our assumptions, and t is unconstrained.

(c) Fully distributed multi-point FSS. Without a dealer, neither party knows the product points on its own: a position is the sum $g_i + h_j$ of one position from each party, and a payload is the product $r_i s_j \alpha(g_i, h_j)$. So cuckoo insertion cannot be run directly on shared items. Agarwal, Raghuraman and Rindal [ARR25] solve this with *reverse* cuckoo hashing. The shared items are first obviously shuffled into $m = wd$ bins, with $d = 2^{\lceil \log_2(\#\text{points}) \rceil}$. The parties then solve, inside the protocol, w binary linear systems for hash functions $h_1, \dots, h_w$ that are consistent with this placement. The solutions are revealed, and this publicly fixes the set of positions in each bin. Each bin is then shared with a *sparse DPF*: a GGM tree in which every node with a single live child gets contracted. A bin over a set S then costs $|S| - 1$ PRG calls and one 2λ -bit correction word per level with a split. The recommended choice $w = 2$ gives $\sum_b |S_b| = 2 \text{Dom}$, so

$$C_{\text{rc}}^{\text{FullEval}} \approx \frac{2 \text{Dom}}{\mu} \text{ PRG calls}, \quad s_{\text{rc}} \approx m \cdot s_{\text{DPF}}(\text{Dom}),$$

Here the keys are those of full-depth trees, since the sparse DPF prunes computation, not correction words. The generation protocol reveals the correction words and leaves each party with a compact key, so we are in the usual setting of a distributed seed generation followed by local expansions. The protocol also supports a *stationary* mode, where the positions (and all position-dependent work) are reused across expansions with fresh payloads. This needs a stationary variant of the assumption [KPRR25], which we do not use here.

Two features of our constructions match this interface exactly. First, the protocol takes positions as XOR-shared bit strings. This is exactly what the parties hold for free when A and H are 2-groups: party 0 knows g_i , party 1 knows h_j , and $g_i + h_j$ is their XOR. This is the same observation that motivates the $\mathbb{F}_2^n$ domains of [LLX⁺26]. In the $q = 16$ setting, positions are shared over $\mathbb{Z}_3^m$ instead, and the ternary version of the sparse DPF applies, with the same bin count. Second, the protocol takes payloads as additive shares. The parties get these from one Gilboa product [Gil99] per point, i.e. $\lceil \log_2 q \rceil$ correlated OTs, plus, for the twisted construction, one 1-out-of- n OT in which party 0 offers the n signed copies $\{(-1)^{\langle y_i, x \rangle} r_i\}_{x \in \mathbb{Z}_2^d}$ and party 1 selects $x = x'_j$. With the Weyl cocycle, the sign becomes the root of unity $\omega^{-y_i x'_j}$, and nothing else changes.

The one place where regularity still matters is the cost of the generation step itself. The linear solver communicates $O(dq^2)$ bits per system with $q = d + \lambda_{\text{stat}}$, and deduplication performs $O(\#\text{points}^2)$ secret equality tests. So a single instance with t^2 points is out of reach for $t \geq 32$. Following [ARR25, §10.1], regular noise instead splits the t^2 products of one SPFSS instance into t instances of t points each, one per coset of size Dom/t , at the same expansion cost. So regularity is no longer needed for expansion. It only helps at generation time, and only in the dealer-free setting.

Consequences. With a multi-point instantiation, the number of PRG calls for one expansion is $I\kappa \text{Dom}/\mu$, independent of t . The noise weight t now only affects the seed size, which grows as $I \cdot m \cdot s_{\text{DPF}} \approx 2c^2 \cdot 2t^2 \cdot s_{\text{DPF}}$ bits, and the one-time generation cost. Every remaining term in the expansion cost scales as c^2 (the transforms, the pointwise products, and, for the twisted construction, the $2c^2 - 1$ dense products). So the trade-off flips: we should now minimize c and let t carry the security requirement, i.e. prefer rate-1/2 codes. For security, the proofs of Sections 3 and 4 go through unchanged. Finally, the twisted construction benefits the most from this: the multi-point instantiation brings its PRG cost down to a few percent of the total, and its expansion is then dominated by the $2c^2 - 1$ honest matrix products, which are plain arithmetic on operands that each party already knows.

6.4 Parameter selection

The estimator. We ported the public Sage estimator that comes with FOLEAGE [BBC⁺24a] to a standalone Python script, with three changes. First, we do all arithmetic in the logarithmic domain. This is what makes 64-bit fields and groups of order 2^{30} tractable, and it brings the running time down from minutes to a few seconds per instance. Second, we use the group $\mathbb{Z}_d^s$ with $d \mid q - 1$, instead of $(\mathbb{Z}/(q - 1)\mathbb{Z})^s$. The case $d = 2$ is the change used in [LLX⁺26] for multiquadratic QA-SD over large prime fields. Third, we can impose a minimum fold size, which is how we handle the twisted variant. Otherwise, the model is the one of [BBC⁺24a, §6]. The attacker folds the module-LPN instance along a subgroup of order l , obtaining a $[cn_f, (c - 1)n_f]_q$ code with $n_f = |G|/l$, and an error whose weight follows the c -fold convolution of the distribution of Proposition 5.19 with block size l . The attacker runs whichever of Prange, Lee–Brickell, Stern, Peters’ Stern, and MMT is cheapest over $\mathbb{F}_q$, together with the $\sqrt{n_f}$ gain from decoding-one-out-of-many [Sen11], and may retry with fresh subgroups until the folded weight is unusually small, at a cost of $(2^{C_w} + c|G|)/\Pr[w]$. We explore fold sizes around the one closest to the Gilbert–Varshamov bound, as well as the unfolded instance. Our port reproduces the two worked examples in the documentation of the estimator to the decimal (156.1 and 128.8 bits), reproduces the re-evaluation of the parameters of [BCCD23] in [BBC⁺24a, Table 2] within three bits, and reproduces the aggressive column of [LLX⁺26, Table 3] within one unit of t .

The twisted variant in the estimator. Generic decoding does not see the twist. In the twisted basis, left multiplication by a has the same group-circulant parity-check structure as QA-SD, with every entry scaled by a public root of unity, so ISD algorithms run at the same parameters. Right multiplication by the $|G|$ basis elements permutes the instances the same way translations do in the commutative case, so the decoding-one-out-of-many gain is also the same. What changes is the family of folds available. Along H , the folds are the usual ones. Along the twisted part, Proposition 5.12 shows that a folding-compatible idempotent of rank r folds to dimension $nr \geq n$, with blocks of size n/r , and the folded weight distribution of Proposition 5.19 only depends on the block size. A product of a rank- r idempotent with an H -fold has this same product block structure. So we analyze the twisted instance with the same machinery, over fold sizes $n_f = nrN'$ with $r \mid n$ and $N' \mid N$: this gives a floor $n_f \geq n$, and otherwise the same set of sizes as the commutative instance.

Results at $q = 2^{64} - 2^{32} + 1$. Table 1 gives the smallest per-block weight that reaches 128 bits for $c \in \{2, 3, 4\}$, together with the estimated security of the working points used in Section 6.5. This covers the commutative assumption over $\mathbb{F}_q[\mathbb{Z}_2^{30}]$ (the sample ring $\mathbb{F}_q[A \times A \times H]$ at $n = 128$, $N = 2^{16}$), and the twisted assumption over $\mathbb{F}_q^\alpha[\mathbb{Z}_2^{14} \times \mathbb{Z}_2^{16}]$. Three observations. First, the minimal weights do not depend on the group order: $s = 24$ gives the same values as $s = 30$, matching [LLX⁺26]. Second, the floor imposed by the twisted variant is almost invisible: for these weights, the best fold of the commutative instance already has size 2^5 to 2^6 , just below $n = 2^7$, so the twisted instance is only one or two bits harder. Third, the estimator reproduces the aggressive weights of [LLX⁺26] (57, 24, 15), which were

Table 1: Output of our port of the FOLEAGE estimator ($q = 2^{64} - 2^{32} + 1$, exact-weight uniform noise, 128-bit target). Top: smallest per-block weight t and the fold size at which the best attack operates. Bottom: estimated security of the working points of Section 6.5.

	commutative, $\mathbb{F}_q[\mathbb{Z}_2^{30}]$		twisted, $A = \mathbb{Z}_2^7, H = \mathbb{Z}_2^{16}$	
	$t_{\min}$	best fold	$t_{\min}$	best fold
$c = 2$	58	2^6	57	2^7
$c = 3$	24	2^6	24	2^7
$c = 4$	15	2^5	15	2^7

(c, t)	commutative	twisted
(2, 64)	143	143
(3, 32)	166	168
(4, 16)	143	143
(2, 128)	272	272
(3, 64)	320	320
(4, 64)	529	529

obtained with the same tool. Their conservative weights (67, 42, 33) come from the linear-test framework of [BCG⁺20a], which we do not model here. Since regular noise over $\mathbb{Z}_2^e$ needs t to be a power of two, we keep two families of working points: the *estimator-based* points (2, 64), (3, 32), (4, 16), each giving at least 143 bits under this model, and the *conservative* points (2, 128), (3, 64), (4, 64), obtained by rounding the linear-test weights up, each giving above 270 bits under this model.

Results at $q = 16$. Table 2 gives the same quantities for $A = \mathbb{Z}_3^4, H = \mathbb{Z}_3^{10}$, i.e. the commutative assumption over $\mathbb{F}_{16}[\mathbb{Z}_3^{18}]$ and the twisted assumption over $\mathbb{F}_{16}^\alpha[\mathbb{Z}_3^8 \times \mathbb{Z}_3^{10}]$ with fold floor $n = 81$. It also gives the output of the isolating-point module for the commutative variant. Folding and generic decoding give essentially the same weights as at 64 bits (slightly higher at $c = 2$, since two errors in a block can now cancel with probability $1/15$), so they do not distinguish the two fields. What does distinguish them is the isolating-point module. At $c = 2$, fixing one variable leaves $2^{14.9}$ complete interpolation lines, on which the correlation attack recovers sub-polynomials of at most three terms. One of the three sub-polynomials of a 70- or 81-term noise polynomial has at most three terms with probability 2^{-22} to 2^{-28} , after which the attack costs about 2^{31} operations. So the commutative construction at $c = 2$ has a 2^{-28} fraction of weak seeds, and we exclude this case. From $c = 3$ on, there are at most a handful of complete lines, so the whole noise polynomial would have to be recovered at once. The bias $(\sqrt{q}/(q-1))^t$ rules this out for $t \geq 5$, and information-set decoding on the isolating code costs $16^{(c-1)t}$. The module reports no attack for any t , and the criterion of [LLX⁺26, App. D] for the interpolation attack is only met from $c = 4$ on. Since regular noise over $\mathbb{Z}_3^e$ needs t to be a power of three, we use the working points (4, 27) for both constructions, (3, 81) as a conservative point, and (2, 81) for the twisted construction only, where $c = 2$ is allowed because the twisted algebra has no isolating points.

6.5 Concrete estimates

Per-operation constants. Table 3 lists the constants we use to turn these counts into time. We measured them with C micro-benchmarks on an Apple M1 Pro (eight cores): fixed-key AES-128 with the ARMv8 cryptographic extensions, in-place Walsh–Hadamard sweeps over arrays of 2^{16} to 2^{24} elements modulo q , and batches of 256 to 1024 independent 128×128 products modulo q (i.e. the sandwich products of the twisted construction), with lazy 192-bit accumulation over the inner index and a single reduction per

Table 2: Output of our port of the FOLEAGE estimator at $q = 16$, $A = \mathbb{Z}_3^4$ ($n = 81$), $H = \mathbb{Z}_3^{10}$, exact-weight uniform noise, 128-bit target. The top table gives the smallest per-block weight from folding and generic decoding, identical for the two variants, and the verdict of the isolating-point module on the commutative variant at that weight, as the $\log_2$ of the expected cost divided by the probability of a weak seed (“ ∞ ”: no attack for any t). The bottom table gives the folding security of the ternary working points.

	$t_{\min}$	best fold	isolating-point attacks (comm.)
$c = 2$	70	3^4	53 bits (2^{-22} weak seeds)
$c = 3$	28	3^4	∞
$c = 4$	17	3^4	∞
$c = 5$	12	3^4	∞
$c = 6$	9	3^4	∞

(c, t)	commutative	twisted
$(2, 81)$	excluded (2^{-28} weak seeds)	161
$(4, 27)$	201	201
$(3, 81)$	362	362

Table 3: Per-operation constants, measured with C micro-benchmarks on an Apple M1 Pro.

operation	time	note
fixed-key AES-128 block	1.70 ns	8 independent streams
length-doubling PRG call	3.4 ns	two AES blocks
add/sub mod q in a Walsh–Hadamard sweep	0.36 ns	per element and level, 0.72 ns per butterfly, memory-bound above 2^{20}
multiply-accumulate mod q	1.01 ns	batched 128×128 products, lazy 192-bit accumulation, 2.2 ns with per-term reduction
one 128×128 product mod q	2.1 ms	schoolbook, one core, 0.4 ms amortized on eight cores
multiply-accumulate in $\mathbb{F}_{16}$	0.105 ns	batched 81×81 products, NEON table lookups, 0.37 ns scalar
one 81×81 product over $\mathbb{F}_{16}$	0.056 ms	one core, 0.009 ms amortized on eight cores
add or ω -scaling in a radix-3 sweep	0.05 ns	per operation, 0.10 ns per element and level, arrays up to 3^{16}

output entry. The 64-bit product kernel is bound by the scalar $64 \times 64 \rightarrow 128$ -bit multiplier, since NEON has no vector multiply of that width. On the eight cores of the machine, the batched products reach 0.19 ns per multiply-accumulate in aggregate. Over $\mathbb{F}_{16}$, one element occupies a byte, a multiplication by a fixed scalar is a 16-entry table lookup applied to 16 elements at once, and the same instruction implements the ω -scalings of the radix-3 butterflies. Batches of 81×81 products run at 0.105 ns per multiply-accumulate on one core, and 0.0175 ns in aggregate. A radix-3 transform costs 0.10 ns per element and level, for arrays up to 41 MiB. These constants are only indicative, and hardware accelerators would change the picture for the products. As a cross-check, [LLX⁺26] measure a 64-bit Walsh–Hadamard transform of size 2^{20} in 13 ms on an EPYC 9754 core, i.e. 0.6 ns per addition, and FOLEAGE [BBC⁺24a] reports that AES takes up two thirds of the time of a full-domain DPF evaluation.

Fully optimized configurations. Table 4 gives, for one matrix triple, the seed size, the operation counts, the resident memory of the natural implementation, and the projected single-core time, for the four working points and three batch sizes. The per-triple expansion cost is independent of N up to logarithmic

Table 4: Fully optimized configurations ($A = \mathbb{Z}_2^7$, $H = \mathbb{Z}_2^e$, distributed reverse-cuckoo multi-point FSS with regular noise, early termination, Strassen), $n = 128$, $\lambda = 128$, per party. “Memory” is the resident working set of the natural implementation. “core-ms/triple” uses the constants of Table 3.

Commutative construction (Section 3), $q = 2^{64} - 2^{32} + 1$, $A = \mathbb{Z}_2^7$, $H = \mathbb{Z}_2^e$							
(c, t)	N	seed/triple	PRG/triple	mult./triple	add./triple	memory	core-ms/triple
$(2, 64)$	2^{12}	7273 B	$1.7 \cdot 10^7$	$3.5 \cdot 10^7$	$5.8 \cdot 10^8$	128.0 GiB	299
$(2, 64)$	2^{16}	520 B	$1.7 \cdot 10^7$	$3.5 \cdot 10^7$	$6.4 \cdot 10^8$	2048.0 GiB	323
$(3, 32)$	2^{12}	4237 B	$3.8 \cdot 10^7$	$7.7 \cdot 10^7$	$1.3 \cdot 10^9$	128.0 GiB	670
$(3, 32)$	2^{16}	301 B	$3.8 \cdot 10^7$	$7.7 \cdot 10^7$	$1.4 \cdot 10^9$	2048.0 GiB	724
$(4, 16)$	2^{12}	1948 B	$6.7 \cdot 10^7$	$1.4 \cdot 10^8$	$2.3 \cdot 10^9$	128.0 GiB	1189
$(4, 16)$	2^{16}	138 B	$6.7 \cdot 10^7$	$1.4 \cdot 10^8$	$2.6 \cdot 10^9$	2048.0 GiB	1286
$(2, 128)$	2^{12}	28049 B	$1.7 \cdot 10^7$	$3.5 \cdot 10^7$	$5.8 \cdot 10^8$	128.0 GiB	299
$(2, 128)$	2^{16}	2013 B	$1.7 \cdot 10^7$	$3.5 \cdot 10^7$	$6.4 \cdot 10^8$	2048.0 GiB	323
$(3, 64)$	2^{12}	16363 B	$3.8 \cdot 10^7$	$7.7 \cdot 10^7$	$1.3 \cdot 10^9$	128.0 GiB	670
$(3, 64)$	2^{16}	1169 B	$3.8 \cdot 10^7$	$7.7 \cdot 10^7$	$1.4 \cdot 10^9$	2048.0 GiB	724
$(4, 64)$	2^{12}	29089 B	$6.7 \cdot 10^7$	$1.4 \cdot 10^8$	$2.3 \cdot 10^9$	128.0 GiB	1189
$(4, 64)$	2^{16}	2078 B	$6.7 \cdot 10^7$	$1.4 \cdot 10^8$	$2.6 \cdot 10^9$	2048.0 GiB	1286
Twisted construction (Section 4), $q = 2^{64} - 2^{32} + 1$, $A = \mathbb{Z}_2^7$, $H = \mathbb{Z}_2^e$							
(c, t)	N	seed/triple	PRG/triple	mult./triple	add./triple	memory	core-ms/triple
$(2, 64)$	2^{12}	5453 B	$1.3 \cdot 10^5$	$1.1 \cdot 10^7$	$8.4 \cdot 10^6$	2.0 GiB	14
$(2, 64)$	2^{16}	406 B	$1.3 \cdot 10^5$	$1.1 \cdot 10^7$	$8.9 \cdot 10^6$	32.0 GiB	14
$(3, 32)$	2^{12}	3214 B	$2.9 \cdot 10^5$	$2.3 \cdot 10^7$	$2.0 \cdot 10^7$	2.0 GiB	31
$(3, 32)$	2^{16}	237 B	$2.9 \cdot 10^5$	$2.3 \cdot 10^7$	$2.1 \cdot 10^7$	32.0 GiB	32
$(4, 16)$	2^{12}	1493 B	$5.2 \cdot 10^5$	$4.0 \cdot 10^7$	$3.6 \cdot 10^7$	2.0 GiB	55
$(4, 16)$	2^{16}	110 B	$5.2 \cdot 10^5$	$4.0 \cdot 10^7$	$3.8 \cdot 10^7$	32.0 GiB	56
$(2, 128)$	2^{12}	20769 B	$1.3 \cdot 10^5$	$1.3 \cdot 10^7$	$8.4 \cdot 10^6$	2.0 GiB	16
$(2, 128)$	2^{16}	1558 B	$1.3 \cdot 10^5$	$1.3 \cdot 10^7$	$8.9 \cdot 10^6$	32.0 GiB	17
$(3, 64)$	2^{12}	12268 B	$2.9 \cdot 10^5$	$2.5 \cdot 10^7$	$2.0 \cdot 10^7$	2.0 GiB	34
$(3, 64)$	2^{16}	913 B	$2.9 \cdot 10^5$	$2.5 \cdot 10^7$	$2.1 \cdot 10^7$	32.0 GiB	34
$(4, 64)$	2^{12}	21809 B	$5.2 \cdot 10^5$	$4.4 \cdot 10^7$	$3.6 \cdot 10^7$	2.0 GiB	60
$(4, 64)$	2^{16}	1623 B	$5.2 \cdot 10^5$	$4.4 \cdot 10^7$	$3.8 \cdot 10^7$	32.0 GiB	60

terms. The batch size only trades seed amortization against memory. For the twisted construction, one 128×128 triple costs 14 to 60 core-milliseconds, depending on c , against 2.1 ms to compute the same product in the clear on the same core. So the silent tax is a factor of 7 to 28, almost entirely due to the $2c^2 - 1$ honest products, and the rate-1/2 points $(2, 64)$ and $(2, 128)$ are the cheapest, by a factor of 2 to 4. For the commutative construction, the same triple costs 0.3 to 1.3 core-seconds, about 20 times more than the twisted construction, and it has one extra constraint: its working set is $E = n^3 N$ elements, already 32 GiB at $N = 2^{10}$. So large batches need an out-of-core implementation that streams along the batch coordinate. Per scalar operation, the twisted construction at $(2, 64)$ spends about 7 ns per scalar multiplication of the underlying matrix products. For comparison, the fastest reported scalar OLE PCG expansion over the same field, the reverse-cuckoo ring-LPN pipeline of [ARR25], produces one OLE every 0.55 μ s on a laptop core, and a matrix product built from scalar OLEs would need n^3 of them.

The $q = 16$ setting. Table 5 repeats this for $q = 16$, $n = 81$, with the ternary DPFs of Equation (21), $\mu = 27$, the distributed multi-point FSS, and the constants of Table 3. The twisted construction produces an 81×81 triple over $\mathbb{F}_{16}$ in 0.6 core-milliseconds at $(2, 81)$, and 2.2 at $(4, 27)$, against 0.056 ms for

Table 5: Fully optimized configurations at $q = 16$ ($A = \mathbb{Z}_3^4$, $H = \mathbb{Z}_3^e$, ternary DPFs with $\mu = 27$, distributed reverse-cuckoo multi-point FSS with regular noise), $n = 81$, $\lambda = 128$, per party. Same conventions as Table 4.

Commutative construction (Section 3), $q = 16$, $A = \mathbb{Z}_3^4$, $H = \mathbb{Z}_3^e$							
(c, t)	N	seed/triple	PRG/triple	mult./triple	add./triple	memory	core-ms/triple
(4, 27)	3^8	5957 B	$9.4 \cdot 10^5$	$3.5 \cdot 10^7$	$7.0 \cdot 10^8$	3.2 GiB	42
(4, 27)	3^{10}	752 B	$9.4 \cdot 10^5$	$3.5 \cdot 10^7$	$7.7 \cdot 10^8$	29.2 GiB	45
(3, 81)	3^8	37455 B	$5.3 \cdot 10^5$	$2.0 \cdot 10^7$	$3.9 \cdot 10^8$	3.2 GiB	24
(3, 81)	3^{10}	4773 B	$5.3 \cdot 10^5$	$2.0 \cdot 10^7$	$4.3 \cdot 10^8$	29.2 GiB	25
Twisted construction (Section 4), $q = 16$, $A = \mathbb{Z}_3^4$, $H = \mathbb{Z}_3^e$							
(c, t)	N	seed/triple	PRG/triple	mult./triple	add./triple	memory	core-ms/triple
(2, 81)	3^8	11754 B	$2.9 \cdot 10^3$	$4.8 \cdot 10^6$	$1.3 \cdot 10^6$	82.1 MiB	0.6
(2, 81)	3^{10}	1578 B	$2.9 \cdot 10^3$	$4.8 \cdot 10^6$	$1.5 \cdot 10^6$	738.9 MiB	0.6
(4, 27)	3^8	4326 B	$1.2 \cdot 10^4$	$1.8 \cdot 10^7$	$5.2 \cdot 10^6$	82.1 MiB	2.1
(4, 27)	3^{10}	571 B	$1.2 \cdot 10^4$	$1.8 \cdot 10^7$	$6.1 \cdot 10^6$	738.9 MiB	2.2
(3, 81)	3^8	26447 B	$6.6 \cdot 10^3$	$1.1 \cdot 10^7$	$3.0 \cdot 10^6$	82.1 MiB	1.3
(3, 81)	3^{10}	3550 B	$6.6 \cdot 10^3$	$1.1 \cdot 10^7$	$3.4 \cdot 10^6$	738.9 MiB	1.4

Table 6: Share of the projected expansion time at $q = 16$, $N = 3^{10}$. “–” means the working point is not used for that construction.

(c, t)	Commutative			Twisted		
	PRG	mult.	add.	PRG	mult.	add.
(2, 81)	–	–	–	2%	85%	13%
(3, 81)	7%	8%	85%	2%	86%	13%
(4, 27)	7%	8%	85%	2%	84%	14%

the same product in the clear: a silent tax of 10 to 40. The commutative construction needs 25 to 45 core-milliseconds. Early termination and the multi-point FSS bring the PRG down to about 2% of the twisted expansion time and 7% of the commutative one (Table 6), so both constructions are now bound by arithmetic: the twisted construction by its $2c^2 - 1$ products, and the commutative one by the $2c^2 E \log_3 E$ operations of its transforms, over a working set of $E = n^3 N$ elements (29 GiB at $N = 3^{10}$, against 0.7 GiB for the twisted construction). Seeds are 570 to 3,600 bytes per triple at $N = 3^{10}$, essentially the same as at 64 bits, since the bin keys barely depend on the field. The expanded output, on the other hand, shrinks to $3 \cdot 81^2 \cdot 4$ bits, i.e. 9.6 KiB per triple and per party, so the expansion factor is only 3 to 17 here. This means a real deployment should optimize the seed, not the expansion.

Distributed generation. Table 7 estimates the one-time communication of a dealer-free generation in the two settings, based on the component costs given in [ARR25]. For each of the $I \cdot t$ coset instances of t points, this counts two binary solves over $d \times (d + 40)$ matrices with $d = t$, one 2λ -bit correction word and two string OTs per level and bin, one field element per bin, and t^2 secret equality tests for deduplication (charged at four bits per AND gate), plus the payload phase of Section 6.3. The estimate uses unit constants for the solver, and should only be read as an order of magnitude. The only measured anchor point is the 150 to 170 MB of one-time communication and 12.5 s of single-core time reported in [ARR25] for a ring-LPN OLE instance of size 2^{20} with four cross terms. Computationally, the generation step performs

Table 7: Estimated one-time communication of the distributed generation (both parties), by the formulas of Section 6.5. This is only an order of magnitude, see text. Left: $q = 2^{64} - 2^{32} + 1$. Right: $q = 16$.

(c, t)	N	Commutative		Twisted	
		FSS generation	payload phase	FSS generation	payload phase
(2, 64)	2^{12}	242.4 MiB	16.5 MiB	200.3 MiB	49.5 MiB
(2, 64)	2^{16}	266.5 MiB	16.5 MiB	224.4 MiB	49.5 MiB
(2, 128)	2^{12}	1.6 GiB	66.0 MiB	1.4 GiB	198.0 MiB
(2, 128)	2^{16}	1.7 GiB	66.0 MiB	1.5 GiB	198.0 MiB
(3, 32)	2^{12}	107.1 MiB	9.3 MiB	85.4 MiB	27.8 MiB
(3, 32)	2^{16}	122.3 MiB	9.3 MiB	97.8 MiB	27.8 MiB
(3, 64)	2^{12}	545.4 MiB	37.1 MiB	450.7 MiB	111.4 MiB
(3, 64)	2^{16}	599.6 MiB	37.1 MiB	504.8 MiB	111.4 MiB
(4, 16)	2^{12}	43.2 MiB	4.1 MiB	34.0 MiB	12.4 MiB
(4, 16)	2^{16}	48.4 MiB	4.1 MiB	39.2 MiB	12.4 MiB
(4, 64)	2^{12}	969.6 MiB	66.0 MiB	801.2 MiB	198.0 MiB
(4, 64)	2^{16}	1.0 GiB	66.0 MiB	897.4 MiB	198.0 MiB

(c, t)	N	Commutative		Twisted	
		FSS generation	payload phase	FSS generation	payload phase
(2, 81)	3^8	–	–	758.8 MiB	3.8 MiB
(2, 81)	3^{10}	–	–	800.4 MiB	3.8 MiB
(3, 81)	3^8	1.9 GiB	346.0 KiB	1.7 GiB	8.5 MiB
(3, 81)	3^{10}	1.9 GiB	346.0 KiB	1.8 GiB	8.5 MiB
(4, 27)	3^8	125.3 MiB	68.3 KiB	99.8 MiB	1.7 MiB
(4, 27)	3^{10}	138.3 MiB	68.3 KiB	112.5 MiB	1.7 MiB

the same tree evaluations as one expansion, using $\log_2(\text{Dom}/(t\mu))$ sequential OT rounds plus the solver rounds. So its cost is about that of one expansion, plus a few seconds of OT-based two-party computation. This cost is incurred once per seed, i.e. once every N triples.

7 Applications

Modern machine-learning workloads are built around large linear-algebraic computations, and much of this work becomes matrix multiplication once the model is evaluated securely. Dense layers multiply batches of activations by weight matrices, training performs further matrix products in the backward pass and in gradient computation, convolutions can be written as matrix products after rearranging the input, and transformer inference repeatedly computes linear projections as well as the products $QK^\top$ and PV in attention. It is therefore unsurprising that matrix multiplication accounts for a substantial part of the cost of privacy-preserving machine learning [MZ17, CKR⁺20, jLHG⁺25].

In an arithmetic MPC protocol, these products can always be reduced to scalar multiplications. For $X \in \mathbb{F}_q^{m \times k}$ and $Y \in \mathbb{F}_q^{k \times r}$, the usual algorithm computes XY from mkr scalar products, and hence can be implemented using the corresponding number of scalar Beaver triples. Matrix triples preprocess the same computation at the level at which it appears in the application. The parties receive shares of matrices $A, B, C = AB$, and use them to evaluate the entire matrix product in the online phase. Existing secure-ML protocols already use such correlations explicitly [MZ17, CKR⁺20, JGB⁺24]; others realize the same matrix multiplications through scalar preprocessing or homomorphic encryption. Thus an efficient PCG for matrix triples applies to a primitive that existing protocols already spend substantial resources generating or emulating.

Also, statistical protocols repeatedly multiply large data matrices by vectors, and in many cases these products can be batched into matrix-matrix multiplication. Matrix multiplication also appears inside cryptographic protocols themselves. Take threshold MAYO for example. Its distributed signing computation contains structured matrix products over a small field. We consider these settings because they arise for different reasons and at rather different dimensions. Together they give a useful picture of where matrix triples are effective and, in particular, of why matrix dimensions such as 64 and 128 are relevant concrete parameters for our constructions.

Our PCGs generate triples for square $n \times n$ matrices, while the matrices in these applications are usually rectangular. We handle this by a simple block decomposition. Partition the dimensions m, k, r into intervals of length at most n . Each pair of blocks is multiplied using an $n \times n$ triple, after completing

only the incomplete boundary blocks with zeros. The number of square products is therefore

$$\left\lceil \frac{m}{n} \right\rceil \left\lceil \frac{k}{n} \right\rceil \left\lceil \frac{r}{n} \right\rceil.$$

Figure 4 shows this decomposition. No padding is needed when the three dimensions are multiples of n , and otherwise the overhead is confined to the boundary blocks.

7.1 Privacy-Preserving Machine Learning

Privacy-preserving machine learning is an immediate application of matrix triples. Existing protocols devote substantial resources to secure matrix multiplication, either by generating the required correlations during preprocessing or by carrying out the products online using homomorphic encryption or related techniques.

SecureML [MZ17], for example, extends Beaver multiplication to shared matrices and uses matrix triples in its training protocols. In its neural-network experiments, the offline phase is the main bottleneck. Chen et al. [CKR⁺20] generate authenticated matrix and convolution triples for maliciously secure inference. Their ResNet-50 convolutions are evaluated using 3298 multiplications of pairs of 128×128 matrices. BumbleBee [LHG⁺25] takes a different approach and performs the matrix multiplications in transformer inference online using homomorphic encryption; it reports that matrix multiplication accounts for 70–95% of the inference time. In each case, secure matrix multiplication is a substantial part of the cost.

These workloads also give matrix dimensions that are directly relevant to our parameters. The ResNet-50 blocks of Chen et al. are an exact fit for $n = 128$: each block product consumes one 128×128 triple and requires no padding.

SecureML trains, among other models, a fully connected network for MNIST with input dimension 784, two hidden layers of width 128, and an output layer of width 10, using batches of 128 examples. The first dense layer on one batch therefore requires the product $(128 \times 784)(784 \times 128)$. With $n = 128$, we split the contracted dimension into seven blocks, so the product uses seven 128×128 triples. Only the final block is incomplete: padding increases the contracted dimension from 784 to 896, giving a padding factor $896/784 \approx 1.14$, or about 14% overhead.

Transformer attention gives a less well-aligned example. GPT-2 base has hidden dimension 768 and 12 attention heads, so each head has dimension 64. With 128 input tokens, the product $QK^\top$ for one head has shape $(128 \times 64)(64 \times 128)$, while PV has shape $(128 \times 128)(128 \times 64)$. For $n = 128$, each product uses one square triple, but one dimension is padded by a factor of two. Taking $n = 64$ removes the padding, but each product then requires four smaller triples. Thus neither choice is uniformly better: the appropriate matrix dimension depends on the shapes occurring in the application.

The ResNet-50 workload also permits a direct comparison with existing matrix-triple generation. Chen et al. [CKR⁺20] report 36.1 seconds of single-thread preprocessing and 12.46 MB of communication per party for one authenticated 128×128 triple. For the 3298 block products used by the ResNet-50 convolutions, they report about 41 GB of preprocessing communication per party and 7.4 hours of preprocessing on 16 threads. Subsequent work improves the homomorphic-encryption component of this approach. Mono and Güneysu [MG23] optimize the underlying homomorphic matrix multiplication, while MatriGear [CHM⁺25] reports about 0.99 seconds of single-thread computation for the HE component of a 128×128 authenticated triple.

For the parameter choice $(c, t) = (2, 64)$ and $N = 2^{16}$ from Table 4, our optimized twisted construction requires 406 bytes of amortized seed per party for each 128×128 triple, with an estimated single-core expansion time of 14.5 ms. Generating the 3298 triples needed for the ResNet-50 block products therefore requires about 1.3 MiB of amortized seed per party and 48 seconds of single-core expansion.

These numbers should not be read as an end-to-end speedup over Chen et al. or MatriGear. Their protocols generate authenticated triples with malicious security over a roughly 128-bit prime field, whereas the concrete figures in Table 4 are for semi-honest, unauthenticated triples over the 64-bit prime $2^{64} - 2^{32} + 1$. Distributed generation also incurs the additional amortized setup cost reported in Table 7. The narrower comparison is nevertheless useful: the 128×128 dimension used in an existing secure-inference workload is exactly one of the concrete regimes for which our PCG performs well.

The arithmetic domain also matters. SecureML, Orca, and BumbleBee, for example, work over $\mathbb{Z}_{2^{64}}$, whereas our present PCGs generate matrix triples over finite fields. The prime-field setting of Chen et al. therefore fits directly. Supporting protocols that work natively over $\mathbb{Z}_{2^{64}}$ would require extending our construction to this ring setting, which we expect to follow naturally by porting the techniques of [LXY25b] to our setting. We leave this extension to future work.

7.2 Batched and Iterative Matrix Computations

A single matrix–vector multiplication makes poor use of a square matrix triple. If

$$X \in \mathbb{F}_q^{m \times k} \quad \text{and} \quad v \in \mathbb{F}_q^k,$$

then embedding v into an $n \times n$ matrix is naturally wasteful. The situation is better when the same matrix is multiplied by many different vectors, as happens in iterative optimization and statistical estimation.

Suppose that a protocol repeatedly computes

$$Xv_1, Xv_2, \dots, Xv_s.$$

For $s \leq n$, we can regard the vectors as the columns of a matrix

$$V = \begin{pmatrix} v_1 & v_2 & \cdots & v_s \end{pmatrix}.$$

Then

$$XV = \begin{pmatrix} Xv_1 & Xv_2 & \cdots & Xv_s \end{pmatrix}.$$

The products need not be evaluated at the same time. The columns of the right-hand side of a matrix triple can be used over successive iterations, while the same mask for X is reused.

A concrete example would be secure Poisson regression where Kelkar et al. [KLS22] consider a two-party protocol in which a data matrix with N_{samp} rows and d features is reused throughout gradient descent. Each iteration contains one multiplication by X and one by $X^\top$. Their protocol uses correlated Beaver triples that reuse the mask of X across these products, and they report that matrix multiplication accounts for more than 90% of both computation and communication in their implementation.

For their benchmark with $N_{\text{samp}} = 10,000$ and $d = 100$, using the columns of a matrix triple across 128 iterations gives the two products $(10,000 \times 100)(100 \times 128)$ and $(100 \times 10,000)(10,000 \times 128)$. With $n = 128$, each of these requires

$$\left\lceil \frac{10,000}{128} \right\rceil = 79$$

square triples. Thus 128 iterations require 158 triples in total, or about 1.23 triples per iteration. The padding overhead is about 29%.

For the parameter choice $(c, t) = (2, 64)$ and $N = 2^{16}$ used above, this corresponds to about 17.9 ms of single-core expansion and 501 bytes of amortized seed per party per iteration. Kelkar et al. report, for the same matrix dimensions, 0.57 seconds and 1.57 MB of communication per iteration for their correlated-triple generation. Their complete offline phase takes 3.116 seconds and 13.5 MB per iteration.

As before, these figures are not directly comparable. Kelkar et al. work over $\mathbb{Z}_{2^{127}}$, whereas our concrete estimates use the 64-bit prime field from Section 6. The use of this example is that it shows that matrix triples can also be effective for computations that are written naturally as a sequence of matrix–vector products, provided that the same matrix is reused.

7.3 Threshold MAYO

Matrix multiplication also appears in threshold implementations of post-quantum signatures. Threshold MAYO [CEN25, ABCN26] distributes the signing algorithm of the MAYO signature scheme [Beu22, BCC⁺24] among parties holding additive shares of the secret key. The protocol of Aranha, Borin, Celi, and Niot [ABCN26] is formulated in the arithmetic black-box model over $\mathbb{F}_{16}$, with a message-independent preprocessing phase and a one-round online phase. Its emulation assumes that the correlated randomness consumed by the arithmetic black box, including multiplication triples and their MACs, is supplied by a trusted party, and leaves its generation open. This gives a direct application for our matrix-triple PCGs.

We consider the Round-2 MAYO2 parameter set [BCC⁺24],

$$(n, m, o, k, q) = (81, 64, 17, 4, 16),$$

for which

$$n - o = 64, ko = 68.$$

We focus on the Explicit-Standard variant of [ABCN26], which uses the standard $m \times ko$ linear system. The Explicit-Depth-Reduced variant replaces part of the signing computation by local operations but enlarges the system to $(k(n - o) + m) \times kn$ and correspondingly increases the offline arithmetic.

Several products in one signing attempt share an operand and can therefore be grouped before applying the square block decomposition from the beginning of this section. For example, the m products

$$VL_i, \quad V \in \mathbb{F}_{16}^{4 \times 64}, \quad L_i \in \mathbb{F}_{16}^{64 \times 17},$$

can be computed as

$$V[L_1 \mid \cdots \mid L_m] : (4 \times 64)(64 \times 1088).$$

Similarly, if

$$U_i = VP_i^{(1)} \in \mathbb{F}_{16}^{4 \times 64},$$

then all m products $U_i V^\top$ can be obtained from

$$\begin{pmatrix} U_1 \\ \vdots \\ U_m \end{pmatrix} V^\top : (256 \times 64)(64 \times 4).$$

Multiplication by the public matrices $P_i^{(1)}$ is linear in the shares of V and requires no multiplication triple.

The same grouping applies in the linear-system solver. Let

$$A \in \mathbb{F}_{16}^{64 \times 68}, \quad R \in \mathbb{F}_{16}^{64 \times 64}, \quad S \in \mathbb{F}_{16}^{68 \times 68},$$

and let

$$U = \text{diag}_k(O) \in \mathbb{F}_{16}^{256 \times 68}.$$

The products RA and Ry can be computed together as $R[A \mid y]$. Once $T = RAS$ has been opened and inverted, the products by $w = T^{-1}h + z$ and by $T^{-1}R$ can also be concatenated. Finally, the block-diagonal structure of U reduces US to products by the smaller oil matrix $O \in \mathbb{F}_{16}^{64 \times 17}$ [ABCN26]. After these regroupings, one Explicit-Standard signing attempt requires matrix multiplications of shapes

$$\begin{array}{ll}
(4 \times 64)(64 \times 1088), & (256 \times 64)(64 \times 4), \\
(64 \times 64)(64 \times 69), & (64 \times 68)(68 \times 68), \\
(64 \times 17)(17 \times 272), & (68 \times 68)(68 \times 65), \\
& (256 \times 68)(68 \times 65).
\end{array}$$

For the $q = 16$ instantiation of our twisted construction, Section 6.4 gives concrete working points with square dimensions $\nu = 27$ and $\nu = 81$. Both are supported by our concrete security estimates. We therefore do not use dimensions such as 5 or 15 merely because they are algebraically admissible: these dimensions lie below the range used in our concrete parameter selection.

Applying the block decomposition to the seven products above gives

ν	triples for the seven products	total
27	123, 30, 27, 27, 33, 27, 90	357
81	14, 4, 1, 1, 4, 1, 4	29.

Thus one MAYO2 signing attempt uses either 357 square 27×27 triples or 29 square 81×81 triples. The 27×27 choice introduces less padding; the 81×81 choice uses far fewer correlations.

The latter choice also gives a substantial reduction in the communication of the Beaver online phase. Scalarizing the seven products requires 2,650,640 scalar Beaver triples and therefore 5,301,280 opened elements of $\mathbb{F}_{16}$. Using independent matrix triples for the square blocks instead requires

$$2 \cdot 357 \cdot 27^2 = 520,506$$

opened field elements for $\nu = 27$, and

$$2 \cdot 29 \cdot 81^2 = 380,538$$

for $\nu = 81$. With 81×81 blocks, the communication due to these openings decreases by a factor of

$$\frac{5,301,280}{380,538} \approx 13.9.$$

This factor concerns only the secure multiplications above.

At the working point $(c, t) = (2, 81)$ with $N = 3^{10}$, Section 6.4 estimates approximately 0.6 core-ms of local expansion and 1578 bytes of amortized seed per party for one 81×81 triple. The 29 triples used in one MAYO2 attempt therefore cost about 17 core-ms of local expansion and 45 KiB of amortized seed per party.

An attempt is restarted when T is singular. For uniform R and S , this occurs with probability about

$$1 - (1 - 1/15)^2 \approx 0.13,$$

so a signature consumes about 1.15 attempts in expectation. Aranha et al. [ABCN26] provision correlated randomness for four attempts, making the probability that all attempts fail smaller than 2^{-11} . With 81×81 blocks this reserve contains $4 \cdot 29 = 116$ triples, corresponding to about 70 core-ms of local expansion and 179 KiB of amortized seed per party. Unused triples can carry over to later signatures, so at the expected consumption of 1.15 attempts the corresponding amortized cost is about 20 core-ms and 51 KiB per signature.

These communication figures do not use mask reuse across block products. A block of one operand can participate in several square products. If the PCG allows the corresponding mask to be reused, the parties need open that operand block only once, as in the matrix tiling of [BCG⁺20b]. For our construction, this requires reusing one party's sparse elements and the multiplier across the relevant

instances. Our present security proof does not establish this programmable extension, so we do not rely on it here.

Threshold MAYO also uses matrix multiplication during key generation. The one-time procedure contains products of the form

$$O^\top (P_i^{(1)} O - P_i^{(2)})$$

[ABCN26]. Across the Round-2 parameter sets, these require only 8–44 additional 81×81 triples after blocking. Since this cost is incurred only once and is a negligible fraction of one PCG batch, we omit it from the signing figures.

There are two remaining gaps between the correlation generated directly by our construction and the actively secure threshold protocol of [ABCN26]. First, our PCGs are two-party. A T -party matrix triple can be built from pairwise programmable instances: each party fixes shares $\mathbf{A}_i, \mathbf{B}_i$, and each pair (i, j) generates shares of $\mathbf{A}_i \mathbf{B}_j$ and $\mathbf{A}_j \mathbf{B}_i$.

Adding the diagonal and cross terms gives a sharing of the full product. Each party participates in $T - 1$ two-party instances, while the online multiplication phase is unchanged. This extension again uses the programmability discussed above.

Second, the actively secure protocol of [ABCN26] authenticates shared values using SPDZ-style MACs under a global secret key. Our matrix triples are unauthenticated. Authenticating $(\mathbf{A}, \mathbf{B}, \mathbf{AB})$ additionally requires shares of

$$\alpha \mathbf{A}, \quad \alpha \mathbf{B}, \quad \alpha \mathbf{AB}$$

for the MAC key α . This is a separate vector-OLE-type correlation, which we do not price here. Our calculation therefore isolates the matrix multiplication preprocessing supplied by the PCG: for MAYO2, 29 81×81 triples per attempt reduce the corresponding Beaver openings by a factor of about 13.9.

AI Disclosure

All ideas, results, and proofs of this paper are from the authors. A first draft of the technical overview, produced by Claude Fable 5.1 from a prompt detailing the content and the flow, was used as a baseline, but heavily reworked and mostly rewritten. All other sections were written by the authors without AI assistance (beyond standard polishing), except for Section 6, where the authors used AI assistance to facilitate the routine work of integrating standard PCG optimizations, counting operations, and estimating runtime for our constructions. Concretely, the authors selected three standard optimizations to integrate, listed works to credit for each optimization, wrote down the cost model, analyzed the effect of optimizations on security, and selected parameters to use in the evaluation. Then, an AI model (Claude Fable 5.1) was instructed to work out the resulting parameters of our two PCGs with all three optimizations included, to compute concrete efficiency estimates, and to write a draft of the result with tables. We subsequently reworked the draft and integrated it to Section 6. We also used Claude Fable 5.1 to port the SageMath script to Python (Section 6.4) and adapt it to the TQA-SD setting (we verified the content of the script it produced) and to run micro-benchmarks of cryptographic operations to obtain more accurate efficiency estimates (Section 6.5).

References

- [ABCN26] Diego F. Aranha, Giacomo Borin, Sofia Celi, and Guilhem Niot. Optimizing and implementing threshold MAYO. Cryptology ePrint Archive, Paper 2026/710, 2026. (Cited on page 6, 73, 74, 75.)
- [ABG⁺24] Amit Agarwal, Elette Boyle, Niv Gilboa, Yuval Ishai, Mahimna Kelkar, and Yiping Ma. Compressing unit-vector correlations via sparse pseudorandom generators. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VIII*, volume 14927 of *LNCS*, pages 346–383. Springer, Cham, August 2024. (Cited on page 3.)
- [ARR25] Amit Agarwal, Srinivasan Raghuraman, and Peter Rindal. Fully distributed multi-point functions for PCGs and beyond. Cryptology ePrint Archive, Report 2025/2294, 2025. (Cited on page 3, 6, 58, 64, 68, 69.)
- [AS22] Damiano Abram and Peter Scholl. Low-communication multiparty triple generation for SPDZ from ring-LPN. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 221–251. Springer, Cham, March 2022. (Cited on page 3, 6.)
- [BBC⁺24a] Maxime Bombar, Dung Bui, Geoffroy Couteau, Alain Couvreur, Clément Ducros, and Sacha Servan-Schreiber. FOLEAGE: $\mathbb{F}_4$ OLE-based multi-party computation for boolean circuits. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part VI*, volume 15489 of *LNCS*, pages 69–101. Springer, Singapore, December 2024. (Cited on page 3, 4, 5, 6, 7, 8, 9, 31, 45, 57, 58, 61, 62, 63, 64, 65, 67.)
- [BBC⁺24b] Maxime Bombar, Dung Bui, Geoffroy Couteau, Alain Couvreur, Clément Ducros, and Sacha Servan-Schreiber. FOLEAGE: $\mathbb{F}_4$ OLE-based multi-party computation for boolean circuits. Cryptology ePrint Archive, Paper 2024/429, 2024. (Cited on page 46.)
- [BCC⁺24] Ward Beullens, Fabio Campos, Sofia Celi, Basil Hess, and Matthias J. Kannwischer. MAYO: Algorithm specification and supporting documentation, round 2. Submission to the NIST call for additional post-quantum signatures, 2024. (Cited on page 73.)
- [BCCD23] Maxime Bombar, Geoffroy Couteau, Alain Couvreur, and Clément Ducros. Correlated pseudorandomness from the hardness of quasi-abelian decoding. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 567–601. Springer, Cham, August 2023. (Cited on page 3, 4, 5, 6, 7, 8, 9, 14, 18, 22, 38, 45, 61, 65.)
- [BCG⁺19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019. (Cited on page 3, 6.)
- [BCG⁺19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Cham, August 2019. (Cited on page 3, 6, 7, 16.)
- [BCG⁺20a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080. IEEE Computer Society Press, November 2020. (Cited on page 66.)

- [BCG⁺20b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 387–416. Springer, Cham, August 2020. (Cited on page 3, 4, 5, 6, 7, 8, 14, 63, 74.)
- [BCG⁺22] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 603–633. Springer, Cham, August 2022. (Cited on page 3, 6.)
- [BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018. (Cited on page 3, 6, 7.)
- [BCH⁺24] Youngjin Bae, Jung Hee Cheon, Guillaume Hanrot, Jai Hyun Park, and Damien Stehlé. Plaintext-ciphertext matrix multiplication and FHE bootstrapping: Fast and fused. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 387–421. Springer, Cham, August 2024. (Cited on page 3.)
- [BDHV25] Charles Bouillaguet, Claire Delaplace, Mickaël Hamdad, and Damien Vergnaud. Practical cryptanalysis of pseudorandom correlation generators based on quasi-abelian syndrome decoding. In Goichiro Hanaoka and Bo-Yin Yang, editors, *ASIACRYPT 2025, Part IV*, volume 16248 of *LNCS*, pages 436–465. Springer, Singapore, December 2025. (Cited on page 4, 12, 61.)
- [Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO’91*, volume 576 of *LNCS*, pages 420–432. Springer, Berlin, Heidelberg, August 1992. (Cited on page 3.)
- [Beu22] Ward Beullens. MAYO: Practical post-quantum signatures from oil-and-vinegar maps. In Riham AlTawy and Andreas Hülsing, editors, *SAC 2021*, volume 13203 of *LNCS*, pages 355–376. Springer, Cham, September / October 2022. (Cited on page 73.)
- [BGH⁺25] Elette Boyle, Niv Gilboa, Matan Hamilis, Yuval Ishai, and Yaxin Tu. Improved constructions for distributed multi-point functions. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 2414–2432. IEEE Computer Society Press, May 2025. (Cited on page 64.)
- [BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367. Springer, Berlin, Heidelberg, April 2015. (Cited on page 7, 13.)
- [BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016. (Cited on page 7, 13, 14, 30, 31, 63.)
- [BGI18] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. Cryptology ePrint Archive, Report 2018/707, 2018. (Cited on page 41.)
- [BGPG20] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, and Shafi Goldwasser. Secure large-scale genome-wide association studies using homomorphic encryption. *Proceedings of the National Academy of Sciences*, 117(21):11608–11613, 2020. (Cited on page 3.)

- [BIB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In Piotr Rudnicki, editor, *8th ACM PODC*, pages 201–209. ACM, August 1989. (Cited on page 3.)
- [BØ23] Pierre Briaud and Morten Øygarden. A new algebraic approach to the regular syndrome decoding problem and implications for PCG constructions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 391–422. Springer, Cham, April 2023. (Cited on page 63.)
- [CD01] Ronald Cramer and Ivan Damgård. Secure distributed linear algebra in a constant number of rounds. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 119–136. Springer, Berlin, Heidelberg, August 2001. (Cited on page 3.)
- [CDKSS25] Geoffroy Couteau, Srinivas Devadas, Alexander Koch, and Sacha Servan-Schreiber. Pseudo-random correlation functions for garbled circuits. In Benny Applebaum and Huijia (Rachel) Lin, editors, *TCC 2025, Part II*, volume 16269 of *LNCS*, pages 462–497. Springer, Cham, December 2025. (Cited on page 3.)
- [CEN25] Soffia Celi, Daniel Escudero, and Guilhem Niot. Share the MAYO: Thresholdizing MAYO. In Ruben Niederhagen and Markku-Juhani O. Saarinen, editors, *Post-Quantum Cryptography - 16th International Workshop, PQCrypto 2025, Part I*, pages 165–198. Springer, Cham, April 2025. (Cited on page 3, 6, 73.)
- [CFC⁺25] Hyunghoon Cho, David Froelicher, Jeffrey Chen, Manaswitha Edupalli, Apostolos Pyrgelis, Juan R. Troncoso-Pastoriza, Jean-Pierre Hubaux, and Bonnie Berger. Secure and federated genome-wide association studies for biobank-scale datasets. *Nature Genetics*, 57(4):809–814, 2025. (Cited on page 3.)
- [CHM⁺25] Hyunho Cha, Intak Hwang, Seonhong Min, Jinyeong Seo, and Yongsoo Song. MatriGear: Accelerating authenticated matrix triple generation with scalable prime fields via optimized HE packing. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 2453–2471. IEEE Computer Society Press, May 2025. (Cited on page 3, 4, 71.)
- [CKL26] Jung Hee Cheon, Minsik Kang, and Junho Lee. Fast batch matrix multiplication in ciphertexts. In *Advances in Cryptology – CRYPTO 2026, Part II*, *LNCS*, pages 558–590. Springer, 2026. (Cited on page 3.)
- [CKM⁺26] Geoffroy Couteau, Alexander Koch, Nikolas Melissaris, Peter Scholl, Sacha Servan-Schreiber, and Xiayi Ye. On compressing linearly shared correlations. In *Theory of Cryptography – TCC 2026*, 2026. To appear. Cryptology ePrint Archive, Paper 2026/255. (Cited on page 3.)
- [CKP07] Ronald Cramer, Eike Kiltz, and Carles Padró. A note on secure computation of the Moore-Penrose pseudoinverse and its application to secure linear algebra. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 613–630. Springer, Berlin, Heidelberg, August 2007. (Cited on page 3.)
- [CKR⁺20] Hao Chen, Miran Kim, Ilya P. Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. Maliciously secure matrix multiplication with applications to private deep learning. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 31–59. Springer, Cham, December 2020. (Cited on page 3, 6, 70, 71.)

- [Con64] S. B. Conlon. Twisted group algebras and their representations. *Journal of the Australian Mathematical Society*, 4(2):152–173, 1964. (Cited on page 19.)
- [CRR21] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 502–534, Virtual Event, August 2021. Springer, Cham. (Cited on page 3, 6.)
- [CS19] Daniele Cozzo and Nigel P. Smart. Sharing the LUOV: Threshold post-quantum signatures. In Martin Albrecht, editor, *17th IMA International Conference on Cryptography and Coding*, volume 11929 of *LNCS*, pages 128–153. Springer, Cham, December 2019. (Cited on page 3.)
- [CT19] Rodolfo Canto-Torres and Jean-Pierre Tillich. Speeding up decoding a code with a non-trivial automorphism group up to an exponential factor. In *IEEE International Symposium on Information Theory, ISIT 2019*, pages 1927–1931, 2019. (Cited on page 45.)
- [CWB18] Hyunghoon Cho, David J. Wu, and Bonnie Berger. Secure genome-wide association analysis using multiparty computation. *Nature Biotechnology*, 36(6):547–551, 2018. (Cited on page 3.)
- [CYW⁺25] Jingwei Chen, Linhan Yang, Wenyuan Wu, Yang Liu, and Yong Feng. Homomorphic matrix operations under bicyclic encoding. *IEEE Transactions on Information Forensics and Security*, 20:1390–1404, 2025. (Cited on page 3.)
- [DFM24] André Duarte, Raul Antonio Ferraz, and César Polcino Milies. Twisted group algebras of abelian groups. *Finite Fields Their Appl.*, 95:102386, 2024. (Cited on page 48, 52, 54.)
- [DK94] Yuriy A. Drozd and Vladimir V. Kirichenko. *Finite-dimensional algebras*. Springer-Verlag, Berlin, 1994. (Cited on page 50, 51, 52.)
- [GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658. Springer, Berlin, Heidelberg, May 2014. (Cited on page 7, 14.)
- [Gil99] Niv Gilboa. Two party RSA key generation. In Michael J. Wiener, editor, *CRYPTO’99*, volume 1666 of *LNCS*, pages 116–129. Springer, Berlin, Heidelberg, August 1999. (Cited on page 3, 64.)
- [GJM⁺24] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. SIGMA: Secure GPT inference with function secret sharing. *PoPETs*, 2024(4):61–79, October 2024. (Cited on page 3.)
- [GL26] Craig Gentry and Yongwoo Lee. Fully homomorphic encryption for matrix arithmetic. In *Advances in Cryptology – CRYPTO 2026, Part II*, *LNCS*, pages 363–395. Springer, 2026. Cryptology ePrint Archive, Paper 2025/1935. (Cited on page 3, 5, 8, 9, 24, 25.)
- [Got99] Daniel Gottesman. Fault-tolerant quantum computation with higher-dimensional systems. *Chaos, Solitons & Fractals*, 10(10):1749–1758, 1999. (Cited on page 6, 60.)
- [GSB⁺17] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Privacy-preserving distributed linear regression on high-dimensional data. *PoPETs*, 2017(4):345–364, October 2017. (Cited on page 3.)

- [HjLHD22] Zhicong Huang, Wen jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 809–826. USENIX Association, August 2022. (Cited on page 3.)
- [HLC⁺22] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, pages 15718–15731, 2022. (Cited on page 3.)
- [HS14] Shai Halevi and Victor Shoup. Algorithms in HElib. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 554–571. Springer, Berlin, Heidelberg, August 2014. (Cited on page 3.)
- [HS18] Shai Halevi and Victor Shoup. Faster homomorphic linear transformations in HElib. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 93–120. Springer, Cham, August 2018. (Cited on page 3.)
- [JGB⁺24] Neha Jawalkar, Kanav Gupta, Arkaprava Basu, Nishanth Chandran, Divya Gupta, and Rahul Sharma. Orca: FSS-based secure training and inference with GPUs. In *2024 IEEE Symposium on Security and Privacy*, pages 597–616. IEEE Computer Society Press, May 2024. (Cited on page 70.)
- [JKLS18] Xiaoqian Jiang, Miran Kim, Kristin E. Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1209–1222. ACM Press, October 2018. (Cited on page 3.)
- [jLHG⁺25] Wen jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and Wenguang Chen. BumbleBee: Secure two-party inference framework for large transformers. In *NDSS 2025*. The Internet Society, February 2025. (Cited on page 3, 6, 70, 71.)
- [Jou26] Antoine Joux. A correlation duet: Correlation attacks on correlation generators. *Cryptology ePrint Archive*, Paper 2026/1126, 2026. (Cited on page 4, 12, 61, 62.)
- [JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 1651–1669. USENIX Association, August 2018. (Cited on page 3.)
- [Kar85] Gregory Karpilovsky. *Projective Representations of Finite Groups*, volume 94 of *Monographs and Textbooks in Pure and Applied Mathematics*. Marcel Dekker, New York, 1985. (Cited on page 18, 19.)
- [KLRS22] Mahimna Kelkar, Phi Hung Le, Mariana Raykova, and Karn Seth. Secure poisson regression. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 791–808. USENIX Association, August 2022. (Cited on page 6, 72.)
- [KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 830–842. ACM Press, October 2016. (Cited on page 3.)

- [KPR⁺26] Vladimir Kolesnikov, Stanislav Peceny, Rahul Rachuri, Srinivasan Raghuraman, Peter Rindal, and Harshal Shah. Block-accumulate codes: Accelerated linear codes for PCGs and ZK. In *Advances in Cryptology – CRYPTO 2026, Part VIII*, LNCS, pages 397–429. Springer, 2026. (Cited on page 3, 6.)
- [KPRR25] Vladimir Kolesnikov, Stanislav Peceny, Srinivasan Raghuraman, and Peter Rindal. Stationary syndrome decoding for improved PCGs. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part I*, volume 16000 of LNCS, pages 284–317. Springer, Cham, August 2025. (Cited on page 3, 6, 64.)
- [KRC⁺20] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CryptTFlow: Secure TensorFlow inference. In *2020 IEEE Symposium on Security and Privacy*, pages 336–353. IEEE Computer Society Press, May 2020. (Cited on page 3.)
- [LLX⁺26] Zhe Li, Hongqing Liu, Chaoping Xing, Yizhou Yao, and Chen Yuan. Faster pseudorandom correlation generators via Walsh–Hadamard transform. In *Advances in Cryptology – CRYPTO 2026, Part VIII*, LNCS, pages 430–461. Springer, 2026. (Cited on page 3, 4, 8, 9, 58, 60, 62, 63, 64, 65, 66, 67.)
- [LXY25a] Zhe Li, Chaoping Xing, Yizhou Yao, and Chen Yuan. Efficient pseudorandom correlation generators for any finite field. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part V*, volume 15605 of LNCS, pages 145–175. Springer, Cham, May 2025. (Cited on page 3, 4, 5, 6.)
- [LXY25b] Zhe Li, Chaoping Xing, Yizhou Yao, and Chen Yuan. Efficient pseudorandom correlation generators over $\mathbb{Z}/p^k\mathbb{Z}$. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part IV*, volume 16003 of LNCS, pages 207–240. Springer, Cham, August 2025. (Cited on page 3, 72.)
- [MG23] Johannes Mono and Tim Güneysu. Implementing and optimizing matrix triples with homomorphic encryption. In Joseph K. Liu, Yang Xiang, Surya Nepal, and Gene Tsudik, editors, *ASIACCS 23*, pages 29–40. ACM Press, July 2023. (Cited on page 3, 71.)
- [Mil26] Sam Miller. 1.2: Twisted group algebras and group cohomology (lecture notes), 2026. (Cited on page 49.)
- [MMST25] Peihan Miao, Alice Murphy, Akshayaram Srinivasan, and Max Tromanhauser. Pseudorandom correlation generators for multiparty beaver triples over $\mathbb{F}_2$. In Goichiro Hanaoka and Bo-Yin Yang, editors, *ASIACRYPT 2025, Part VII*, volume 16251 of LNCS, pages 474–504. Springer, Singapore, December 2025. (Cited on page 3, 6.)
- [MR18] Payman Mohassel and Peter Rindal. ABY³: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 35–52. ACM Press, October 2018. (Cited on page 3.)
- [MV24] Jules Maire and Damien Vergnaud. Secure multi-party linear algebra with perfect correctness. *CiC*, 1(1):29, 2024. (Cited on page 3.)
- [MW08] Payman Mohassel and Enav Weinreb. Efficient secure linear algebra in the presence of covert or computationally unbounded adversaries. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of LNCS, pages 481–496. Springer, Berlin, Heidelberg, August 2008. (Cited on page 3.)

- [MZ17] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*, pages 19–38. IEEE Computer Society Press, May 2017. (Cited on page 3, 6, 70, 71.)
- [NIW⁺13] Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. Privacy-preserving matrix factorization. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 801–812. ACM Press, November 2013. (Cited on page 3.)
- [NW06] Kobbi Nissim and Enav Weinreb. Communication efficient secure linear algebra. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 522–541. Springer, Berlin, Heidelberg, March 2006. (Cited on page 3.)
- [NWI⁺13] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy*, pages 334–348. IEEE Computer Society Press, May 2013. (Cited on page 3.)
- [Par25] Jai Hyun Park. Ciphertext-ciphertext matrix multiplication: Fast for large matrices. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VIII*, volume 15608 of *LNCS*, pages 153–180. Springer, Cham, May 2025. (Cited on page 3.)
- [PRL⁺23] Hongwu Peng, Ran Ran, Yukui Luo, Jiahui Zhao, Shaoyi Huang, Kiran Thorat, Tong Geng, Chenghong Wang, Xiaolin Xu, Wujie Wen, and Caiwen Ding. LinGCN: Structural linearized graph convolutional network for homomorphically encrypted inference. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, pages 20679–20694, 2023. (Cited on page 3.)
- [PZ88] J. Patera and H. Zassenhaus. The Pauli matrices in n dimensions and finest gradings of simple lie algebras of type a_{n-1} . *Journal of Mathematical Physics*, 29(3):665–673, 1988. (Cited on page 20.)
- [PZM⁺24] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. BOLT: Privacy-preserving, accurate and efficient inference for transformers. In *2024 IEEE Symposium on Security and Privacy*, pages 4753–4771. IEEE Computer Society Press, May 2024. (Cited on page 3.)
- [RRKK23] Pascal Reisert, Marc Rivinius, Toomas Krips, and Ralf Küsters. Overdrive LowGear 2.0: Reduced-bandwidth MPC without sacrifice. In Joseph K. Liu, Yang Xiang, Surya Nepal, and Gene Tsudik, editors, *ASIACCS 23*, pages 372–386. ACM Press, July 2023. (Cited on page 3, 4.)
- [RRT23] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. Expand-convolute codes for pseudorandom correlation generators from LPN. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 602–632. Springer, Cham, August 2023. (Cited on page 3, 6.)
- [RXW⁺22] Ran Ran, Nuo Xu, Wei Wang, Gang Quan, Jieming Yin, and Wujie Wen. CryptoGCN: Fast and scalable homomorphically encrypted graph convolutional network inference. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, pages 37676–37689, 2022. (Cited on page 3.)
- [Sch60] Julian Schwinger. Unitary operator bases. *Proceedings of the National Academy of Sciences*, 46(4):570–579, 1960. (Cited on page 6, 20.)

- [Sen11] Nicolas Sendrier. Decoding one out of many. In Bo-Yin Yang, editor, *PQCrypto 2011*, pages 51–67. Springer, Berlin, Heidelberg, November / December 2011. (Cited on page [61](#), [65](#).)
- [Ser77] Jean-Pierre Serre. *Linear Representations of Finite Groups*, volume 42 of *Graduate Texts in Mathematics*. Springer, New York, 1977. (Cited on page [17](#).)
- [SGRR19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: Improved constructions and implementation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1055–1072. ACM Press, November 2019. (Cited on page [64](#).)
- [Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969. (Cited on page [60](#).)
- [Ter99] Audrey Terras. *Fourier Analysis on Finite Groups and Applications*, volume 43 of *London Mathematical Society Student Texts*. Cambridge University Press, Cambridge, 1999. (Cited on page [17](#), [18](#), [22](#).)
- [Wey50] Hermann Weyl. *The Theory of Groups and Quantum Mechanics*. Dover Publications, New York, 1950. English translation of the second revised German edition. (Cited on page [6](#), [20](#).)
- [WGC19] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. *PoPETs*, 2019(3):26–49, July 2019. (Cited on page [3](#).)
- [WWP22] Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. Piranha: A GPU platform for secure computation. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 827–844. USENIX Association, August 2022. (Cited on page [3](#).)
- [WZJ23] Songlei Wang, Yifeng Zheng, and Xiaohua Jia. SecGNN: Privacy-preserving graph neural network training and inference as a cloud service. *IEEE Transactions on Services Computing*, 16(4):2923–2938, 2023. (Cited on page [3](#).)

(a) The rectangular product we want

$$\begin{array}{ccc}
 \begin{array}{|c|c|c|c|c|c|c|} \hline & & & & & & \\ \hline & & & & & & \\ \hline & & & & & & \\ \hline & & & & & & \\ \hline & & & & & & \\ \hline & & & & & & \\ \hline \end{array} & \times & \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} & = & \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \\
 X \in \mathbb{F}_q^{5 \times 7} & & Y \in \mathbb{F}_q^{7 \times 4} & & XY \in \mathbb{F}_q^{5 \times 4}
 \end{array}$$

↓ choose square block size $n = 3$

(b) Partition the matrices into blocks

$$\begin{array}{ccc}
 \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} & \times & \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \\
 X & & Y
 \end{array}$$

Interior blocks are already 3×3 .
Only boundary blocks are smaller.

↓ pad only the incomplete boundary blocks

(c) Complete the boundary blocks with zeros

$$\begin{array}{ccc}
 \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} & \times & \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \\
 X : 5 \times 7 \longrightarrow 6 \times 9 & & Y : 7 \times 4 \longrightarrow 9 \times 6
 \end{array}$$

original entries
 zero padding

Hence the product uses square 3×3 matrix multiplications.

Figure 4: Using square matrix triples for a rectangular product. We first partition the original matrices into blocks of side length at most $n = 3$, and then complete only the incomplete boundary blocks with zeros.