A New Approach to Large Party Beaver-Style MPC with Small Computational Overhead

Aayush Jain*, Huijia Lin†, Nuozhou Sun‡

Abstract

Secure multi-party computation (MPC) enables N parties to jointly evaluate any function over their private inputs while preserving confidentiality. While decades of research have produced concretely efficient protocols for small to moderate numbers of participants, scaling MPC to thousands of parties remains a central challenge. Most of the existing approaches either incur per-party costs linear in N, due to pairwise computations, or rely on heavy cryptographic tools such as homomorphic encryption, which introduces prohibitive overheads when evaluating Boolean circuits.

In this work, we introduce a new lightweight approach to designing semi-honest MPC protocols with per-party, per-gate computation and communication costs that are independent of N. Our construction leverages the Sparse Learning Parity with Noise (Sparse LPN) assumption in the random oracle model to achieve per-gate costs of $O(k^2 \cdot c(\lambda))$ computation and $O(c(\lambda))$ communication, where k is the sparsity parameter for the Sparse LPN assumption and $c(\lambda)$ is an arbitrarily small super-constant in the security parameter λ. Assuming Sparse LPN remains hard for any super-constant sparsity, this yields the first semi-honest MPC protocol in the dishonest-majority setting with per-party, per-gate costs bounded by an arbitrarily small super-constant overhead in λ.

Structurally, our MPC instantiates a Beaver-style MPC with the required correlations generated efficiently. Departing from prior approaches that generate Beaver triples either silently (Boyle et al., 2019; 2020; 2022) or through homomorphic computation (Damgård et al., 2012), our focus is on efficiently generating a weaker correlation. In particular, using Sparse LPN we show that if we relax the correctness requirement in generating random Beaver triples to permit a tunably small inverse-polynomial error probability, such triples can be *silently* generated with arbitrarily small super-constant per-party computation. We then show that such correlations can be used in an efficient online phase similar to Beaver's protocol (with a tiny super-constant factor blow-up in communication).

*Carnegie Mellon University. Email: aayushja@andrew.cmu.edu.
†UW. Email: rachel@cs.washington.edu.
‡Carnegie Mellon University. Email: nuozhous@andrew.cmu.edu.

Contents

1 Introduction **1**
- 1.1 Our Results . . . 3
- 1.2 Related Works . . . 5

2 Technical Overview **6**
- 2.1 New Techniques for Leveraging Sparse LPN and LPN Variants . . . 6
- 2.2 Construction and Definition of Noisy Pseudocorrelation Function . . . 9
- 2.3 MPC Path 1: Error Correction and Security Amplification . . . 10
- 2.4 MPC Path 2: MPC via Fault Tolerance . . . 12
- 2.5 Extensions . . . 14
- 2.6 Comparison with Concrete Schemes for Large-Party Settings . . . 14

3 Preliminaries **16**
- 3.1 Learning Parity with Noise . . . 17
- 3.2 Sparse Learning Parity with Noise . . . 18

4 Useful Results on Learning Parity with Noise **18**

5 Noisy Pseudocorrelation Function for Beaver Triples **23**
- 5.1 Construction of Noisy PCF . . . 24

6 Extensions **28**
- 6.1 Working with a Constant Fraction of Corruptions . . . 28
  - 6.1.1 Tool: Pseudorandom zero sharing . . . 29
  - 6.1.2 Construction of Noisy PCF from PRZS . . . 31
- 6.2 Working with Polynomial-Size Fields . . . 34
- 6.3 Reduced Storage in Honest Majority Setting . . . 35

7 MPC Protocols with Small Super-Constant Overhead **36**
- 7.1 MPC Protocol Using Security Amplification . . . 36
- 7.2 MPC Protocol via Fault Tolerance . . . 40

8 Concrete Efficiency **44**

9 References **47**

1 Introduction

Multi-party computation (MPC) [Yao82, GMW87] allows N parties to jointly and securely compute any function $f(x_1, \ldots, x_N)$ of their private inputs $(x_1, \ldots, x_N)$. Motivated by applications in AI, data analytics, and cross-organization collaboration, there is a rapidly growing demand for protocols that remain efficient in the large-party setting—ranging from cross-silo collaborations to federated learning scenarios involving thousands of participants [KMA+21, BIK+17].

Large Party Challenge. While protocol design has made remarkable progress, achieving efficiency (even) in the semi-honest model with a large number of parties remains challenging. In many of the most well-known paradigms, such as GMW protocol [GMW87], the per-party cost per gate is $O(N)$, which directly reduces efficiency as N grows. While Beaver's protocol [Bea92] can achieve $O(1)$ per-party computation[1], efficiently generating the required Beaver triples remains a challenging problem. Even with more refined ways of generating Beaver triples—such as via OT extension [ALSZ13, KPR18] or pseudorandom correlation techniques [BCGI18, BCG+19, BCG+20b, BCG+22]—the protocols still incur either $O(N)$ computation or expensive pairwise correlation costs. Very few lightweight techniques are known that avoid this $O(N)$ barrier without resorting to heavy cryptographic tools such as fully homomorphic encryption (FHE) [Gen09] or indistinguishability obfuscation ($i\mathcal{O}$) [GGH+13], which, although asymptotically independent of N still incur large $\mathsf{poly}(\lambda)$ overheads in the security parameter and remain far from practical. This motivates the following question:

> *Can we design lightweight semi-honest MPC protocols that achieve substantially better per-party, per-gate computation costs in the large-party setting?*

In this work, we present a new semi-honest protocol that surpasses all previously known approaches in terms of computation cost, while maintaining low overall communication. In particular, our protocol achieves per-party, per-gate computation and communication of $O(c(\lambda))$, where c is an arbitrarily small super-constant function of the security parameter λ. Our construction builds on a new way of leveraging the well-studied *Sparse Learning Parity with Noise (Sparse LPN)* assumption, which has been investigated extensively as a standard hardness assumption in cryptography [Ale03]. We refer to Table 1 for a comparison of the computation and communication profiles of representative semi-honest MPC protocols from different approaches, and to Section 2.6 for further comparison.

In a nutshell, our protocol—like other lightweight MPC methods that rely on Beaver correlations [DPSZ12, ALSZ13, KPR18, BCGI18, BCG+19, BCG+20b, BCG+22]—first generates Beaver correlations and then executes Beaver MPC. However, unlike programmable PCG-based approaches, which suffer from the cost of pairwise PCG evaluations, our protocol's per-party, per-gate computation is independent of N, yielding an $O(N)$-fold speed-up in computation while incurring only a c-factor increase in communication, where c is an arbitrarily small super-constant in the security parameter λ. In contrast to approaches that generate Beaver correlations via homomorphic computation [DPSZ12], our protocol improves both computation and communication by at least polylogarithmic factors, making it potentially attractive for low-latency applications in large party settings. The following theorem summarizes our main result:

[1] In the semi-honest model with point-to-point channels, Beaver's protocol achieves $O(1)$ per-party computation per gate if one party (rotated in a round-robin manner) aggregates the masked shares and returns the result, instead of requiring all parties to broadcast.

Approach	Computation (per party / gate)	Communication (per party / gate)	Assumption
GMW [GMW87]	$O(N)$	$O(N)$	OT
Beaver's MPC with Beaver triple correlation [Bea92]	$O(1)$	$O(1)$	
Beaver triples via HE [DPSZ12]	$\mathsf{poly}\log(\lambda+N)$[a]	$\mathsf{poly}\log(\lambda+N)$	Ring LWE
Beaver triples via OT [ALSZ13, KPR18]	$O(N)$	$O(\lambda \cdot N)$	OT
Beaver triples via programmable PCGs/PCFs [BCG+19, BCG+20b, BCG+22]	$O(N)$	$O(1)$[b]	LPN variants
HSS-based protocol [DIJL23]	$\mathsf{poly}(\lambda)$	$o(1)$[c]	Sparse LPN
FHE-based MPC [Gen09, GHS12]	$\mathsf{poly}(\lambda)$[d]	$o(1)$[e]	LWE variants
iO-based MPC [GGH+13]	$\mathsf{poly}(\lambda)$	$o(1)$[e]	iO
Our protocol $\pi_{\mathsf{MPC}}^{\mathsf{CorrectTriple}}$	$O(k^2 \cdot c(\lambda))$[f]	$O(c(\lambda))$	k-Sparse LPN & ROM
Our protocol $\pi_{\mathsf{MPC}}^{\mathsf{FaultTolerance}}$	$O(k^2 \cdot c(\lambda))$	$O(c(\lambda))$	k-Sparse LPN & ROM

[a] In [DPSZ12], the authors only claim $O(N)$ per-party computation for the preprocessing phase that generates Beaver triples, as their focus is malicious security. Using the same technique without MACs and proofs yields the stated computation and communication costs.

[b] Using non-programmable PCGs/PCFs for OT requires $O(N)$ communication, as in the GMW protocol.

[c] [DIJL23]'s MPC protocol achieves this communication on layered circuits. In certain cases, it can trade computation for communication, yielding only a small super-constant computation overhead (see Section 1.2).

[d] [GHS12] achieves $O(\mathsf{poly}\log(\lambda+N))$ performance for circuits with bounded width. But for general circuits, the computation runs in $\mathsf{poly}(\lambda)$ time.

[e] In this case, the communication cost depends on the number of inputs and outputs rather than on the overall circuit size.

[f] $c(\lambda)$ is an arbitrarily small super-constant factor in λ.

Table 1: Comparison of per-party computation and communication costs (per gate) for representative semi-honest MPC protocols. Here, λ denotes the security parameter, N the number of parties, and the reported costs are measured in bits when evaluating Boolean circuits. (See Section 3 for details on the computational and communication models.)

Theorem 1.1. *(Informal) Assume the k-Sparse LPN assumption. Given reusable initial preprocessing, there exists a semi-honest MPC protocol in the random oracle model for N parties and $N-1$ corruptions that securely evaluates any Boolean circuit C with per-party, per-gate computation cost $O(k^2 \cdot c(\lambda))$ and communication cost $O(c(\lambda))$, where $c(\lambda)$ is an arbitrarily small super-constant in the security parameter λ. Importantly, these costs are independent of the number of parties N.*

By making the assumption that Sparse LPN remains hard for any $k \in \omega(1)$, we obtain the first semi-honest MPC protocol in the dishonest-majority setting with only an arbitrarily small super-constant overhead.

1.1 Our Results

Our main contribution is a new notion of pseudorandom correlation function (PCF) called a noisy PCF. In this work, we show that noisy PCFs are easier to design than standard PCFs, especially in the large-party setting, and they offer substantially better efficiency as a function of the number of parties. We construct such a noisy PCF by devising new techniques based on LPN and sparse LPN. The difference from prior PCFs is that a noisy PCF may output faulty correlations with a tunable inverse-polynomial probability ϵ. In addition, we show that the noisy nature of our triples can be overcome by working with a slight modification of Beaver MPC with an arbitrarily small super-constant overhead. We now go over all our contributions in more detail.

Result 1: Noisy PCF for Multi-Party Beaver Correlations. We formalize and construct from the Sparse Learning Parity with Noise assumption the notion of a noisy PCF for Beaver correlations. These are PCFs that output pseudorandom sharings of Beaver triples. However, unlike regular (non-noisy) PCFs, each triple, with a small tunable inverse-polynomial probability ϵ, might be faulty. In other words, with ϵ probability the triple might correspond to the form $(a, b, a \cdot b + e)$ where a, b are random bits but e could be non-zero and potentially known to the corrupted parties.

There are several advantages of considering our noisy PCFs, especially in the multi-party setting over standard PCGs and PCFs. First and foremost, noisy PCFs can directly be designed to support any number of parties with an evaluation time that is independent of N, unlike all prior programmable PCGs which suffered from pairwise evaluation. Moreover, as we show, noisy PCFs can be constructed from well-studied assumptions such as Sparse LPN, or even standard LPN, while some prior PCGs/PCFs [BCG+19, BCCD23, BCG+22, CD23, BCM+24] rely on novel assumptions or special code assumptions. Thirdly, as described shortly, the noisy nature of our PCFs can be overcome by designing Beaver-MPC–style protocols with a very small super-constant overhead.

We explain the nuances of this new definition in the Technical Overview, Section 2, and the formal details can be found in Definition 5.1. Our theorem shows:

Theorem 1.2 (Informal)**.** *Let λ be the security parameter. Let the number of parties $N = N(\lambda)$ be any polynomial of λ. Assume that Sparse LPN with polynomial dimension $n(\lambda)$ and sparsity parameter $k(\lambda)$, noise probability $\varepsilon = \varepsilon(\lambda)$ is secure. Then, there exists a noisy PCF scheme for $N - 1$ out of N corruptions for Beaver correlations with triple error probability $\epsilon = \Omega(N \cdot \varepsilon)$ and evaluation time $O(k^2)$.*

We show two different MPC approaches to use such a noisy PCF.

Result 2: Beaver MPC via Fault-Tolerant Circuits. As the first approach, we show how to directly use faulty triples in a Beaver-MPC–style protocol to securely compute functions in the online phase. One can observe that in Beaver MPC, if one uses a noisy triple instead of a noiseless one, such a triple corresponds to evaluating a faulty multiplication gate (faulty with error probability ϵ). Thus, if the parties first encode the circuit that they want to compute into its fault-tolerant version using classical results by Von Neumann and others [VN56a, Pip85], they can still compute the desired function. If ϵ is a constant, any circuit C of size s can be compiled into another circuit robust against an ϵ-fraction of faulty gates, with size $O(s \log s)$. In our case, where ϵ is inverse-polynomial, one can construct fault-tolerant circuits of size $O(s)$. This yields a linear per-party communication and computation overhead in the size of the circuit.

There is, however, one subtle challenge. At the conclusion of the computation, the output must be securely opened, as in standard Beaver-based MPC. In our fault-tolerant setting, the final output may still contain a small but inverse-polynomial probability error. While simply repeating the

evaluation of the computation a super-constant number of times yields the correct output, the revealed noise might depend on the internal state of the computation, potentially jeopardizing security.

We show that if, before opening, the parties compute a small (super-constant) number of layers of iterated majority circuits, the resulting output can be securely opened and the error leakage provably does not reveal any sensitive information about the internal computation. An overview of this idea can be found in the Technical Overview section, with full details provided in Section 7.2.

Result 3: Error Correction and Security Amplification. Beyond fault tolerance, we also develop a general method to correct errors and amplify security. Specifically, we design a simple two-message protocol that consumes κ triples and identifies one that is correct with probability at least $1 - O(\epsilon^\kappa)$.

These correct triples, however, satisfy only a weak security guarantee. Concretely, with probability at least $1 - O(\epsilon^\kappa)$, each triple is indistinguishable from a uniformly random Beaver triple. However, with a small (inverse-polynomial) probability, the triple may instead correspond to pseudorandom shares of $(a, b, a \cdot b)$ in which one of the inputs, a or b, is slightly biased.

Such triples can then be used to evaluate any circuit that is resilient to side-channel attacks, or equivalently, any circuit compiled into a leakage-resilient form using prior techniques [GR12, ISW03, IS24]. Better leakage compilers would directly yield more efficient protocols in our approach.

Instead of relying on a generic leakage compiler, we provide a very simple and concrete alternative. In this process, every multiplication gate is replaced by a security-amplified multiplication. Namely, instead of computing $x \cdot y$ from shares of x and y, each party locally generates random shares of random values $x_1, \ldots, x_\beta$ for some parameter β subject to $x = x_1 + \ldots + x_\beta$ (and similarly $y_1, \ldots, y_\beta$ for y). The parties then compute shares of $x \cdot y$ by computing shares of $(\sum_i x_i) \cdot (\sum_j y_j)$ using β^2 correct but weakly secure triples.

In terms of asymptotic overhead, this process requires $\kappa \cdot \beta^2$ noisy PCF triples to perform one multiplication. Relying on LPN/Sparse LPN assumptions with sufficiently small inverse-polynomial noise, one can choose both κ, β to be arbitrarily small super-constant factors, thereby obtaining standard negligible security.

This approach is described further in the Technical Overview, with full details in Section 7.1.

Result 4: New Ways to Leverage LPN and its Variants. Our results are enabled by a new way to leverage the LPN assumption. Namely, we consider indistinguishability of the distribution of the following kind:

$$\begin{gathered}\{\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s}_1\rangle + e_{1,i}, \langle \boldsymbol{a}_i, \boldsymbol{s}_2\rangle + e_{2,i}, \langle \boldsymbol{a}_i, \boldsymbol{s}_1\rangle \cdot \langle \boldsymbol{a}_i, \boldsymbol{s}_2\rangle + e_{3,i}\}_{i\in[m]} \\ \approx_c \\ \{\boldsymbol{a}_i, r_{1,i} + e_{1,i}, r_{2,i} + e_{2,i}, r_{1,i} \cdot r_{2,i} + e_{3,i}\}_{i\in[m]},\end{gathered}$$

where $\boldsymbol{a}_i$'s are chosen as random coefficients according to the LPN assumption and r's are random bits. We show that the above indistinguishability follows from LPN. In fact, the coefficient vectors can be chosen according to any variant of LPN, e.g., Sparse LPN. We believe that these new techniques to use LPN may find use in other applications. These LPN-related lemmata can be found in Section 4.

Result 5: Case of Constant Fraction Corruption, Performance Optimizations and Other Extensions. Beyond our core construction, we also investigate several refinements that further improve concrete efficiency in different practical scenarios. These extensions highlight the flexibility of

our approach in adapting to both the structure of the participating parties and the nature of the computation.

For instance, when only a constant fraction of parties may be corrupted, we reduce the pre-processing storage size from growing quadratically in N to be independent of N by introducing a novel pseudorandom zero-sharing technique.

We also show that our framework extends naturally to other small prime moduli greater than 2, demonstrating that it is not confined to Boolean circuits. Finally, in settings with an honest majority (which is common in settings related to blockchain), we present additional optimizations that substantially reduce resource requirements both in computation and communication.

Taken together, these refinements expand the applicability of our techniques and underscore their potential in a broad range of multi-party computation deployments. Details about these extensions can be found in Section 6.

1.2 Related Works

Current PCGs/PCFs Pseudorandom correlation generators (PCGs) expand short correlated seeds into long streams of correlated randomness without further interaction between parties [BCGI18, BCG+19, BCCD23]. Boyle *et al.* later introduced pseudorandom correlation functions (PCFs) [BCG+20a], which generalize PCGs by allowing parties to derive arbitrarily many correlations from a single setup.

Our primitive, a noisy PCF, differs from prior PCFs in both motivation and design. Whereas earlier constructions emphasized producing exact correlations—often in the two-party case and only secondarily extended to many parties—our focus is scalability. We trade a small amount of noise for significantly reduced per-party overhead in large-party settings. This divergence in goals yields distinct benefits: noisy PCFs are efficient and scalable, while traditional PCFs provide exactness where accuracy is critical. Thus, the two approaches are best seen as complementary rather than directly comparable.

Multi-party Homomorphic Secret Sharing [DIJL23] proposed a multi-party homomorphic secret sharing (HSS) scheme from Sparse LPN, yielding a sub-linear MPC protocol (see Table 1). Under the hardness of k-Sparse LPN for super-constant k, their HSS supports computation with arbitrarily small super-constant overhead for a constant number of multiplication rounds. Thus, if restricted to a constant number of layers per round—rather than $\log\log(|C|)$ layers—their protocol achieves efficient evaluation for most of the circuit.

In contrast to ours, however, two bottlenecks limit their efficiency. First, the parties must run an auxiliary MPC that evaluates majority gates per output, incurring a $\mathsf{poly}(\lambda, N)$ overhead. This overhead arises because even small output errors could compromise secret keys, whereas our protocol remains secure regardless of how outputs are revealed. Second, their evaluation homomorphically implements majority gates at each step, costing $O(k^{2+c(\lambda)})$ per gate, while even addition gates remain non-free. Together, these constraints prevent their protocol from achieving the small per-party, per-gate overhead that our approach attains.

MPC with Small Computational Overhead [IKOS08] showed that secure two-party computation can be achieved with constant-factor overhead in the semi-honest model, using randomized encodings and OT extensions. Building on this, [BCG+23] constructed a constant-overhead PCG for OT under constant-sparsity LPN, enabling constant-overhead MPC even against malicious adversaries. However, both results remain limited to the two-party setting.

Our protocol, while in general achieving only super-constant overhead, is designed to address the core challenges of the N-party setting. Notably, under constant-sparsity LPN (and other parameters also constant), our construction can in principle attain constant overhead. In this case, however, the number of gates that can be evaluated is bounded by a polynomial, and the final output inherits an inverse-polynomial error rate.

2 Technical Overview

In this section, we give an overview of the techniques used in this work. All our results are based on new methods for leveraging Sparse LPN (and, more generally, standard LPN and its variants). Sparse LPN, rather than standard LPN, is used to ensure that the per-party computation needed to evaluate a gate in MPC is sub-logarithmic/super-constant in the number of parties. We begin by presenting the new techniques and then proceed to the formal definitions of noisy PCF and MPC in Sections 2.2 and 2.3.

Before presenting the techniques, we first describe our computational model (see Section 3 for details). Like prior work on constant overhead cryptography [BCG+23], computations are represented as circuits, and the cost is measured by the circuit size. In this model, computing the inner product between a public k-sparse vector $\boldsymbol{a}$ and a secret vector $\boldsymbol{s}$ incurs computational cost $O(k)$. Furthermore, because our protocol and many prior protocols require a one-time preprocessing step to distribute initial correlated randomness, this computation is not counted toward the cost if it can be reused in many executions of the protocol.

2.1 New Techniques for Leveraging Sparse LPN and LPN Variants

Our core technique for obtaining noisy Beaver triples that are useful for multi-party computation relies on a new way to exploit the structure hidden within the Learning Parity with Noise (LPN) assumption. Recall that the LPN assumption posits indistinguishability of randomly generated noisy *linear* samples of the form

$$\langle \boldsymbol{a}, \boldsymbol{s} \rangle + e,$$

where $\boldsymbol{a}$ is a randomly sampled coefficient vector, $\boldsymbol{s}$ is a fixed but randomly generated secret vector, and e is a noise bit, typically sampled from a Bernoulli distribution. In this work, we primarily consider the field of operation to be the Boolean field $\mathbb{F}_2$; however, the LPN assumption naturally generalizes to any prime field.

The key insight behind our work is that the hardness of distinguishing perturbed random linear equations extends further: when the modulus q is small, LPN implies that one can postulate hardness of distinguishing noisy random rank-one *quadratic equations* from an appropriate random distribution. This opens the door to constructing pseudorandom noisy triples.

Basic idea. Recall that Beaver triples consist of correlations of the form (u, v, w) where u and v are random bits and w is their product. In this work, we consider a natural relaxation where w only equals $u \cdot v$ with high probability.

Toward this, a natural starting point is to represent u and v as LPN samples:

$$u = \langle \boldsymbol{a}, \boldsymbol{s}_1 \rangle + e_1, \quad v = \langle \boldsymbol{a}, \boldsymbol{s}_2 \rangle + e_2,$$

where $\boldsymbol{s}_1, \boldsymbol{s}_2$ are independent secrets and e_1, e_2 are Bernoulli noise bits. The challenge is to obtain a corresponding w that behaves like the product $u \cdot v$. If we naively multiply these expressions, we

obtain:

$$u \cdot v = \big(\langle \boldsymbol{a}, \boldsymbol{s}_1 \rangle + e_1\big) \cdot \big(\langle \boldsymbol{a}, \boldsymbol{s}_2 \rangle + e_2\big).$$

Note that this expands as a quadratic function of the secrets and the error vectors. While the purely linear terms in the secrets $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ as well as the term $\langle \boldsymbol{a}, \boldsymbol{s}_1 \rangle \cdot \langle \boldsymbol{a}, \boldsymbol{s}_2 \rangle$ are secret-sharing friendly, it is not clear how to deal with the error-related terms as they need to be distributively generated. Distributed point functions can help [GI14], but they lack efficient instantiations in the large-party setting. Indeed, all known constructions of multi-party point functions pay a polynomial computational overhead in the number of parties.

In this work, we let go of the requirement to distributively compute a perfectly correct Beaver correlation. In other words, we will generate shares of correlations (u, v, w) where u and v are random and $w = u \cdot v$ only with high probability.

Namely, instead of generating shares of w as shares of $u \cdot v$ where u and v are LPN samples as above, we generate shares of w where w is a *noisy quadratic function* of the secrets:

$$w = \langle \boldsymbol{a} \otimes \boldsymbol{a},\ \boldsymbol{s}_1 \otimes \boldsymbol{s}_2 \rangle + e_3 = \langle \boldsymbol{a}, \boldsymbol{s}_1 \rangle \langle \boldsymbol{a}, \boldsymbol{s}_2 \rangle + e_3,$$

where e_3 is an independent Bernoulli error. Over $\mathbb{F}_2$,

$$u \cdot v = \langle \boldsymbol{a}, \boldsymbol{s}_1 \rangle \langle \boldsymbol{a}, \boldsymbol{s}_2 \rangle \ + \ e_1 \langle \boldsymbol{a}, \boldsymbol{s}_2 \rangle \ + \ e_2 \langle \boldsymbol{a}, \boldsymbol{s}_1 \rangle \ + \ e_1 e_2.$$

Thus w matches $u \cdot v$ except when the noise terms flip the product, and correctness is guaranteed except with small probability.

The remaining question is whether such a quadratic variant of LPN still preserves security. At first sight this assumption may look unfamiliar, or even suspicious: after all, it involves leakage that is quadratic in the secret. The key contribution of our work is to show that this quadratic assumption is not a new hardness assumption at all, but rather can be reduced back to the standard LPN problem. In other words, quadratic LPN is simply LPN in disguise. This insight forms the starting point for our construction, and is formalized later in Lemma 2.1.

Lemma 2.1 (Informal)**.** *Assume that the LPN assumption with dimension n holds. Then for randomly chosen $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ from $\mathbb{F}_2^n$, the following distributions are indistinguishable:*

$$\begin{array}{llll} \{\boldsymbol{a}_i, & \langle \boldsymbol{a}_i, \boldsymbol{s}_1 \rangle + e_{1,i}, & \langle \boldsymbol{a}_i, \boldsymbol{s}_2 \rangle + e_{2,i}, & \langle \boldsymbol{a}_i, \boldsymbol{s}_1 \rangle \cdot \langle \boldsymbol{a}_i, \boldsymbol{s}_2 \rangle + e_{3,i}\}_{i \in [m]} \\ & & \approx_c & \\ \{\boldsymbol{a}_i, & r_{1,i} + e_{1,i}, & r_{2,i} + e_{2,i}, & r_{1,i} \cdot r_{2,i} + e_{3,i}\}_{i \in [m]} \end{array}$$

where $\boldsymbol{a}_i$ are randomly chosen LPN coefficient vectors, each of the errors $\{e_{1,i}, e_{2,i}, e_{3,i}\}_{i \in [m]}$ is sampled i.i.d. as Bernoulli with some probability $\varepsilon \in (0, 1)$, and $r_{1,i}, r_{2,i}$ are uniformly random bits. (We write $r_{j,i} + e_{j,i}$ to keep the syntax parallel and because later we use correlated noise; for independent e this term is simply uniform.)

Looking ahead, Lemma 2.1 is the conceptual core of our noisy PCF construction. We will return to describing how to construct a noisy PCF based on the above lemma and our MPC framework shortly. For now, we describe how the proof of the above lemma goes through.

Proving the LPN with Noisy Product Leakage. We now describe how to prove Lemma 2.1. The intuition of our first step is to remove one of $\boldsymbol{s}_1$ or $\boldsymbol{s}_2$ while still showing hardness. Suppose we gradually move from the first "planted" distribution in the equation to the second "random" distribution by considering the following changes.

1. Replacing $\langle \boldsymbol{a}_i, \boldsymbol{s}_1 \rangle$ with uniformly random bits for different $\boldsymbol{a}_i$.

2. Then, replacing the second inner product $\langle \boldsymbol{a}_i, \boldsymbol{s}_2 \rangle$ with uniformly random bits for different $\boldsymbol{a}_i$.

To argue the indistinguishability of the first change described above, we need to argue that the following distributions are indistinguishable:

$$\{\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s}_1 \rangle + e_{1,i}, \langle \boldsymbol{a}_i, \boldsymbol{s}_2 \rangle + e_{2,i}, \langle \boldsymbol{a}_i, \boldsymbol{s}_1 \rangle \cdot \langle \boldsymbol{a}_i, \boldsymbol{s}_2 \rangle + e_{3,i}\}_{i \in [m]},$$
$$\{\boldsymbol{a}_i, r_{1,i} + e_{1,i}, \langle \boldsymbol{a}_i, \boldsymbol{s}_2 \rangle + e_{2,i}, r_{1,i} \cdot \langle \boldsymbol{a}_i, \boldsymbol{s}_2 \rangle + e_{3,i}\}_{i \in [m]}.$$

We claim that even if $\boldsymbol{s}_2$ is leaked (i.e., known to the adversary), these two distributions remain computationally indistinguishable under an appropriate LPN assumption.

For indices $i \in [m]$ where $\langle \boldsymbol{a}_i, \boldsymbol{s}_2 \rangle = 0$, the tuple reduces to reasoning about samples of the form $\{\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s}_1 \rangle + e_{1,i}, e_{3,i}\}$. The first component consists of standard LPN samples with noise probability ε, while the last term $e_{3,i}$ is an independent Bernoulli variable that can be simulated.

For indices i where $\langle \boldsymbol{a}_i, \boldsymbol{s}_2 \rangle = 1$, security follows if we can argue the following indistinguishability:

$$\{\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s}_1 \rangle + e_{1,i}, \langle \boldsymbol{a}_i, \boldsymbol{s}_1 \rangle + e_{3,i}\}_{i \in [m]}$$
$$\approx_c$$
$$\{\boldsymbol{a}_i, r_{1,i} + e_{1,i}, r_{1,i} + e_{3,i}\}_{i \in [m]}$$

We denote this indistinguishability as the 2LPN assumption (short for "two-copy LPN"), which states that repeated LPN samples with the same coefficient but independent errors are indistinguishable from a canonical distribution. This assumption is the crux of our argument and forms the basis for the security of our construction. The second indistinguishability mentioned earlier follows in a similar manner.

Two-Copy LPN is as secure as LPN: At first glance, 2LPN might look insecure, since we sample twice with the same $\boldsymbol{a}_i$ but fresh noise. If $e_{1,i}$ and $e_{3,i}$ are independent Bernoulli errors with probability ε, then taking the two copies and outputting their agreement (and otherwise guessing) seems to reduce the noise to $O(\varepsilon^2)$. Indeed, the event $e_{1,i} = e_{3,i}$ has probability $\varepsilon^2 + (1-\varepsilon)^2$, and conditioned on equality the probability that both errors are 1 is $\rho = \frac{\varepsilon^2}{\varepsilon^2 + (1-\varepsilon)^2} = O(\varepsilon^2)$, so agreement is biased toward 0. Nevertheless, the joint distribution

$$\left\{\boldsymbol{a}_i,\ \langle \boldsymbol{a}_i, \boldsymbol{s}_1 \rangle + e_{1,i},\ \langle \boldsymbol{a}_i, \boldsymbol{s}_1 \rangle + e_{3,i}\right\}$$

can be generated from a single LPN sample $b_i = \langle \boldsymbol{a}_i, \boldsymbol{s}_1 \rangle + e_i$ where e_i is a Bernoulli error with probability ρ: output (b_i, b_i) with probability $\varepsilon^2 + (1-\varepsilon)^2$, and otherwise output $(b_i, b_i + 1)$ or $(b_i + 1, b_i)$ with equal probability. Hence 2LPN is no easier than LPN, up to replacing ε by $O(\varepsilon^2)$.

Connection to Noise-Smudging. Another useful perspective is to view 2LPN as a "smudging lemma" for LPN, analogous to the smudging lemma for LWE. Suppose we are given two LPN samples

$$\left\{\boldsymbol{a}_i,\ \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + e_{1,i},\ \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + e_{2,i}\right\}_{i \in [m]}.$$

At a minimum, we must hide $\boldsymbol{s}$ under LPN hardness. Observe that the adversary's view can already be simulated if one knows

$$\left\{\boldsymbol{a}_i,\ \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + e_{1,i},\ e_{1,i} + e_{2,i}\right\}_{i \in [m]}.$$

In the LWE setting, where errors are discrete Gaussians, one argues that given the sum $e_{1,i} + e_{2,i}$ the conditional distribution of $e_{1,i}$ is still Gaussian, and thus indistinguishable [GMPW20]. In our setting the errors are Bernoulli. Here too, conditioning on $e_{1,i} + e_{2,i}$ yields a Bernoulli distribution for $e_{1,i}$ (with a parameter depending on the sum). This shows that the two-sample view can be reduced back to the standard LPN assumption.

This "smudging" viewpoint is insightful because it lets us reason about more general error distributions. In particular, later we will consider noise where $(e_{1,i}, e_{2,i})$ are not fully independent. Although such distributions may look contrived, they preserve correctness of our PCF scheme while enabling a tighter reduction to LPN with higher error probabilities.

Different Noise Distributions and Sparse LPN A subtle issue arises in the proof of our basic construction. In particular, the reduction incurs a *quadratic loss*: Lemma 2.1 holds with target noise rate ε only if we assume hardness of LPN with noise rate $O(\varepsilon^2)$. This mismatch is problematic, since it requires starting from an artificially smaller noise parameter than is necessary for our nPCF. Concretely, this leads to a quadratic blow-up in the dimension of the underlying LPN instances, which in turn has a significant impact on storage efficiency.

To address this, we introduce a correlated noise distribution, denoted $\mathcal{D}_e^3(\varepsilon)$, designed to eliminate the quadratic degradation. The distribution is defined as follows:

1. Sample control bits c_1, c_2 independently from the Bernoulli distribution $\mathsf{Ber}(\varepsilon)$.
2. Set $e_1 = 0$ if $c_1 = 0$, otherwise sample e_1 uniformly at random. Similarly, set $e_2 = 0$ if $c_2 = 0$, otherwise sample e_2 uniformly at random.
3. For e_3, set it to 0 if both c_1 and c_2 are 0, and otherwise sample e_3 uniformly at random.
4. Output (e_1, e_2, e_3).

We claim that Lemma 2.1 still holds when the noise terms are sampled from the distribution $\mathcal{D}_e^3(\varepsilon)$, and the proof is similar. Furthermore, a careful inspection shows that our proof strategy can work with any variant of LPN, not necessarily standard LPN; for instance, it works with Sparse LPN. For details, see Section 4.

2.2 Construction and Definition of Noisy Pseudocorrelation Function

We start by giving our construction below followed by our definitional framework for noisy PCF.

As the reader can perhaps guess, our construction is really intuitive and works as follows. In the setup phase, we simply issue to each party P_i an additive secret share $[\![\boldsymbol{s}_1]\!]_i$ of secret $\boldsymbol{s}_1$, $[\![\boldsymbol{s}_2]\!]_i$ of $\boldsymbol{s}_2$, and the share of the tensor product $[\![\boldsymbol{s}_1 \otimes \boldsymbol{s}_2]\!]$. Then, during the evaluation, each party invokes the random oracle to obtain a sparse vector $\boldsymbol{a}$, then computes and releases $\langle \boldsymbol{a}, [\![\boldsymbol{s}_1]\!]_i \rangle + e_{i,1}$, $\langle \boldsymbol{a}, [\![\boldsymbol{s}_2]\!]_i \rangle + e_{i,2}$, $\langle \boldsymbol{a} \otimes \boldsymbol{a}, [\![\boldsymbol{s}_1 \otimes \boldsymbol{s}_2]\!] \rangle + e_{i,3}$, where the error terms are privately sampled from the above-mentioned distribution (the one in which e_1, e_2, e_3 are chosen in a dependent manner using control bits). By applying a variant of Lemma 2.1, and under the Sparse LPN assumption, we can securely generate noisy Beaver triples in this manner.

To formalize our constructions, we introduce the notion of *noisy pseudorandom correlation functions* (noisy PCFs). A noisy PCF is specified by two algorithms, nPCF.Gen and nPCF.Eval. The randomized setup algorithm nPCF.Gen takes as input the security parameter λ, the number of parties N, and a corruption threshold t. It generates and outputs a collection of keys $\{k_i\}_{i \in [N]}$, one for each party. The deterministic evaluation algorithm nPCF.Eval is then run locally by each

party. On input its key k_i and a string $\boldsymbol{x}$, party i outputs a vector $\boldsymbol{y}_i$. A noisy PCF should have the following properties:

- *Correctness*: Let ϵ be the noise rate of the noisy PCF. The correctness property requires the noisy PCF to generate correct Beaver correlations with at least $1 - \epsilon$ probability.
- *Weak pseudorandomness*: Fix any corruption set $T \subseteq [N]$ with $|T| \leq t$ and any input $\boldsymbol{x}$. Write $\boldsymbol{y}_i = \mathsf{nPCF.Eval}(k_i, \boldsymbol{x})$, and interpret $\boldsymbol{y}_i = (u_i, v_i, w_i) \in \mathbb{F}_2^3$ as additive shares. There exists a PPT *noise indicator* $\mathbb{I}_A$ such that, given the corrupted keys k_T (and the public input $\boldsymbol{x}$), it outputs an error vector $\boldsymbol{e} \in \mathbb{F}_2^3$ for which the conditional distribution of $\{\boldsymbol{y}_i\}_{i \notin T}$ given $(k_T, \{\boldsymbol{y}_i\}_{i \in T})$ is computationally indistinguishable from the uniform distribution over all completions of the fixed corrupted shares to an additive sharing of $(u, v, u \cdot v) + \boldsymbol{e}$, where $(u, v) \leftarrow \{0,1\}^2$ are uniform.

A natural question is why our definition requires a noise indicator that depends on the adversary's keys, rather than allowing the error to be sampled independently. Indeed, such independence would be the most natural and strongest extension of the original PCF definition. Unfortunately, the noise-smudging technique in our construction prevents this independence: The error vector $\boldsymbol{e} = (e_1, e_2, e_3)$ is the sum of the error contributions from all parties. If the corrupted parties introduce non-zero errors, then with high probability the aggregate error will also be non-zero, creating a bias that necessitates a relaxed definition.

Nevertheless, this relaxed notion still suffices for MPC. In fact, our formulation is more general, encompassing a broader class of noisy PCFs and ensuring that our transformations remain widely applicable.

With minor modifications, the protocol for generating noisy Beaver triples described above can be naturally cast within the framework of noisy PCFs. (See Section 5 for details.) We emphasize that when sampling $\boldsymbol{a}$ from the k-sparse vector distribution, only $O(k^2)$ bit operations are required for one evaluation, yielding an efficient method for producing noisy triples. However, the noise rate of nPCF is at least $\Omega(N \cdot \varepsilon)$, where ε is the noise rate under the Sparse LPN assumption. To maintain a fixed noise rate, and given the standard tradeoff between dimension n and ε in Sparse LPN, the dimension must scale linearly with the number of parties, which in turn implies a quadratic storage cost. In Section 6, we present an optimization that reduces this storage overhead under the assumption of a constant corruption fraction. Before that, we describe two approaches to achieving secure MPC using only a small super-constant number of noisy triples per gate.

2.3 MPC Path 1: Error Correction and Security Amplification

If the error rate of a noisy PCF is already negligible, then its output triples can be treated as ideal Beaver triples. When this is not the case, our first idea is to apply error correction to further reduce the error rate. This approach unfolds in two steps:

- **Error detection.** We design a protocol that identifies triples which, with overwhelming probability, contain no error. Concretely, the protocol takes a "target" triple together with κ noisy "test" triples (each with error probability ϵ), and it detects whether the target triple is faulty. The protocol ensures that the target triple is correct except with probability $O(\epsilon^\kappa)$.
- **Security amplification.** Even after correctness is ensured, the resulting triples may still suffer from security issues: with inverse-polynomial probability, an adversary can learn partial information by observing when an error term was non-zero in the above detection procedure. This can jeopardize the security of the target triple. To address this, we conservatively treat

such triples as *potentially insecure*, and then show how to transform them into fully secure triples by combining many correct-but-weak triples through a security amplification step.

We begin by outlining the intuition behind the first step. Suppose the parties share a target noisy triple (α, β, γ) where $\gamma = \alpha \cdot \beta + \mu$ for some (potentially non-zero) error term μ, and they wish to determine whether this triple is a correct Beaver triple. If the parties had access to a secure Beaver triple (a, b, c), they could easily test correctness by computing $\mu = \gamma - \alpha \cdot \beta$ and reconstructing μ. In our setting, however, all available triples are themselves noisy, which prevents a direct and error-free test. Instead, when using a noisy triple (a, b, c) for testing, the parties effectively learn $\mu + c - a \cdot b$ rather than the desired μ. While this introduces additional noise, the situation can be remedied by performing multiple such tests in parallel and aggregating the outcomes via a local majority. If we execute $\kappa \in \omega(1)$ tests, then the parties can reliably infer the true value of μ with overwhelming probability.

Although this first step lets the parties predict, with overwhelming probability, whether the target triple is correct, it can inadvertently compromise the *security* of that triple. Each test is implemented via a Beaver-style multiplication using a noisy test triple (a_j, b_j, c_j), which opens the masked differences

$$\Delta_{a,j} := \alpha - a_j \qquad \text{and} \qquad \Delta_{b,j} := \beta - b_j,$$

and reconstructs a value of the form

$$\widetilde{\mu}_j = \mu + e_j, \qquad \text{where} \qquad e_j := c_j - a_j b_j$$

(up to an inessential sign). Thus, when the target triple is correct ($\mu = 0$), the transcript reveals information about the residuals e_j of the test triples (in particular, whether $e_j = 0$). Our nPCF construction, however, does not guarantee independence between e_j and (a_j, b_j): viewing a generated triple as

$$a_j = r_{1,j} + \eta_{1,j}, \quad b_j = r_{2,j} + \eta_{2,j}, \quad c_j = r_{1,j} r_{2,j} + \eta_{3,j},$$

the residual equals

$$e_j = \eta_{3,j} - \eta_{1,j} b_j - \eta_{2,j} a_j + \eta_{1,j} \eta_{2,j},$$

and a corrupted party may have partial knowledge/control over the η's through its Sparse-LPN samples. Consequently, conditioning on the revealed value of e_j (or just the event $e_j = 0$) can bias the adversary's posterior on (a_j, b_j); since $\Delta_{a,j}, \Delta_{b,j}$ are opened, this bias directly propagates into leakage about (α, β).

As a toy illustration over $\mathbb{F}_2$, in the extreme case $(\eta_{1,j}, \eta_{2,j}, \eta_{3,j}) = (1, 0, 0)$ we have $e_j = b_j$, so observing $\widetilde{\mu}_j = 0$ forces $b_j = 0$ and opening $\Delta_{b,j} = \beta - b_j$ reveals β. Therefore, even after correctness detection, we treat the output as *correct-but-weak* and apply a separate security amplification step to obtain pseudorandom Beaver triples.

An important observation is that the probability of obtaining a triple with non-zero noise terms is low, meaning that in most executions the parties will be working with correct and secure triples. To leverage this fact, we propose a security amplification technique for computing an AND gate. Concretely, given input bits x, y, the parties first apply the above procedure to identify β^2 correct (though potentially insecure) triples from the noisy ones, for some parameter $\beta \in \omega(1)$. Next, each party locally generates shares of β random values $x_1, \ldots, x_\beta$ such that $\bigoplus_i x_i = x$, and similarly $y_1, \ldots, y_\beta$ with $\bigoplus_i y_i = y$. Observe that

$$\bigoplus_i x_i \cdot \bigoplus_j y_j = \bigoplus_{i,j} x_i \cdot y_j = x \cdot y,$$

so the parties can compute each product $x_i \cdot y_j$ with the correct triples generated before, and then combine the results to obtain additive shares of $x \cdot y$. Let $(a_{i,j}, b_{i,j}, c_{i,j})$ denote the triple used to compute $x_i \cdot y_j$ for $i, j \in [\beta]$. If any party introduces a non-zero noise term—either directly in $(a_{i,j}, b_{i,j}, c_{i,j})$ or in one of the test triples used to validate it—then the triple $(a_{i,j}, b_{i,j}, c_{i,j})$ becomes insecure. Fortunately, the vast majority of triples do not suffer from this issue and remain safe to be used as random Beaver triples. A key observation is that it suffices for security if there exists at least one special row index i and one special column index j such that all triples of the form $(a_{i,\ell}, b_{i,\ell}, c_{i,\ell})$ (for $\ell \in [\beta]$) and $(a_{\ell,j}, b_{\ell,j}, c_{\ell,j})$ are secure. In this case, the amplification method described above guarantees a correct and secure multiplication. When $\beta = \omega(1)$ and the probability that each triple is bad is only inverse-polynomial, the existence of such indices holds with overwhelming probability.

The details can be found in Section 7.1.

Connection to Leakage Compilers and Side-Channel Attacks. An important point to note is that the step of doing secure multiplication may not be the only way to implement the circuit securely. Each time an insecure but correct triple is used, it results in a leakage of the state for evaluating that multiplication gate. However, these leakages are random and occur with a small inverse-polynomial probability. Therefore, if the circuit is already resilient to such leakages, this step may be skipped. This will save approximately β^2 overhead in our final MPC.

In the future, it might be possible to design highly efficient leakage compilers [GR12, ISW03, IS24], and moreover, studying side channel attacks on cryptographic circuits such as AES [OMPR05] is an important area on its own. We believe that such a study will help advance efficient MPC through our framework.

2.4 MPC Path 2: MPC via Fault Tolerance

On the other hand, noisy Beaver triples generated by our nPCF might still be helpful even without special preprocessing. If we forget about security for a minute, a natural idea is to use these correlations as in Beaver MPC, as if they formed correct triples, but to compute a fault-tolerant version $\widehat{C}$ of the same circuit C. Recall that a fault-tolerant circuit produces with high probability the same output even if some small fraction ϵ of the wire values are corrupted at random. Classical works in the 1950s [VN56b] showed that any circuit C of size s can be compiled to its fault-tolerant version $\widehat{C}$ with size $O(s \log s)$ provided ϵ is a small constant. In our case, ϵ is inverse-polynomially small, and $\widehat{C}$ can even be of size $O(s)$.

Note that this is enabled because if one uses a Beaver triple (a, b, c) where $c = a \cdot b + e$ as in Beaver MPC to compute a multiplication gate on input bits x and y, it can be seen that the computed output becomes $x \cdot y + e$. Therefore, a faulty Beaver triple amounts to computing a faulty multiplication gate with an added error e. The fault-tolerant circuit guarantees that the computed output is correct with large probability. To guarantee overwhelmingly correct output, however, we might have to run $\omega(1)$ copies in parallel and take a majority vote at the end.

What about security? Indeed, as we described previously when working with noisy triples, one needs to be a bit careful, as whenever (e_1, e_2, e_3) is partially guessable to the adversary, learning whether a triple is correct or not can create a bias in the triple values. When a circuit is computed and a multiplication gate is 'opened', one might learn whether the triple was correct or not, and the adversary's educated guess about (e_1, e_2, e_3) can compromise the security of the protocol.

We note, however, that for most of the evaluation this is never a problem. The reason is that in Beaver MPC, the outputs of internal multiplication gates are never opened in the clear. They are

always masked with one of the first two coordinates of a fresh Beaver triple. In our setting, note that in the ideal correlation, the triples are of the form $(a + e_1, b + e_2, a \cdot b + e_3)$ where (e_1, e_2, e_3) are independent of random bits a and b. This means that the marginal distributions of the first two coordinates, even given (e_1, e_2, e_3), are random. Therefore, in the ideal world, the entire transcript coming from the honest parties, until the very end, is statistically distributed as random.

The problematic part now becomes: how do we securely learn the computed outputs at the end? Recall that we are running $\kappa = \omega(1)$ executions of Beaver MPC on the fault-tolerant circuit $\widehat{C}$ in parallel. Opening the κ output bits computed this way at the end could be dangerous for security because this will leak information about which of the executions is noisy, which could lead to attacks in the same spirit as the attack described previously or at the very least make it unclear how to arrive at a security proof.

Decoding via Iterated Majority Our fix for this issue is simple. The idea is that we do not open the κ computed outputs, as above, but pass them through β layers of majority circuits on κ bits. To be precise, each layer takes κ input bits $\boldsymbol{z} \in \{0,1\}^\kappa$ and outputs $\underbrace{(\mathsf{Maj}(\boldsymbol{z}), \ldots, \mathsf{Maj}(\boldsymbol{z}))}_{\kappa \text{ times}}$.

These layers are evaluated as before using noisy triples generated by nPCF. Although this does not change the accuracy guarantees that more than half of the κ repetitions hold the correct value throughout the computation, it helps substantially with security. In particular, we show that after these β layers, the κ output bits of the final layer can be opened, exactly as in Beaver MPC, without compromising security.

This is proven in Theorem 7.2. We describe the idea only at an intuitive level here. The idea relies on a chaining/coupling type argument. Namely, one can show that the distribution of errors (whether the output of an execution is correct or not) only depends on the function output $C(x)$. The intuition is to find one layer of iterated majority so that all sensitive information is concealed after the layer. The condition we need is that all Beaver triples involved in that layer are secure and correct triples.

This is because one can show that with overwhelming probability, for each of the layers, the κ (noisy) outputs have a stable majority of $C(x)$. Because of this, when all triple correlations are secure and correct, the output of this layer will consist of κ shares of the correct function output $C(x)$. Moreover, because the Beaver triple correlations are secure, the shares held by honest parties will appear random to the adversary and will be independent of the noise in the κ executions of $\widehat{C}(x)$. Since the noise information in the κ executions of $\widehat{C}(x)$ is concealed, the subsequent computations and the final output cannot reveal any sensitive information. Therefore, if there exists a layer of iterated majority that uses only secure Beaver triples, directly opening the output bits after the entire iterated majority computation will preserve security. In one layer of the iterated majority computation, the κ majority gates with κ inputs are evaluated, requiring $\mathsf{poly}(\kappa)$ Beaver triples for the computation. Since κ is relatively small and the noise rate of nPCF is sufficiently low, there is a reasonable probability that all Beaver triples involved are secure, i.e. $\mathbb{I}_A$ outputs zero vectors for all triples. Given that there are β layers of iterated majority, with overwhelming probability, at least one layer will consist entirely of secure Beaver triples, ensuring the security of the protocol.

The details can be found in Section 7.2.

2.5 Extensions

We now discuss several extensions of our results that offer efficiency advantages in different useful settings.

- **Constant Fraction of Corruptions (Section 6.1).** Our basic construction from Sparse LPN suffers from quadratic storage in N: because the effective noise rate is $\eta = O(\epsilon/N)$, the dimension must grow linearly with N, and storing the tensor $\boldsymbol{s} \otimes \boldsymbol{s}$ incurs an N^2 overhead. This bottleneck disappears if only a constant fraction of parties are corrupted. In this regime, pseudorandom zero sharing (PRZS) enables parties to mask outputs so that only the global sum needs to form a valid Sparse LPN sample. From the adversary's perspective, distinguishing then reduces to Sparse LPN with error rate $\varepsilon = O(\epsilon \cdot (1 - \alpha))$ for α-fraction corruptions, instead of $\varepsilon = O(\epsilon/N)$. Consequently, the storage costs become independent of the number of parties. Furthermore, PRZS can be instantiated using a sparse random graph: if each party holds keys for only $\omega(\log \lambda)$ edges, then with high probability the honest subgraph remains connected, ensuring pseudorandomness while keeping per-party computation independent of N.

- **Polynomial-size Fields (Section 6.2).** Our framework extends naturally from $\mathbb{F}_2$ to polynomial-size fields $\mathbb{F}_q$. The construction remains unchanged, with noise analysis adjusted to the larger field. This allows applications where computation over larger fields is more natural.

- **Honest Majority (Section 6.3).** In the honest-majority setting, quadratic storage can be avoided via Shamir secret sharing over $\mathbb{F}_q$ with $q > N$. Its multiplicative property enables computation without storing $\boldsymbol{s} \otimes \boldsymbol{s}$, reducing computation to $O(k)$ field operations (plus PRZS and PRF calls). For Boolean circuits, we can use extension fields and project outputs back to $\mathbb{F}_2$.

2.6 Comparison with Concrete Schemes for Large-Party Settings

In this section, we discuss the practicality of our scheme and compare it with state-of-the-art protocols for large-party MPC with a trusted once-for-all setup. That is, after setup, the parties should be able to evaluate an arbitrary number of circuits. For an asymptotic comparison in the case of $N - 1$ corruptions, see Table 1; here we focus only on concrete efficiency.

While recent PCG/PCF-based techniques have led to efficient two-party MPC [BCG+19, BCG+20b, BCG+22], the concrete efficiency of state-of-the-art protocols for large-party MPC remains far from satisfactory. Even for moderately large settings such as $N = 100$, known protocols have prohibitively large communication, and their computation often depends quadratically on N. Designing efficient MPC protocols tailored to the large-party setting is therefore a pressing open question. Since our protocol can be optimized depending on whether the number of corruptions is $N - 1$, αN for a constant $\alpha < 1$, or below an honest-majority threshold, we compare our protocol with the state of the art in these three settings.

The main conclusion of this comparison is twofold:

- First, the state of the art for MPC in the large-party setting is not yet very competitive concretely.

- Second, our protocol has comparable communication efficiency, and in some cases may even improve communication over the state of the art. Its per-party, per-gate computation is

theoretically $\widetilde{O}(1)$, but the current instantiation is not as computationally efficient in practice as some alternatives. The main reason is the storage bottleneck for the seed component $s\mathrm{vect}s$, whose length depends on N in the $N-1$ corruption case. For realistic parameters, this storage must reside in RAM, which slows the protocol because computation requires random seed lookups. We believe that improving this storage cost is an important direction toward competitive practical computation efficiency, and we outline some possible directions below.

We now discuss the comparison in more detail in the three cases.

Case of $N-1$ Corruptions. In the full dishonest-majority setting, the most natural baselines are homomorphic-encryption/SPDZ-style Beaver-triple generation [DPSZ12, AS22] and silent-preprocessing approaches based on programmable PCGs/PCFs [BCG+19, BCG+20b, BCG+22, BCCD23]. Packed Ring-LWE or Ring-LPN techniques amortize expensive work across many triples, but, for example, in thousand-party MPC they require arithmetic over moderately large moduli, roughly in the 60–80 bit range. Thus the communication, typically in the range of 240–320 bits per party per gate, and the arithmetic constants remain substantial even before accounting for implementation overheads.

Existing PCG/PCF approaches avoid most of this communication, but their known routes to multiparty Beaver correlations currently either (i) rely on setting up pairwise programmable PCF correlations that are extremely slow and involve computing N correlations per party per gate [MMST25], or (ii) require pairwise programmable PCGs that must be refreshed periodically by each pair of parties, yielding both per-party, per-gate communication and computation that scale with N [BCG+19]. Both approaches are currently impractical for the large-party setting.

Our construction is, in some sense, closer to the SPDZ construction in terms of total communication. It can significantly reduce communication while keeping the theoretical computation lower than both strategies above, but the tradeoff in the $N-1$ corruption case is storage. Our current storage is prohibitive in the large-party setting because each party needs to store shares of $s\mathrm{vect}s$, whose size grows quadratically in N. Any improvement in succinctly storing tensor correlations would make our protocol compelling in the $N-1$ corruption case. In Section 8 , we provide concrete estimates for a small value, $N=5$.

Case of $\alpha \cdot N$ Corruptions. We next consider the case of a constant fraction $\alpha \cdot N$ of corruptions, still in the dishonest-majority regime. This is the setting where the PRZS-based extension of Section 6.1 makes our protocol considerably more efficient, since it no longer needs to store seeds whose size grows with N. In this setting, in addition to the SPDZ-type approaches discussed above, the two most relevant state-of-the-art approaches are:

- TinyKeys [HOSS18] gives a variant of GMW using OT extension with short symmetric keys. Instead of targeting only the full dishonest majority setting, it exploits the presence of multiple honest parties: security is based on the concatenation of all honest parties' keys, yielding lower communication as the honest minority grows.
- SuperPack [EGP+23] is primarily a theoretical result aimed at minimizing online communication when a constant fraction of parties are honest. For each multiplication gate, it needs only constant total communication in the case of $\alpha \cdot N$ corruptions.

For a concrete comparison, consider the 1000-party case with 80% corruptions. Compared with TinyKeys, the communication costs are in the same range, roughly 337 kbits per gate in case of TinyKeys versus 250 kbits per gate for our protocol in total, but our computation currently involves

about $90{,}000$ random lookups into roughly 1 TB of RAM, yielding throughput below 100 gates per second. This may be improved by further engineering optimizations, but it remains slower than TinyKeys and even the SPDZ-style approach. Still, it may be competitive with other approaches when communication is the dominant resource.

On the other hand, SuperPack is theoretically better, but only in the online phase. Its online phase is very attractive, but its preprocessing relies on pairwise OLE correlations. For large N, generating and distributing these pairwise OLEs becomes the dominant bottleneck, so the online asymptotics do not directly translate into a practical thousand-party preprocessing protocol.

Case of Honest Majority. In the honest-majority setting, our Shamir-sharing optimization of Section 6.3 substantially reduces both storage and computation. Still, this regime already has highly optimized protocols such as ATLAS [GLO+21], which achieves much better efficiency: for $N = 1000$, ATLAS requires about 20 bits per party per gate, compared with about 56 bits per party per gate for our estimate.

Overall, large-party computation in the dishonest-majority setting remains a longstanding problem. Our scheme does not significantly change this practical landscape by itself, and should primarily be viewed as a theoretical improvement for now. However, there are several directions whose resolution could make our approach much more favorable over other existing methods:

- Can Bernoulli errors be generated efficiently in a distributed manner in the multi-party setting?
- Can tensor correlations be shared more compactly in the N-party setting?
- Are there efficient fault-tolerant and leakage compilers?

3 Preliminaries

Notation. Let $\mathbb{N} = \{1, 2, \dots\}$ be the natural numbers, define $[a, b]$ to be the set $\{a, a+1, \dots, b\}$, and abbreviate $[1, n]$ as $[n]$. Our logarithms are in base 2. We denote the characteristic function as $\mathbb{I}(x)$, which maps a Boolean variable to 1 if true, and 0 otherwise. For a finite set S or distribution $\mathcal{D}$, we write $x \leftarrow S$ or $x \leftarrow \mathcal{D}$ to denote uniformly sampling x from S or $\mathcal{D}$. We denote the security parameter by λ; our parameters depend on λ, e.g. $n = n(\lambda)$, and we often drop the explicit dependence. We abbreviate PPT for probabilistic polynomial-time. Our adversaries are non-uniform PPT ensembles $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$. We write $\mathsf{negl}(\lambda)$ to denote negligible functions in λ. Two ensembles of distributions $\{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\{\mathcal{D}'_\lambda\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable if for any non-uniform PPT adversary $\mathcal{A}$ there exists a negligible function negl such that $\mathcal{A}$ can distinguish between the two distributions with probability at most $\mathsf{negl}(\lambda)$. For $q \in \mathbb{N}$ that is a prime power, we write $\mathbb{F}_q$ to denote the finite field with q elements. We write vectors and matrices in boldface, e.g. $\boldsymbol{v} \in \mathbb{F}^m$ and $\boldsymbol{A} \in \mathbb{F}^{n\times m}$. We denote the Bernoulli distribution over a finite field $\mathbb{F}_q$ with noise rate $\varepsilon \in (0, 1)$ by $\mathsf{Ber}(\mathbb{F}_q, \varepsilon)$; this distribution gives 0 with probability $1-\varepsilon$, and a random non-zero element of $\mathbb{F}_q$ with probability ε. The field parameter will be omitted when working with the binary field.

Communication Cost. Throughout the paper, we work in the setting with private point-to-point channels, where the communication network forms a complete graph (i.e., every pair of parties can communicate directly). No broadcast channel is assumed. The per-party communication cost is measured by counting the sum of bits received and sent by the party. Note that in this model,

aggregation of an additive share can still be done in amortized $O(1)$ cost, by letting one party (rotated in a round-robin manner) aggregate the shares and return the result.

Computational Cost. We adopt the computational model from [BCG+23], which has also been used in several prior works (e.g., [IKOS08, FLY22, HR22]). In this model, protocols are represented as Boolean circuits with bounded fan-in gates. The circuits take as inputs the parties' internal states and incoming messages, and produce as outputs the messages sent to other parties. The computational cost is then defined as the number of gates in these circuits. The per-gate cost is defined as $\sum_{i\in[\text{Rounds}]} |C_i|/|C|$, where C_i denotes the circuit corresponding to the i-th round of the protocol, and C is the original circuit to be evaluated. A reusable preprocessing phase is not included in the per-gate computational cost, provided that it runs in $\mathsf{poly}(\lambda, N)$ time and remains independent of the circuit size $|C|$. Since our protocols are analyzed in the random oracle model, and all inputs to the oracle consist of public information, we assume that the circuits are defined relative to the outputs of the random oracle.

3.1 Learning Parity with Noise

Using the Bernoulli distribution, we now define the core assumption used throughout this paper: the LPN assumption. We begin by introducing the LPN distribution and then formally state the assumption as follows.

Definition 3.1 ($\mathcal{X}$-LPN distribution)**.** *Let $\lambda \in \mathbb{N}$ be the security parameter, $n = n(\lambda)$ be the dimension, $m = m(\lambda) \in \mathbb{N}$ the number of samples, $q = q(\lambda) \in \mathbb{N}$ the field size, and $\varepsilon \in (0,1)$ the noise rate. For distribution $\mathcal{X} = \{\mathcal{X}_\lambda\}$ over $\mathbb{F}_q^n$, we define the $\mathcal{X}$-LPN distribution $\mathcal{D}_{\mathsf{LPN}_{n,m,\varepsilon,q}}(\mathcal{X})$ to be the output distribution of the following process:*

- *Sample $\boldsymbol{s} \leftarrow \mathbb{F}_q^n$ uniformly at random.*
- *Sample $\boldsymbol{a}_i \leftarrow \mathcal{X}$ for all $i \in [m]$.*
- *Sample $e_i \leftarrow \mathsf{Ber}(\mathbb{F}_q, \varepsilon)$ for all $i \in [m]$.*
- *Compute $b_i = \langle \boldsymbol{a}_i, \boldsymbol{s}\rangle + e_i$ for all $i \in [m]$. Output $\{\boldsymbol{a}_i, b_i\}_{i\in[m]}$.*

Similarly, we define $\mathcal{D}^{\mathsf{rand}}_{\mathsf{LPN}_{n,m,\varepsilon,q}}(\mathcal{X})$ to be identical to the distribution $\mathcal{D}_{\mathsf{LPN}_{n,m,\varepsilon,q}}(\mathcal{X})$ except that for all $i \in [m]$, b_i is chosen uniformly at random from $\mathbb{F}_q$. Moreover, when we mention $\boldsymbol{A} \in \mathbb{F}^{n\times m}$ in the context related to LPN, we refer to the matrix obtained by concatenating $\{\boldsymbol{a}_i^T\}_{i\in[m]}$.

Assumption 3.1 (The $(\varepsilon, q, n, m, \mathcal{X})$-$\mathsf{LPN}$ Assumption)**.** *We say that the (ε, q, n, m)-LPN assumption holds if the following two distributions are computationally indistinguishable:*

$$\left\{\mathcal{D}_{\mathsf{LPN}_{n,m,\varepsilon,q}}(\mathcal{X})\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathcal{D}^{\mathsf{rand}}_{\mathsf{LPN}_{n,m,\varepsilon,q}}(\mathcal{X})\right\}_{\lambda\in\mathbb{N}}.$$

Remark 3.1. When $\mathcal{U}$ denotes the uniform distribution over $\mathbb{F}_q^n$, the $\mathcal{U}$–LPN distribution is called the standard LPN distribution. Accordingly, the (ε, q, n, m)–LPN assumption under $\mathcal{U}$ is referred to as the standard LPN assumption. When the distribution $\mathcal{X}$ is omitted, it is understood to mean the standard LPN setting.

LPN is a standard assumption, and there has been a lot of analysis [EKM17, BCGI18] for its security and concrete parameters. For the LPN assumption, hard noise regimes include error rates between $\Omega(1)$ and $\omega\left(\frac{\log n}{n}\right)$, where best known attacks require $2^{O(n\cdot\varepsilon)}$ time. Moreover, if the noise rate is n^{-c} for constant $c < 1$, the LPN assumption can support up to a subexponential number of samples.

3.2 Sparse Learning Parity with Noise

Now we introduce the *sparse learning parity with noise (*sLPN*)* assumption, which is a specific case of the $\mathcal{X}$-LPN assumption. sLPN is a natural variant of the LPN assumption, where each column of the public matrix is k-sparse for a parameter k. It was first introduced by Alekhnovich [Ale03], who used it for obtaining hardness of approximation results; variants of the sLPN assumption were subsequently used for constructing local pseudorandom generators [AIK08], cryptography with constant computational overhead [IKOS08], public-key encryption schemes [ABW10], pseudorandom correlation generators [BCGI18], and more.

Definition 3.2 (Sparse LPN distribution)**.** *Let $\lambda \in \mathbb{N}$ be the security parameter, $n = n(\lambda)$ be the dimension, $m = m(\lambda) \in \mathbb{N}$ the number of samples, $k = k(\lambda) \leq n$ the sparsity parameter, $q = q(\lambda) \in \mathbb{N}$ the field size, and $\varepsilon = \varepsilon(\lambda) \in (0, 1)$ the noise rate. Let $\mathcal{X}$ be the uniform distribution over k-sparse vectors in $\mathbb{F}_q^n$. Then, we define the sparse* LPN *distribution $\mathcal{D}_{\mathsf{sLPN}_{n,m,k,\varepsilon,q}}$ to be $\mathcal{D}_{\mathsf{LPN}_{n,m,\varepsilon,q}}(\mathcal{X})$. Similarly, we define $\mathcal{D}^{\mathsf{rand}}_{\mathsf{sLPN}_{n,m,k,\varepsilon,q}}$ to be $\mathcal{D}^{\mathsf{rand}}_{\mathsf{LPN}_{n,m,\varepsilon,q}}(\mathcal{X})$.*

We now state our Sparse LPN assumption.

Assumption 3.2 (The $(\varepsilon, q, k, n, m)$-sLPN Assumption)**.** *Let $\lambda \in \mathbb{N}$ be the security parameter, $\varepsilon = \varepsilon(\lambda) \in (0, 1)$ be the noise rate, and $k = k(\lambda) \in \mathbb{N}$ be the sparsity parameter. $q = q(\lambda)$ is a sequence of prime powers computable in $\mathsf{poly}(\lambda)$ time, $n = n(\lambda)$ is the dimension, and $m = m(\lambda)$ represents the number of samples. We say that the $(\varepsilon, q, k, n, m)$-*sLPN *assumption holds if the following two distributions are computationally indistinguishable:*

$$\left\{\mathcal{D}_{\mathsf{sLPN}_{n,m,k,\varepsilon,q}}\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathcal{D}^{\mathsf{rand}}_{\mathsf{sLPN}_{n,m,k,\varepsilon,q}}\right\}_{\lambda\in\mathbb{N}}.$$

When using this assumption, we typically choose $k = \omega(1)$ to be a super-constant. Formulations of Sparse LPN are well-studied and believed to be hard over $\mathbb{F}_2$ when $k \geq 3$ is a constant (see for example [Fei02, Ale03, ABW10]). However, Sparse LPN with constant sparsity can only support at most $m = n^{k/2}$ samples, which is a fixed polynomial and does not align with our desire to have an arbitrary polynomial number of samples.

4 Useful Results on Learning Parity with Noise

In this section, we present several results that serve as foundational components of our construction. Beyond their immediate relevance to this work, these results hold significant potential for applications in other contexts.

We start by recalling the definition of LPN distributions shown in Definition 3.1, where $\mathcal{D}_{\mathsf{LPN}}(\mathcal{X})$ describes the distribution:

$$\{\boldsymbol{a}_i, \boldsymbol{b}_i = \langle \boldsymbol{a}_i, \boldsymbol{s}\rangle + e_i\}_{i\in[m]},$$

where $\boldsymbol{s} \leftarrow \mathbb{F}_q^n$ and for all $i \in [m]$, $\boldsymbol{a}_i \leftarrow \mathcal{X}$, $e_i \leftarrow \mathsf{Ber}(\mathbb{F}_q, \varepsilon)$. Similarly, $\mathcal{D}^{\mathsf{rand}}_{\mathsf{LPN}}(\mathcal{X})$ describes a similar distribution except the inner product $\langle \boldsymbol{a}_i, \boldsymbol{s}\rangle$ is replaced by a uniformly random value.

Now, we state an assumption that is equivalent to the $\mathcal{X}$-LPN assumption when the field size is small. For the same parameters, the two-copy LPN distribution $\mathcal{D}_{\mathsf{2LPN}_{n,m,\varepsilon,q}}(\mathcal{X})$ is defined as:

$$\{\boldsymbol{a}_i, \boldsymbol{b}_{1,i} = \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + e_{1,i}, \boldsymbol{b}_{2,i} = \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + e_{2,i}\}_{i \in [m]},$$

where $\boldsymbol{s} \leftarrow \mathbb{F}_q^n$ and for all $i \in [m]$, $\boldsymbol{a}_i \leftarrow \mathcal{X}$, $e_{1,i}, e_{2,i} \leftarrow \mathsf{Ber}(\mathbb{F}_q, \varepsilon)$. Similarly, we let $\mathcal{D}^{\mathsf{rand}}_{\mathsf{2LPN}_{n,m,\varepsilon,q}}(\mathcal{X})$ be identical to the distribution $\mathcal{D}_{\mathsf{2LPN}_{n,m,\varepsilon,q}}$ except that for all $i \in [m]$, we replace the inner product $\langle \boldsymbol{a}_i, \boldsymbol{s} \rangle$ with a value chosen uniformly at random from $\mathbb{F}_q$. The new assumption with parameters (ε, q, n, m) simply states that $\mathcal{D}_{\mathsf{2LPN}_{n,m,\varepsilon,q}}$ is indistinguishable from $\mathcal{D}^{\mathsf{rand}}_{\mathsf{2LPN}_{n,m,\varepsilon,q}}$, which holds under the LPN assumption as shown by the following lemma 4.1.

Lemma 4.1. *Let λ be the security parameter. For modulus $q = 2$, noise rate $\varepsilon = \varepsilon(\lambda)$, dimension $n = n(\lambda) \in \mathsf{poly}(\lambda)$ and number of samples $m = m(\lambda) \in \mathsf{poly}(\lambda)$, assuming that $(\varepsilon', q, n, m, \mathcal{X})$-LPN assumption holds for $\varepsilon' = \frac{\varepsilon^2}{1-2\varepsilon(1-\varepsilon)}$, then we have*

$$\left\{\mathcal{D}_{\mathsf{2LPN}_{n,m,\varepsilon,q}}(\mathcal{X})\right\}_{\lambda \in \mathbb{N}} \approx_c \left\{\mathcal{D}^{\mathsf{rand}}_{\mathsf{2LPN}_{n,m,\varepsilon,q}}(\mathcal{X})\right\}_{\lambda \in \mathbb{N}}.$$

Remark 4.1. This lemma naturally extends to the case of non-constant modulus q, provided that q is bounded by a polynomial in λ, assuming the $(\varepsilon', q, n, m, \mathcal{X})$-LPN assumption holds for $\varepsilon' = O(\varepsilon^2/q)$. However, since our work primarily focuses on the binary field, we present here a shorter and clearer proof specialized to the case $q = 2$.

Proof. To prove the lemma, our goal is to reduce the distinguishing problem between $\mathcal{D}_{\mathsf{2LPN}_{n,m,\varepsilon,q}}(\mathcal{X})$ and $\mathcal{D}^{\mathsf{rand}}_{\mathsf{2LPN}_{n,m,\varepsilon,q}}(\mathcal{X})$ to the distinguishing problem of $\mathcal{X}$-LPN. To demonstrate this reduction, we construct an adversary $\mathcal{A}'$ that can break the corresponding LPN assumption by invoking an adversary $\mathcal{A}$, which distinguishes between the distributions $\mathcal{D}_{\mathsf{2LPN}_{n,m,\varepsilon,q}}(\mathcal{X})$ and $\mathcal{D}^{\mathsf{rand}}_{\mathsf{2LPN}_{n,m,\varepsilon,q}}(\mathcal{X})$.

Specifically, we construct a simulator such that, when the input is sampled from the distribution $\mathcal{D}_{\mathsf{LPN}_{n,m,\varepsilon,q}}(\mathcal{X})$, the output follows the distribution $\mathcal{D}_{\mathsf{2LPN}_{n,m,\varepsilon',q}}(\mathcal{X})$, and when the input is sampled from the distribution $\mathcal{D}^{\mathsf{rand}}_{\mathsf{LPN}_{n,m,\varepsilon,q}}(\mathcal{X})$, the output follows the distribution $\mathcal{D}^{\mathsf{rand}}_{\mathsf{2LPN}_{n,m,\varepsilon',q}}(\mathcal{X})$. With this simulator, the adversary $\mathcal{A}'$ can readily distinguish between the distributions. This is achieved by first invoking the simulator to transform the distribution and subsequently executing the adversary $\mathcal{A}$ on the simulator's output.

The simulator operates as follows:

- Take $\{\boldsymbol{a}_i, b_i\}_{i \in [m]}$ as the input, which is either sampled from $\mathcal{D}_{\mathsf{LPN}_{n,m,\varepsilon',q}}(\mathcal{X})$ or $\mathcal{D}^{\mathsf{rand}}_{\mathsf{LPN}_{n,m,\varepsilon',q}}(\mathcal{X})$.
- For each $i \in [m]$:
  - Sample a bit from the Bernoulli distribution $c_i \leftarrow \mathsf{Ber}(\mathbb{F}_2, 2\varepsilon(1-\varepsilon))$.
  - If $c_i = 1$, set $b_{1,i}$ to be uniformly random, and $b_{2,i} = 1 - b_{1,i}$.
  - If $c_i = 0$, set $b_{1,i} = b_{2,i} = b_i$.
- Output $\{\boldsymbol{a}_i, b_{1,i}, b_{2,i}\}_{i \in [m]}$.

Here, the bit c_i serves as an indicator of whether $b_{1,i}$ and $b_{2,i}$ differ. The event $b_{1,i} \neq b_{2,i}$ occurs with probability $2\varepsilon(1-\varepsilon)$. Conditioned on this event, the symmetry between $b_{1,i}$ and $b_{2,i}$ ensures that $b_{1,i}$ is uniformly distributed over $\mathbb{F}_2$. When instead $b_{1,i} = b_{2,i}$, the distribution $\mathcal{D}_{\mathsf{2LPN}_{n,m,\varepsilon,q}}(\mathcal{X})$ reduces to the $\mathcal{X}$-LPN distribution with noise rate ε'. This completes the proof.

□

Building on this, we can define and analyze the following variant of LPN, termed Quadratic LPN (QLPN), which can be proven secure based on Lemma 4.1. The distribution $\mathcal{D}_{\mathsf{QLPN}_{n,m,\varepsilon,q}}(\mathcal{X})$ is defined as:

$$\mathcal{D}_{\mathsf{QLPN}_{n,m,\varepsilon,q}}(\mathcal{X}) = \\ \{\boldsymbol{a}_i, u_i = \langle \boldsymbol{a}_i, \boldsymbol{s}_1 \rangle + e_{1,i}, v_i = \langle \boldsymbol{a}_i, \boldsymbol{s}_2 \rangle + e_{2,i}, w_i = \langle \boldsymbol{a}_i \otimes \boldsymbol{a}_i, \boldsymbol{s}_1 \otimes \boldsymbol{s}_2 \rangle + e_{3,i} \}_{i \in [m]},$$

where $\boldsymbol{s}_1, \boldsymbol{s}_2 \leftarrow \mathbb{F}_q^n$, and for all $i \in [m]$, $\boldsymbol{a}_i \leftarrow \mathcal{X}$, with Bernoulli noise distribution $e_{1,i}, e_{2,i}, e_{3,i} \leftarrow \mathsf{Ber}(\mathbb{F}_q, \varepsilon)$.

We also define the distribution $\mathcal{D}^{\mathsf{rand}}_{\mathsf{QLPN}_{n,m,\varepsilon,q}}(\mathcal{X})$ which is identical to $\mathcal{D}_{\mathsf{QLPN}_{n,m,\varepsilon,q}}(\mathcal{X})$, except that the terms $\langle \boldsymbol{a}_i, \boldsymbol{s}_1 \rangle$ and $\langle \boldsymbol{a}_i, \boldsymbol{s}_2 \rangle$ are replaced with uniformly random values in $\mathbb{F}_q$.

By using Lemma 4.1, we can show that any polynomial-time adversary cannot distinguish between the distributions $\mathcal{D}_{\mathsf{QLPN}_{n,m,\varepsilon,q}}(\mathcal{X})$ and $\mathcal{D}^{\mathsf{rand}}_{\mathsf{QLPN}_{n,m,\varepsilon,q}}(\mathcal{X})$ for $q = 2$, assuming the LPN assumption with proper parameters. This underscores a critical insight underlying our main result: Even with the introduction of a quadratic term, the security of the LPN assumption is preserved for small field sizes. Specifically, this result allows us to generate pseudorandom noisy triples that are indistinguishable from the tuple $(r_1 + e_1, r_2 + e_2, r_1 r_2 + e_3)$.

However, the reduction used in the security proof introduces a quadratic loss in the noise rate, leading to a much larger dimension n and negatively impacting efficiency. To overcome this issue, we propose an alternative noise distribution for the terms $(e_{1,i}, e_{2,i}, e_{3,i})$. Instead of sampling them as independent random values from a Bernoulli distribution, we let them follow a joint distribution denoted by $\mathcal{D}_e^3$.

Definition 4.1 (Custom noise distribution $\mathcal{D}_e^3$)**.** *The distribution $\mathcal{D}_e^3(\mathbb{F}, \varepsilon)$ that samples three elements in $\mathbb{F}$ is generated by the following process:*

1. *Sample control bits c_1 and c_2 independently from the Bernoulli distribution $\mathsf{Ber}(\mathbb{F}_2, \varepsilon)$.*
2. *Set $e_1 = 0$ if $c_1 = 0$, otherwise sample e_1 uniformly at random from $\mathbb{F}$. Similarly, set $e_2 = 0$ if $c_2 = 0$, otherwise sample e_2 uniformly at random from $\mathbb{F}$.*
3. *Then, set $e_3 = 0$ if both c_1 and c_2 are 0, otherwise sample e_3 uniformly at random from $\mathbb{F}$.*
4. *Output (e_1, e_2, e_3).*

We set $\mathbb{F} = \mathbb{F}_2$ when the field is not explicitly specified. With this noise distribution, we now define the distributions $\mathcal{D}_{\mathsf{QLPN}'_{n,m,\varepsilon,q}}(\mathcal{X})$ and $\mathcal{D}^{\mathsf{rand}\,'}_{\mathsf{QLPN}_{n,m,\varepsilon,q}}(\mathcal{X})$. $\mathcal{D}_{\mathsf{QLPN}'_{n,m,\varepsilon,q}}(\mathcal{X})$ and $\mathcal{D}^{\mathsf{rand}\,'}_{\mathsf{QLPN}_{n,m,\varepsilon,q}}(\mathcal{X})$ are identical to the distributions $\mathcal{D}_{\mathsf{QLPN}_{n,m,\varepsilon,q}}(\mathcal{X})$ and $\mathcal{D}^{\mathsf{rand}}_{\mathsf{QLPN}_{n,m,\varepsilon,q}}(\mathcal{X})$, respectively, except that the noise $(e_{1,i}, e_{2,i}, e_{3,i})$ is drawn from $\mathcal{D}_e^3(\varepsilon)$ for all $i \in [m]$. Under this new noise distribution, Lemma 4.2 shows that the reduction loss from the LPN assumption is mitigated.

Lemma 4.2. *Let λ be the security parameter. For modulus $q = 2$, noise rate $\varepsilon = \varepsilon(\lambda)$, dimension $n = n(\lambda) \in \mathsf{poly}(\lambda)$, distribution $\mathcal{X} = \{\mathcal{X}_\lambda\}_\lambda$ and number of samples $m = m(\lambda) \in \mathsf{poly}(\lambda)$, assuming that the $(\varepsilon', q, n, m, \mathcal{X})$-$\mathsf{LPN}$ assumption holds for $\varepsilon' = \frac{\varepsilon}{4-2\varepsilon}$, then the following holds:*

$$\left\{\mathcal{D}_{\mathsf{QLPN}'_{n,m,\varepsilon,q}}(\mathcal{X})\right\}_{\lambda \in \mathbb{N}} \approx_c \left\{\mathcal{D}^{\mathsf{rand}\,'}_{\mathsf{QLPN}_{n,m,\varepsilon,q}}(\mathcal{X})\right\}_{\lambda \in \mathbb{N}}.$$

Corollary 4.1. *Let λ be the security parameter. For modulus $q = 2$, noise rate $\varepsilon = \varepsilon(\lambda)$, dimension $n = n(\lambda) \in \mathsf{poly}(\lambda)$, sparsity parameter $k = k(\lambda)$, and number of samples $m = m(\lambda) \in \mathsf{poly}(\lambda)$, let*

*$\mathcal{X}$ denote the uniform distribution over k-sparse vectors in $\mathbb{F}_2^n$, assuming that the $(\varepsilon', q, n, m,)$-*sLPN *assumption holds for $\varepsilon' = \frac{\varepsilon}{4-2\varepsilon}$, then the following holds:*

$$\left\{\mathcal{D}_{\mathsf{QLPN}'_{n,m,\varepsilon,q}}(\mathcal{X})\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathcal{D}^{\mathsf{rand}\ '}_{\mathsf{QLPN}_{n,m,\varepsilon,q}}(\mathcal{X})\right\}_{\lambda\in\mathbb{N}}.$$

Proof. To prove this lemma, we argue by contradiction. That is, assuming the existence of an adversary that distinguishes the distributions $\mathcal{D}_{\mathsf{QLPN}'_{n,m,\varepsilon,q}}$ and $\mathcal{D}^{\mathsf{rand}\ '}_{\mathsf{QLPN}_{n,m,\varepsilon,q}}$, we are able to construct an adversary for the LPN assumption. We conclude this proof by establishing the following three claims.

Claim 4.1. *There exists a hybrid distribution $\mathcal{D}_{\mathsf{Hyb}}$ such that if an adversary $\mathcal{A}$ can distinguish the hybrid distribution from either $\mathcal{D}_{\mathsf{QLPN}'_{n,m,\varepsilon,q}}(\mathcal{X})$ or $\mathcal{D}^{\mathsf{rand}\ '}_{\mathsf{QLPN}_{n,m,\varepsilon,q}}(\mathcal{X})$, then for a specific distribution $\mathcal{B}$, there exists a polynomial-time adversary $\mathcal{A}'$ that can distinguish the following two distributions by invoking $\mathcal{A}$:*

$$\mathcal{D}_1(\mathcal{B}) = \{\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s}_1\rangle + e_{1,i}, b_i + e_{2,i}, \langle \boldsymbol{a}_i, \boldsymbol{s}_1\rangle \cdot b_i + e_{3,i}\}_{i\in[m]},$$
$$\mathcal{D}_2(\mathcal{B}) = \{\boldsymbol{a}_i, r_{1,i} + e_{1,i}, b_i + e_{2,i}, r_{1,i} \cdot b_i + e_{3,i}\}_{i\in[m]},$$

where $\boldsymbol{s}_1 \leftarrow \mathbb{F}_2^n$, for $i \in [m]$, $\boldsymbol{a}_i \leftarrow \mathcal{X}, (e_{1,i}, e_{2,i}, e_{3,i}) \leftarrow \mathcal{D}_e^3(\varepsilon), r_{1,i} \leftarrow \mathbb{F}_2$, $\mathcal{B}$ is a publicly known distribution of $\{b_i\}_{i\in[m]}$ and b_i may depend on $\boldsymbol{a}_i$.

Claim 4.2. *Let $\mathcal{B}$ be any distribution over $\{b_i\}_{i\in[m]}$ that can be efficiently sampled, where b_i may depend on $\boldsymbol{a}_i$. Let $\mathcal{D}_e^2(\varepsilon)$ be the distribution generated by the following process:*

1. *Sample a control bit c from the Bernoulli distribution $\mathsf{Ber}(\mathbb{F}_2, \varepsilon)$.*
2. *Set e_1 and e_2 to zero if $c = 0$, otherwise sample e_1 and e_2 uniformly at random in $\mathbb{F}$.*
3. *Output (e_1, e_2).*

If an adversary $\mathcal{A}$ can distinguish distributions $\mathcal{D}_1(\mathcal{B})$ and $\mathcal{D}_2(\mathcal{B})$, then there exists a polynomial-time adversary $\mathcal{A}'$ that can distinguish the following distributions by invoking $\mathcal{A}$:

$$\mathcal{D}_3 = \left\{\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s}\rangle + e'_{1,i}, e'_{2,i}\right\}_{i\in[m]}, \quad \mathcal{D}_4 = \left\{\boldsymbol{a}_i, r_i + e'_{1,i}, e'_{2,i}\right\}_{i\in[m]},$$

where $(e'_{1,i}, e'_{2,i})$ is sampled from the distribution $\mathcal{D}_e^2(\varepsilon)$.

Claim 4.3. *If an adversary $\mathcal{A}$ can distinguish $\mathcal{D}_3$ and $\mathcal{D}_4$, then there exists a polynomial-time adversary $\mathcal{A}'$ that can distinguish the LPN distributions $\mathcal{D}_{\mathsf{LPN}_{\varepsilon',q,n,m,\mathcal{X}}}$ and $\mathcal{D}^{\mathsf{rand}}_{\mathsf{LPN}_{\varepsilon',q,n,m,\mathcal{X}}}$ by invoking $\mathcal{A}$.*

It is straightforward to observe that by combining the claims we can establish the proof of this lemma.

□

Proof of Claim 4.1. We consider the sequence of hybrids:

- Hyb_0 : This hybrid is identical to distribution $\mathcal{D}_{\mathsf{QLPN}'_{n,m,\varepsilon,q}}(\mathcal{X})$.
- Hyb_1 : This hybrid is the distribution

$$\mathcal{D}_{\mathsf{Hyb}} = \{\boldsymbol{a}_i, r_i + e_{1,i}, \langle \boldsymbol{a}_i, \boldsymbol{s}_2\rangle + e_{2,i}, \langle \boldsymbol{a}_i, \boldsymbol{s}_2\rangle \cdot r_i + e_{3,i}\}_{i\in[m]},$$

  where $\boldsymbol{a}_i, \boldsymbol{s}_2, r_i$ are sampled uniformly at random, and $(e_{1,i}, e_{2,i}, e_{3,i})$ is sampled from $\mathcal{D}_e^3(\varepsilon)$.

- Hyb_2 : This hybrid is identical to distribution $\mathcal{D}^{\mathsf{rand}\,\prime}_{\mathsf{QLPN}_{n,m,\varepsilon,q}}(\mathcal{X})$.

If the theorem does not hold, then an adversary can distinguish Hyb_0 from Hyb_2, and it must also be able to distinguish either between Hyb_0 and Hyb_1, or between Hyb_1 and Hyb_2.

- If the adversary can distinguish Hyb_0 and Hyb_1, we complete the proof of Claim 4.1 by specifying the distribution $\mathcal{B}$ over $\{b_i\}$ to be the same as that of $\{\langle \boldsymbol{a}_i, \boldsymbol{s}\rangle\}$, where $\boldsymbol{s}$ is sampled uniformly in $\mathbb{F}_2^n$. It is straightforward to verify that by replacing b_i with $\langle \boldsymbol{a}_i, \boldsymbol{s}\rangle$, the distribution $\mathcal{D}_1(\mathcal{B})$ becomes identical to Hyb_0, while $\mathcal{D}_2(\mathcal{B})$ matches Hyb_1.
- If the adversary can distinguish between Hyb_1 and Hyb_2, then due to the symmetry between $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, an adversary can also distinguish the following distribution from Hyb_2:

  $$\mathcal{D}'_{\mathsf{Hyb}} = \{\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s}_1\rangle + e_{1,i}, r_i + e_{2,i}, \langle \boldsymbol{a}_i, \boldsymbol{s}_1\rangle \cdot r_i + e_{3,i}\}_{i\in[m]}\,.$$

  Thus, we can complete the proof by specifying the distribution $\mathcal{B}$ over $\{b_i\}$ to be the same as that of $\{r_i\}$.

□

Proof of Claim 4.2. To prove this claim, we construct a simulator such that the output distribution of this simulator is identical to $\mathcal{D}_1(\mathcal{B})$ when it receives inputs from the distribution $\mathcal{D}_3$, and the output distribution matches $\mathcal{D}_2(\mathcal{B})$ when given inputs sampled from the distribution $\mathcal{D}_4$. The simulator works as follows:

- Take as input a sample $\left\{\boldsymbol{a}_i, b'_i, e'_{2,i}\right\}_{i\in[m]}$ either from $\mathcal{D}_3$ or $\mathcal{D}_4$.
- Sample $\{b_i\}_{i\in[m]}$ from $\mathcal{B}$ with respect to $\{\boldsymbol{a}_i\}_{i\in[m]}$.
- For each $i \in [m]$:
  - Sample a control bit $c_{2,i}$ from $\mathsf{Ber}(\mathbb{F}_2, \varepsilon)$.
  - If $c_{2,i} = 1$, set the i-th entry of the output to $(\boldsymbol{a}_i, b'_i, r_{1,i}, r_{2,i})$, where $r_{1,i}$ and $r_{2,i}$ are sampled uniformly at random.
  - If $c_{2,i} = 0$, set the i-th entry of the output to $(\boldsymbol{a}_i, b'_i, b_i, b'_i \cdot b_i + e'_{2,i})$. Namely, if $b_i = 1$, the entry is $(\boldsymbol{a}_i, b'_i, 1, b'_i + e'_{2,i})$, and if $b_i = 0$, the entry is $(\boldsymbol{a}_i, b'_i, 0, e'_{2,i})$.

We now justify why the simulator works. In the simulator, the bit $c_{2,i}$ represents the second control bit in the distribution $\mathcal{D}_e^3$ used by the i-th entry, either in $\mathcal{D}_1(\mathcal{B})$ or in $\mathcal{D}_2(\mathcal{B})$ that we aim to simulate. We treat the noise $e'_{2,i}$ as $e_{3,i}$ in these distributions. To argue the correctness of the simulator, we focus on showing that the output of this simulator follows $\mathcal{D}_1(\mathcal{B})$ when given inputs from $\mathcal{D}_3$. A similar argument applies for $\mathcal{D}_2(\mathcal{B})$ and $\mathcal{D}_4$. We consider the following three cases:

- **Case $c_{2,i} = 1$:** In this case, the noise terms $e_{2,i}$ and $e_{3,i}$ are uniformly random and mask all information about the last two terms. Thus, the simulator sets these terms to be uniformly random. The correctness follows from the fact that the second terms of $\mathcal{D}_1(\mathcal{B})$ and $\mathcal{D}_3$ follow identical distributions.
- **Case $c_{2,i} = 0$ and $b_i = 0$:** Here, since $c_{2,i} = 0$, both $e_{1,i}$ and $e_{3,i}$ are determined by the control bit $c_{1,i}$. The conditional distribution of $(e_{1,i}, e_{3,i})$ is precisely $\mathcal{D}_e^2(\varepsilon)$. When the simulator receives inputs from $\mathcal{D}_3$, it generates $(\boldsymbol{a}_i, b'_i, \langle \boldsymbol{a}_i, \boldsymbol{s}\rangle + e'_{1,i}, 0, e'_{2,i})$ for $(e'_{1,i}, e'_{2,i}) \sim \mathcal{D}_e^2(\varepsilon)$, which matches the desired distribution.

- **Case $c_{2,i} = 0$ and $b_i = 1$:** This case is similar to the previous one, with the only difference being in the last term. The simulator outputs $(\boldsymbol{a}_i, b'_i, \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + e'_{1,i}, 1, \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + e'_{1,i} + e'_{2,i})$ for $(e'_{1,i}, e'_{2,i}) \sim \mathcal{D}_e^2(\varepsilon)$, given inputs from $\mathcal{D}_3$. Notably, the pair $(e'_{1,i}, e'_{1,i} + e'_{2,i})$ also follows the distribution $\mathcal{D}_e^2(\varepsilon)$, confirming the correctness.

Similar to the proof of Lemma 4.1, we can directly construct an adversary adv' based on the adversary $\mathcal{A}$ by first invoking the simulator, and then executing $\mathcal{A}$ on the simulator's output, which completes the proof.

□

Proof of Claim 4.3. For the last claim, we once again adopt the proof strategy of Lemma 4.1. This involves constructing another simulator such that the output distribution of this simulator is identical to $\mathcal{D}_3$ when it receives inputs sampled from the distribution $\mathcal{D}_{\mathsf{LPN}_{n,m,\varepsilon',q}}$, and matches $\mathcal{D}_4$ when given inputs sampled from the distribution $\mathcal{D}^{\mathsf{rand}}_{\mathsf{LPN}_{n,m,\varepsilon',q}}$. The simulator is outlined as follows:

- Take $\{\boldsymbol{a}_i, b_i\}_{i\in[m]}$ as input, where the input is drawn from either $\mathcal{D}_{\mathsf{LPN}_{n,m,\varepsilon',q}}$ or $\mathcal{D}^{\mathsf{rand}}_{\mathsf{LPN}_{n,m,\varepsilon',q}}$.
- For each $i \in [m]$:
  - Sample $e'_i \leftarrow \mathsf{Ber}(\varepsilon/2)$.
  - If $e'_i = 1$, set b'_i to be uniformly random.
  - If $e'_i = 0$, set $b'_i = b_i$.
- Output $\{\boldsymbol{a}_i, b'_i, e'_i\}_{i\in[m]}$.

The correctness of this simulator is derived from analyzing the marginal distribution of the error terms. In this simulator, the "leaked noise" e'_i is sampled directly. It is straightforward to see that $e'_i = 1$ occurs with probability $\varepsilon/2$, at which point b'_i should be uniformly random over $\mathbb{F}_2$. Conversely, when $e'_i = 0$, the distribution conforms to the $\mathcal{X}$-LPN distribution with noise rate ε'. Thus, the simulator accurately simulates the distribution we want, and this construction implies that $\mathcal{D}_3 \approx_c \mathcal{D}_4$ under the assumption that the $(\varepsilon', 2, n, m, \mathcal{X})$-LPN assumption holds. □

5 Noisy Pseudocorrelation Function for Beaver Triples

With tools from Section 4 prepared, we now give the construction of a noisy PCF for Beaver triples. In this section, we mainly focus on the binary field $\mathbb{F}_2$.

A Beaver triple is a tuple (u, v, w), where u and v are randomly chosen values in a given field ($\mathbb{F}_2$ in this section), and w is their product, $w = u \cdot v$. Beaver triples are a fundamental resource for secure multiparty computation: given shares of (u, v, w), the parties can multiply arbitrary secret inputs by reducing the multiplication to linear operations combined with one use of the triple. This allows secure multiplication without revealing the individual operands, while keeping the communication complexity low. In the setting of correlation generation, we require these triples to be pre-shared among the parties in such a way that they remain indistinguishable from truly random correlations. Later on, we use $\mathcal{D}_{\mathsf{Beaver}}(\mathbb{F})$ to represent the truly random distribution of Beaver triples, which samples u, v uniformly from $\mathbb{F}$ and sets $w = u \cdot v$.

The notion of pseudorandom correlation function, denoted by PCF, was proposed in [BCG+20a]. We now give the tweaked definition for noisy PCF (denoted by nPCF) that supports correlations among more than two parties.

Definition 5.1 (Noisy Pseudorandom Correlation Function for Beaver Triples)**.** *A noisy PCF scheme for Beaver triples consists of the following algorithms:*

- $\mathsf{nPCF.Gen}(1^\lambda, 1^N, t \in [0, N-1], \epsilon)$ *is a probabilistic polynomial-time algorithm that takes the security parameter* 1^λ*, the number of parties* 1^N*, an upper bound on corrupted parties* t*, and noise rate* ϵ *as input, and outputs* N *evaluation keys* $k_1, \cdots, k_N$*.*
- $\mathsf{nPCF.Eval}(K_i, x)$ *is a deterministic polynomial-time algorithm that takes the evaluation key* K_i *assigned to party* i *and input value* x*, and outputs* $y_i = (u_i, v_i, w_i) \in \mathbb{F}^3$*, the portion of the correlation held by party* i*.*

Let N, t *and* ϵ *be functions of* λ*, and let* $\mathcal{X} = \{\mathcal{X}_\lambda\}$ *describe a family of distributions parameterized by* λ*. We say that* $(\mathsf{nPCF.Gen}, \mathsf{nPCF.Eval})$ *is a* (N, t)*-noisy pseudorandom correlation function for Beaver triples in* $\mathbb{F}$ *with respect to input distribution* $\mathcal{X}$ *with noise rate* ϵ *if the following condition holds:*

- (N, t)-Weak pseudorandomness with respect to input distribution $\mathcal{X}$*. There exists a PPT noise indicator function* $\mathbb{I}_A : \{0,1\}^* \to \mathbb{F}^3$ *associated with the* nPCF *scheme, such that the following holds: For any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for any* $\lambda \in \mathbb{N}$*,* $N = N(\lambda)$ *and* $t = t(\lambda)$*, the adversary* $\mathcal{A}$*'s advantage in experiment* $\mathsf{Exp}^{\mathsf{nPCF}}_{\mathcal{A},N,t,\mathcal{X}_\lambda}(\lambda)$ *(shown in Figure 1) is bounded by* $\mathsf{negl}(\lambda)$*.*
- ϵ-Error Probability*. For the noise indicator* $\mathbb{I}_A$ *associated with the* nPCF *scheme, there exists a negligible function* negl *such that, for any* $\lambda \in \mathbb{N}$ *and set of corrupted parties* $T \subseteq [N(\lambda)]$ *with* $|T| = t(\lambda)$*, the following holds with probability at least* $1 - \mathsf{negl}(\lambda)$ *over the choice of keys* $\{K_i\}_{i\in[N]} \leftarrow \mathsf{nPCF.Gen}(1^\lambda, 1^N, t, \epsilon)$*: the noise rate indicated by the noise indicator is bounded by* ϵ*, i.e.,*

$$\Pr\left[\mathbb{I}_A(T, \{K_i\}_{i\in T}, x) \neq \mathbf{0} \mid x \leftarrow \mathcal{X}_\lambda\right] < \epsilon(\lambda) + \mathsf{negl}(\lambda).$$

Remark 5.1. We adapt the notion of a noisy PCF by requiring the simulator $\mathsf{Sim}_{\mathsf{PCF}}$ to have a specific form that contains a noise indicator. For a standard PCF, the noise indicator always outputs $\mathbf{0}$. For a noise indicator satisfying the *ϵ-Error Probability*, the condition of *(N, t)-Weak pseudorandomness* ensures that the Beaver triples are simulatable from the adversary's perspective, while the results of the nPCF evaluation remain consistent with the Beaver triple correlation in most cases (when the noise indicator outputs $\mathbf{0}$). The condition should occur with probability $1 - \epsilon$ by the *ϵ-Error Probability* property.

5.1 Construction of Noisy PCF

We now provide our construction of a nPCF scheme based on the sparse LPN assumption, as defined in Assumption 3.2. We then provide the security proof and analyze the computational efficiency.

Ingredients, Notations and Parameter Settings. The construction makes use of the following primitives:

- λ is the security parameter, $\mathbb{F}$ is the finite field, N is the number of parties, and t is the corruption threshold. N should be a polynomial in λ, t is set to be $N - 1$, and $\mathbb{F}$ is typically $\mathbb{F}_2$.

Experiment $\mathsf{Exp}^{\mathsf{nPCF}}_{\mathcal{A},N,t,\mathcal{X}}(\lambda)$

- $\mathcal{A}(1^\lambda, 1^N, t)$ chooses a set of corrupted parties $T \subseteq [N]$ of size t.
- The challenger samples $b \leftarrow \{0,1\}$, then generates the keys $\{K_i\}_{i\in[N]} \leftarrow \mathsf{nPCF.Gen}(1^\lambda, 1^N, t, \epsilon)$ and sends the keys of corrupted parties $\{K_i\}_{i\in T}$ to $\mathcal{A}$.
- $\mathcal{A}$ now makes arbitrary oracle queries. For each query, the challenger replies as follows:
  - Sample input value x uniformly at random from distribution $\mathcal{X}$.
  - Compute $\boldsymbol{y}_{i,0} \leftarrow \mathsf{nPCF.Eval}(K_i, x)$ for all $i \in T$.
  - If $b = 0$, compute $\boldsymbol{y}_{i,0} \leftarrow \mathsf{nPCF.Eval}(K_i, x)$ for all $i \notin T$.
  - If $b = 1$, sample $\boldsymbol{y}_{i,1}$ for $i \notin T$ with the following simulator $\mathsf{Sim}_{\mathsf{PCF}}$: $\mathsf{Sim}_{\mathsf{PCF}}$ first evaluates the noise indicator $\mathbb{I}_A(T, \{K_i\}_{i\in T}, x)$ to obtain a noise term $\boldsymbol{e} = (e_1, e_2, e_3) \in \mathbb{F}^3$. Then, $\mathsf{Sim}_{\mathsf{PCF}}$ samples a Beaver triple $\boldsymbol{y} = (u, v, w) \leftarrow \mathcal{D}_{\mathsf{Beaver}}(\mathbb{F})$, and samples shares of honest parties $\{\boldsymbol{y}_{i,1}\}_{i\notin T}$ uniformly at random, conditioned on $\sum_{i\in T} \boldsymbol{y}_{i,0} + \sum_{i\notin T} \boldsymbol{y}_{i,1} = \boldsymbol{y} + \boldsymbol{e}$.
  - Reply with x and $\{\boldsymbol{y}_{i,b}\}_{i\notin T}$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs a bit b'. The advantage of $\mathcal{A}$ is $|2 \times \Pr[b' = b] - 1|$.

Figure 1: Experiment in (N, t)-Weak pseudorandomness definition

- n, k, ε correspond to the dimension, the sparsity parameter, and the noise rate of sparse LPN. In the scheme, n, k should be polynomials in λ, and ε is an inverse polynomial in λ.
- PRF is a pseudorandom function from $\{0,1\}^\lambda \to \{0,1\}^\lambda$.
- ϵ is the noise rate of noisy PCF. In the scheme, we will set ϵ to satisfy $\epsilon \geq 1 - \left(1 - \frac{4\varepsilon}{1+2\varepsilon}\right)^{2N}$.
- $\mathcal{X}$ is the input distribution of the noisy PCF that samples a vector $\boldsymbol{a}$ uniformly from k-sparse vectors in $\mathbb{F}_2^n$ and a bit string r uniformly from $\{0,1\}^\lambda$.

The construction of the noisy pseudorandom correlation function is shown in Figure 2, where we omit the details of sampling from $\mathcal{D}^3_e(\mathbb{F}, \eta)$ for clarity. Recall that this distribution reduces to Bernoulli sampling (Definition 4.1). For global noise rate ϵ and N parties, the local rate is $\eta = \epsilon/N$, so naive sampling would require $O(\log N - \log \epsilon)$ random bits per term. To remove the N-dependence, we instead generate error terms in blocks of λ terms. For each block, we first sample the number of non-zero entries; by a Chernoff bound, this is at most $O(\log \lambda)$ with probability $1 - 2^{-\Omega((\log\lambda)^2)}$, allowing larger outcomes to be ignored. Specifying the x error positions then costs $\log \lambda$ bits each. Overall, the procedure uses only $\mathsf{poly}\log(\lambda)$ random bits for λ terms, and its runtime is independent of N.

Proof of Security. We now provide the proof of the following theorem.

Theorem 5.1. *Consider the parameter settings described in paragraph 5.1. Assume that the $(\varepsilon, 2, k, n, m)$-sparse LPN assumption holds for all polynomials $m(\lambda) \in \mathsf{poly}(\lambda)$. Then, the scheme shown in Figure 2 is a (N, t)-noisy pseudorandom correlation function for Beaver triples in $\mathbb{F}_2$ with respect to input distribution $\mathcal{X}$*

Noisy PCF for Beaver triples

- $\mathsf{nPCF.Gen}(1^\lambda, 1^N, t, \epsilon) \to \{K_i\}_{i\in[N]}$. On input 1^λ, 1^N, t, and ϵ,
  - Sample two secret vectors $\boldsymbol{s}_1 \leftarrow \mathbb{F}^n, \boldsymbol{s}_2 \leftarrow \mathbb{F}^n$ uniformly at random and generate the additive shares of $\boldsymbol{s}_1, \boldsymbol{s}_2$ and $\boldsymbol{s}_1 \otimes \boldsymbol{s}_2$. Let the shares be denoted by $[\![\boldsymbol{s}_1]\!], [\![\boldsymbol{s}_2]\!]$ and $[\![\boldsymbol{s}_1 \otimes \boldsymbol{s}_2]\!]$.
  - Generate PRF keys $\mathsf{PRF}.K_i \leftarrow \mathsf{PRF.Gen}(1^\lambda)$ for $i \in [N]$.
  - Let $K_i = (\mathsf{PRF}.K_i, [\![\boldsymbol{s}_1]\!]_i, [\![\boldsymbol{s}_2]\!]_i, [\![\boldsymbol{s}_1 \otimes \boldsymbol{s}_2]\!]_i)$ and output $\{K_i\}_{i\in[N]}$.
- $\mathsf{nPCF.Eval}(K_i, x) \to y_i \in \mathbb{F}^3$. On input K_i, x,
  - Parse K_i as $(\mathsf{PRF}.K_i, [\![\boldsymbol{s}_1]\!]_i, [\![\boldsymbol{s}_2]\!]_i, [\![\boldsymbol{s}_1 \otimes \boldsymbol{s}_2]\!]_i)$.
  - Interpret input x as a vector $\boldsymbol{a} \in \mathbb{F}^n$ and a bit string $r \in \{0,1\}^\lambda$.
  - Evaluate PRF to obtain $r_i \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K_i, r)$.
  - Sample noise terms $(e_{i,1}, e_{i,2}, e_{i,3}) \leftarrow \mathcal{D}_e^3(\mathbb{F}, \eta)$ using randomness r_i, where η satisfies $(1-\eta)^{2N} \geq 1-\epsilon$ and $\mathcal{D}_e^3(\mathbb{F}, \eta)$ is the distribution in Definition 4.1.
  - Compute $u_i = \langle \boldsymbol{a}, [\![\boldsymbol{s}_1]\!]_i \rangle + e_{i,1}, v_i = \langle \boldsymbol{a}, [\![\boldsymbol{s}_2]\!]_i \rangle + e_{i,2}, w_i = \langle \boldsymbol{a} \otimes \boldsymbol{a}, [\![\boldsymbol{s}_1 \otimes \boldsymbol{s}_2]\!]_i \rangle + e_{i,3}$.
  - Output $\boldsymbol{y}_i = (u_i, v_i, w_i)$.
- Noise indicator $\mathbb{I}_A(T, \{K_i\}_{i\in T}, x) \to e \in \mathbb{F}^3$,
  - For $i \in T$, parse K_i as $(\mathsf{PRF}.K_i, [\![\boldsymbol{s}_1]\!]_i, [\![\boldsymbol{s}_2]\!]_i, [\![\boldsymbol{s}_1 \otimes \boldsymbol{s}_2]\!]_i)$.
  - Parse input x as a vector $\boldsymbol{a} \in \mathbb{F}^n$ and a bit string $r \in \{0,1\}^\lambda$.
  - For $i \in T$, evaluate $r_i \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K_i, r)$.
  - For $i \in T$, sample noise terms $(e_{i,1}, e_{i,2}, e_{i,3}) \leftarrow \mathcal{D}_e^3(\mathbb{F}, \eta)$ using randomness r_i.
  - For $i \notin T$, sample noise terms $(e_{i,1}, e_{i,2}, e_{i,3}) \leftarrow \mathcal{D}_e^3(\mathbb{F}, \eta)$ with fresh randomness.
  - Compute $e_1 = \sum_{i\in[N]} e_{i,1}, e_2 = \sum_{i\in[N]} e_{i,2}, e_3 = \sum_{i\in[N]} e_{i,3}$.
  - Output $e = (e_1, e_2, e_3)$.

Figure 2: Construction of PCF for noisy Beaver triples

with noise rate ϵ. Moreover, each nPCF *evaluation requires $O(k^2)$ computational steps, along with a* PRF *evaluation.*

Remark 5.2. In the construction of Figure 2, we employ PRF to ensure that the Eval function is deterministic on its input. However, when using this nPCF in semi-honest MPC protocols, each party could instead generate the noise with fresh local randomness, as the determinism via PRF is not required for security in this setting, but only serves to simplify the formal definition.

Remark 5.3. When analyzing the computational cost of nPCF.Eval, we count only its preprocessed input complexity by treating it as a circuit defined with respect to the input x, rather than as a circuit that explicitly takes x as input. This perspective aligns better with our computational model, since later we will incorporate public randomness as inputs.

Proof. We begin by considering the error probability determined by the noise indicator $\mathbb{I}_A$, where the noise term is in the form of the summation of $\{(e_{i,1}, e_{i,2}, e_{i,3})\}_{i\in[N]}$. By the pseudorandomness

property of PRF, the triple $(e_{i,1}, e_{i,2}, e_{i,3})$ sampled using randomness from PRF should be computationally indistinguishable from one sampled directly from the distribution $\mathcal{D}_e^3(\eta)$; thus, the error probability is indistinguishable from the case where all triples $(e_{i,1}, e_{i,2}, e_{i,3})$ follow the distribution $\mathcal{D}_e^3(\eta)$. In this case, each tuple $(e_{i,1}, e_{i,2}, e_{i,3})$ is zero with probability at least $(1-\eta)^2$, and by the choice of η according to the construction, all noise terms are zero with probability at least $1-\epsilon$. This establishes the desired ϵ-*error probability*.

To prove that the scheme satisfies (N, t)-*Weak pseudorandomness*, as defined in Definition 5.1, we consider the case where the adversary corrupts $t = N - 1$ parties. The proof proceeds via a sequence of hybrid experiments:

- Hyb_0: This hybrid corresponds to the experiment $\mathsf{Exp}^{\mathsf{nPCF}}_{\mathcal{A},N,t,\mathcal{X}}(\lambda)$ with $b = 0$.
- Hyb_1: This hybrid is identical to Hyb_0, except that when computing $y_{i,0} \leftarrow \mathsf{nPCF.Eval}(K_i, x)$ for the honest party, the noise vector $\boldsymbol{e}_i$ is generated using fresh randomness, rather than derived from the PRF evaluation randomness r_i.

  Since the PRF key K_i is hidden from the adversary, and the probability that the adversary queries the same PRF input is negligible, the outputs remain pseudorandom. Hence, by the pseudorandomness property of the PRF, the adversary cannot distinguish Hyb_0 from Hyb_1.
- Hyb_2: This hybrid is identical to Hyb_1, except that the oracle query responses are reformulated as follows:
  - Let $u = \langle \boldsymbol{a}, \boldsymbol{s}_1 \rangle$, $v = \langle \boldsymbol{a}, \boldsymbol{s}_2 \rangle$, and $w = \langle \boldsymbol{a} \otimes \boldsymbol{a}, \boldsymbol{s}_1 \otimes \boldsymbol{s}_2 \rangle$. Sample (e_1, e_2, e_3) independently from $\mathcal{D}_e^3(\eta)$.
  - For the honest party i, define

$$u_i = u - \sum_{i' \in T} \langle \boldsymbol{a}, [\![\boldsymbol{s}_1]\!]_{i'} \rangle + e_1, \quad v_i = v - \sum_{i' \in T} \langle \boldsymbol{a}, [\![\boldsymbol{s}_2]\!]_{i'} \rangle + e_2,$$

$$w_i = w - \sum_{i' \in T} \langle \boldsymbol{a} \otimes \boldsymbol{a}, [\![\boldsymbol{s}_1 \otimes \boldsymbol{s}_2]\!]_{i'} \rangle + e_3.$$

  - Reply to the adversary with $\boldsymbol{y}_{i,0} = (u_i, v_i, w_i)$.

  This change is purely representational. Thus, Hyb_2 is identical to Hyb_1.
- Hyb_3: This hybrid is identical to Hyb_2, except that (u, v, w) in oracle responses is now sampled directly from $\mathcal{D}_{\mathsf{Beaver}}(\mathbb{F}_2)$.

  If the adversary makes m oracle queries, then distinguishing between Hyb_2 and Hyb_3 reduces to distinguishing between the following two distributions:

$$\left\{ \boldsymbol{a}^j,\ \langle \boldsymbol{a}^j, \boldsymbol{s}_1 \rangle + e_1^j,\ \langle \boldsymbol{a}^j, \boldsymbol{s}_2 \rangle + e_2^j,\ \langle \boldsymbol{a}^j \otimes \boldsymbol{a}^j, \boldsymbol{s}_1 \otimes \boldsymbol{s}_2 \rangle + e_3^j \right\}_{j \in [m]}$$
$$\approx_c \left\{ \boldsymbol{a}^j,\ u^j + e_1^j,\ v^j + e_2^j,\ w^j + e_3^j \right\}_{j \in [m]},$$

  where $\boldsymbol{s}_1, \boldsymbol{s}_2$ are uniformly random vectors, $\boldsymbol{a}^j$ is sampled from the set of k-sparse vectors, (e_1^j, e_2^j, e_3^j) are drawn from $\mathcal{D}_e^3(\eta)$, and (u^j, v^j, w^j) are sampled from $\mathcal{D}_{\mathsf{Beaver}}(\mathbb{F}_2)$.

By Lemma 4.2, this indistinguishability holds whenever $\frac{\eta}{4-2\eta} \geq \varepsilon$ under the $(\varepsilon, 2, k, n, m)$-sLPN assumption. In fact, the left-hand side matches $\mathcal{D}'_{\mathsf{QLPN},n,m,\eta,2}$, and the lemma reduces the claim to proving

$$\mathcal{D}^{\mathsf{rand}}_{\mathsf{QLPN}_{n,m,\eta,2}} \approx_c \left\{ \boldsymbol{a}^j,\ u^j + e_1^j,\ v^j + e_2^j,\ w^j + e_3^j \right\}_{j\in[m]},$$

where $(u^j, v^j, w^j) \leftarrow \mathcal{D}_{\mathsf{Beaver}}(\mathbb{F}_2)$ and $(e_1^j, e_2^j, e_3^j) \leftarrow \mathcal{D}_e^3(\eta)$.

By the definition of $\mathcal{D}^{\mathsf{rand}}_{\mathsf{QLPN}_{n,m,\eta,2}}$, these two distributions are identical. Moreover, the condition $\frac{\eta}{4-2\eta} \geq \varepsilon$ is always satisfied whenever $\epsilon \geq 1 - \left(1 - \frac{4\varepsilon}{1+2\varepsilon}\right)^{2N}$, and we assume the sparse LPN assumption for all polynomials $m \in \mathsf{poly}(\lambda)$. Hence, we conclude that $\mathsf{Hyb}_2 \approx_c \mathsf{Hyb}_3$.

- Hyb_4: This hybrid corresponds to the experiment $\mathsf{Exp}^{\mathsf{nPCF}}_{\mathcal{A},N,t,\mathcal{X}}(\lambda)$ with $b = 1$.

  Compared with Hyb_3, this is again only a representational change. Thus, Hyb_3 and Hyb_4 are identical.

With these hybrids, we complete the proof of this theorem.

□

Remark 5.4. In this construction of nPCF, each evaluation requires $O(k^2)$ time plus a single PRF call, and importantly, this cost is independent of the number of parties. Our design does not exploit any special structure of sparse LPN samples. In fact, a construction based solely on the standard LPN assumption is also possible, but it incurs substantially higher costs, with evaluation time growing quadratically in the dimension, which may grow with the number of parties. As we show in later sections, when only a constant fraction of parties is corrupted, relying purely on the standard LPN assumption still yields secure and efficient results.

6 Extensions

In this section, we present several extensions to our scheme aimed at improving performance and enhancing functionality.

6.1 Working with a Constant Fraction of Corruptions

In Section 5, we presented a construction of a nPCF that supports efficient multi-party computation and illustrates our core technique for generating Beaver triples. Although the computation is efficient, storage becomes the main bottleneck for scalability: to keep the nPCF noise rate below ϵ, we require Sparse LPN with noise rate approximately $\varepsilon = \epsilon/N$. This, in turn, forces the LPN dimension to grow proportionally with N in order to maintain concrete security. While the evaluation cost remains modest, the storage required to support large N quickly becomes prohibitive.

We now show how to overcome this issue in settings where only a constant fraction of parties are corrupted. Such assumptions naturally arise in large-scale systems, such as blockchains. In this setting, our new construction largely follows the same outline as before, but apart from Sparse LPN, it also leverages a new pseudorandom zero-sharing (PRZS) scheme. We prove that, under constant-fraction corruption, a PRZS scheme with computation time and storage complexity independent of N, combined with the Sparse LPN assumption, yields a nPCF whose computation and storage are both independent of N.

6.1.1 Tool: Pseudorandom zero sharing

The first ingredient we require is a pseudorandom zero-sharing scheme, denoted by PRZS and introduced in [CDI05]. In this work, we focus primarily on the PRZS scheme in the context of additive secret sharing under the assumption that only a constant fraction of parties may be corrupted. We will show that in this setting we can construct a PRZS scheme whose shares can be computed using $O(\log \lambda)$ calls to the PRF.

Our definitions, however, are stated in full generality, allowing the corruption threshold t to be set arbitrarily during setup. It is specifically when $t \leq \alpha \cdot N$ for some constant $\alpha \in (0, 1)$ that our construction achieves improved efficiency over prior work.

Definition 6.1 (Pseudorandom Zero-Sharing)**.** *Let* $(\mathsf{PRZS.Gen}, \mathsf{PRZS.Eval})$ *be a pair of polynomial-time algorithms defined as follows:*

- $\mathsf{PRZS.Gen}(1^\lambda, 1^N, t \in [0, N-1])$ *is a probabilistic polynomial-time algorithm that takes the security parameter* 1^λ*, the number of parties* 1^N*, and an upper bound on the number of corrupted parties* t *as input, and outputs* N *evaluation keys* $K_1, \cdots, K_N$*.*
- $\mathsf{PRZS.Eval}(K_i, x \in \{0,1\}^\lambda)$ *is a deterministic polynomial-time algorithm that takes the key* K_i *assigned to party* i *and input value* x*, and outputs a value* z_i*.*

We say that $(\mathsf{PRZS.Gen}, \mathsf{PRZS.Eval})$ *is a pseudorandom zero-sharing scheme for* $N(\lambda)$ *parties over a field* $\mathbb{F}$ *that resists up to* $t(\lambda)$ *corruptions, if the following properties hold:*

- **Correctness.** *For any* $\lambda, N \in \mathbb{N}$ *and* $t \in [0, N-1]$*,*

$$\Pr\left[\sum_{i=1}^{N} z_i = 0 \;\middle|\; \begin{array}{l} \{K_i\}_{i\in[N]} \leftarrow \mathsf{PRZS.Gen}(1^\lambda, N, t) \\ x \leftarrow \{0,1\}^\lambda \\ z_i \leftarrow \mathsf{PRZS.Eval}(K_i, x), \forall i \in [N] \end{array}\right] = 1.$$

- (N, t)**-Pseudorandomness.** *For polynomials* $N(\lambda), t(\lambda) \in [0, N(\lambda) - 1]$*, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for any* $\lambda \in \mathbb{N}$*,* $N = N(\lambda)$ *and* $t \in [0, t(\lambda)]$*, the adversary's advantage in experiment* $\mathsf{Exp}^{\mathsf{PRZS}}_{\mathcal{A},N,t}(\lambda)$ *(shown in Figure 3) is bounded by* $\mathsf{negl}(\lambda)$*.*

Construction of Pseudorandom Zero-Sharing with logarithmic calls to the PRF We now present a pseudorandom zero-sharing construction with improved efficiency based on a PRF. The full construction is detailed in Figure 4. The proposed PRZS is compatible with any field $\mathbb{F}_q$, provided that the PRF generates values in $\mathbb{F}_q$. In this work, we primarily focus on the field $\mathbb{F}_2$.

Remark 6.1. The construction in Figure 4 is the same as in [CDI05] if p is set to be 1.

Theorem 6.1. *For any* $N(\lambda), t(\lambda)$ *bounded by polynomials in* λ*, if* p *is set to* $\omega\left(\frac{\log(\lambda)}{N-t}\right)$ *and* PRF *outputs values in* $\mathbb{F}$*, then the scheme shown in Figure 4 is a pseudorandom zero-sharing scheme for* $\mathbb{F}$ *satisfying Definition 6.1 that supports* N *parties and resists up to* t *corrupted parties, and each party evaluates* $O(p \cdot N)$ PRF *calls for each zero-sharing.*

Proof. As evident from the scheme, for each zero sharing, the share obtained by each party is a linear combination of PRF evaluations. Formally, we have

$$z_i = \sum_{(u,v)\in\mathcal{E}, v=i} \mathsf{PRF.Eval}(\mathsf{PRF}.K_{u,v}, x) - \sum_{(u,v)\in\mathcal{E}, u=i} \mathsf{PRF.Eval}(\mathsf{PRF}.K_{u,v}, x).$$

Experiment $\mathsf{Exp}^{\mathsf{PRZS}}_{\mathcal{A},N,t}(\lambda)$

- Execute $\mathcal{A}(1^\lambda, 1^N, t)$ to get a set of indices $T \subset [N]$ satisfying $|T| \leq t$.
- Now the challenger samples $b \leftarrow \{0,1\}$, then evaluates the key generation $\{K_i\}_{i\in[N]} \leftarrow \mathsf{PRZS.Gen}(1^\lambda, 1^N, t)$ and sends keys of corrupted parties $\{K_i\}_{i\in T}$ to $\mathcal{A}$.
- $\mathcal{A}$ now makes arbitrary oracle queries. For each query, the challenger receives a distinct (previously not queried) input value $x \in \{0,1\}^\lambda$ and replies as follows:
  - Compute $y_{i,0} \leftarrow \mathsf{PRZS.Eval}(K_i, x)$ for all $i \in [N]$.
  - Sample $y_{i,1} \in \mathbb{F}$ for all $i \notin T$ uniformly conditioned on $\sum_{i\in T} y_{i,0} + \sum_{i\notin T} y_{i,1} = 0$.
  - Send x and $\{y_{i,b}\}_{i\notin T}$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs a bit b'. The advantage of $\mathcal{A}$ is $|2 \times \Pr[b' = b] - 1|$.

Figure 3: PRZS Security Experiment

Let $\boldsymbol{z} = (z_1, \ldots, z_N)$ denote the vector of PRZS evaluations. Using the notation $\boldsymbol{r}$ to represent the vector of PRF evaluations, we have $\boldsymbol{z} = \boldsymbol{M} \cdot \boldsymbol{r}$ for some matrix $\boldsymbol{M}$. If we model the parties as vertices and the pairs in $\mathcal{E}$ as edges, then we obtain a graph and the matrix $\boldsymbol{M}$ is exactly the oriented incidence matrix of the graph.

With this observation, we invoke hybrid arguments to prove the lemma.

- Hyb_0: This hybrid is the experiment $\mathsf{Exp}^{\mathsf{PRZS}}_{\mathcal{A},N,t}(\lambda)$ with $b = 0$.
- Hyb_1: This hybrid is the same as Hyb_0, except that for all PRF keys $\mathsf{PRF}.K_{u,v}$, if parties u, v are both uncorrupted, the challenger uses random values to replace the evaluation results of $\mathsf{PRF.Eval}(\mathsf{PRF}.K_{u,v}, x)$. The indistinguishability between Hyb_0 and Hyb_1 comes directly from the security of PRF.
- Hyb_2: This hybrid is the experiment $\mathsf{Exp}^{\mathsf{PRZS}}_{\mathcal{A},N,t}(\lambda)$ with $b = 1$. To show the indistinguishability between Hyb_1 and Hyb_2, we argue that with high probability over the generation of keys and the choice of T, for each oracle query, $\left\{y'_{i,0}\right\}_{i\notin T}$ are uniformly random conditioned on $\sum_{i\in[N]} y'_{i,0} = 0$. Here, $y'_{i,0}$ represents the value obtained from the evaluation of $\mathsf{PRZS.Eval}(K_i, x)$, except that random values are used in place of PRF evaluations if $\mathsf{PRF}.K_{u,v}$ satisfies the condition that both u and v are not corrupted, as in Hyb_1.

  By expanding $y'_{i,0}$, we have

$$y'_{i,0} = \sum_{(u,v)\in E_i, s.t. u\in T \vee v\in T} (-1)^{\mathbb{I}(u=i)} \mathsf{PRF.Eval}(\mathsf{PRF}.K_{u,v}, x) + \sum_{(u,v)\in E_i, s.t. u,v\notin T} (-1)^{\mathbb{I}(u=i)} r_{u,v},$$

  where $\{r_{u,v}\}$ are all uniformly random values.

  We now focus only on the second summation. That is, we want to show $\{y'_i\}_{i\notin T}$ is a uniformly random zero-sharing, where y'_i only computes the second summation:

$$y'_i = \sum_{(u,v)\in E_i, s.t. u,v\notin T} (-1)^{\mathbb{I}(u=i)} r_{u,v},$$

Pseudorandom Zero-Sharing

- $\mathsf{PRZS.Gen}(1^\lambda, 1^N, t) \to \{K_i\}_{i\in[N]}$. On input 1^λ, 1^N, and t,
  - Let $\mathcal{E}$ be an initially empty set of index pairs, and independently put each pair (u, v) into $\mathcal{E}$ with probability p for all $u, v \in [N]$ satisfying $u < v$. Here, p is a parameter used to balance the efficiency and error probability.
  - Generate a PRF key $\mathsf{PRF}.K_{u,v}$ for each pair in $\mathcal{E}$.
  - For $i \in [N]$, let K_i be the set of PRF keys $\{\mathsf{PRF}.K_{u,v} \mid (u = i \vee v = i) \wedge (u, v) \in \mathcal{E}\}$.
  - Output $\{K_i\}_{i\in[N]}$.
- $\mathsf{PRZS.Eval}(K_i, x) \to z_i$. On input K_i, x,
  - Parse K_i as $\{\mathsf{PRF}.K_{u,v}\}_{(u,v)\in E_i}$.
  - For $(u, v) \in E_i$, compute $z_{i,(u,v)} \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K_{u,v}, x)$.
  - Output $z_i = \sum_{(u,v)\in E_i} (-1)^{\mathbb{I}(u=i)} z_{i,(u,v)}$.

Figure 4: Construction of PRZS

If we use a vector $\boldsymbol{y}'$ to represent $\{y'_i\}_{i\notin T}$ and use a vector $\boldsymbol{r}'$ to represent $\{r_{u,v}\}_{u,v\notin T}$, then we also have $\boldsymbol{y}' = \boldsymbol{M}' \cdot \boldsymbol{r}'$, where $\boldsymbol{M}'$ is the incidence matrix of the induced subgraph formed from vertex set $[N]/T$.

As is well known in graph theory, the oriented incidence matrix has row rank $|V| - 1$ if the undirected graph is connected, where V is the set of vertices. Combined with the fact that $\sum_{i\notin T} y'_i = 0$, it is easy to see that if the subgraph induced by the set $[N]/T$ is connected, we would have $\boldsymbol{y}'$ uniformly sampled at random conditioned on $\sum_{i\notin T} y'_i = 0$.

Lemma 6.1 ([ER60]). *For a graph with vertex set V where each edge appears with probability p independently, if $p \geq \beta \frac{\log |V|}{|V|}$, then the probability that the graph is connected is $1 - \widetilde{O}(n^{-\beta})$.*

In our case, each edge appears with probability $p \in \omega\left(\frac{\log(\lambda)}{N-t}\right)$, and the probability that the subgraph is disconnected is bounded by

$$\widetilde{O}\left((N-t)^{-\omega\left(\frac{\log\lambda}{\log(N-t)}\right)}\right) \leq 2^{-\omega(\log\lambda)},$$

which is negligible. Therefore, Hyb_1 and Hyb_2 are computationally indistinguishable, completing the proof.

□

6.1.2 Construction of Noisy PCF from PRZS

We now present the construction of nPCF in Figure 5, which essentially augments the output values with a pseudorandom zero-sharing. (For succinctness, the noise indicator remains unchanged and is omitted from the figure.) The key point is that this modification allows us to establish the following theorem.

Noisy PCF for Beaver triples with a constant fraction of corruptions

- $\mathsf{nPCF.Gen}(1^\lambda, 1^N, t) \to \{K_i\}_{i\in[N]}$. On input 1^λ, 1^N and t,
  - Sample secrets $\boldsymbol{s}_1 \leftarrow \mathbb{F}^n, \boldsymbol{s}_2 \leftarrow \mathbb{F}^n$ uniformly at random and generate the additive shares of $\boldsymbol{s}_1, \boldsymbol{s}_2$ and $\boldsymbol{s}_1 \otimes \boldsymbol{s}_2$, denoted by $[\![\boldsymbol{s}_1]\!], [\![\boldsymbol{s}_2]\!]$ and $[\![\boldsymbol{s}_1 \otimes \boldsymbol{s}_2]\!]$.
  - Generate PRZS keys $\{\mathsf{PRZS}.K_i\}_{i\in[N]} \leftarrow \mathsf{PRZS.Gen}(1^\lambda, N, t)$.
  - Generate PRF keys $\mathsf{PRF}.K_i \leftarrow \mathsf{PRF.Gen}(1^\lambda)$ for $i \in [N]$.
  - Let $K_i = (\mathsf{PRZS}.K_i, \mathsf{PRF}.K_i, [\![\boldsymbol{s}_1]\!]_i, [\![\boldsymbol{s}_2]\!]_i, [\![\boldsymbol{s}_1 \otimes \boldsymbol{s}_2]\!]_i)$ and output $\{K_i\}_{i\in[N]}$.
- $\mathsf{nPCF.Eval}(K_i, x) \to y_i \in \mathbb{F}^3$. On input K_i, x,
  - Parse K_i as $(\mathsf{PRZS}.K_i, \mathsf{PRF}.K_i, [\![\boldsymbol{s}_1]\!]_i, [\![\boldsymbol{s}_2]\!]_i, [\![\boldsymbol{s}_1 \otimes \boldsymbol{s}_2]\!]_i)$.
  - Interpret input x as a vector $\boldsymbol{a} \in \mathbb{F}^n$ and a bit string $r \in \{0,1\}^\lambda$.
  - Evaluate PRF to obtain $r_i \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K_i, r)$.
  - Sample noise terms $(e_{i,1}, e_{i,2}, e_{i,3}) \leftarrow \mathcal{D}_e^3(\eta)$ using randomness r_i, where η satisfies $(1-\eta)^{2N} \geq 1-\epsilon$ and $\mathcal{D}_e^3(\mathbb{F}, \eta)$ is the distribution in Definition 4.1.
  - For $j \in \{1,2,3\}$, compute $z_{i,j} \leftarrow \mathsf{PRZS.Eval}(\mathsf{PRZS}.K_i, (x, j))$.
  - Compute $u_i = \langle \boldsymbol{a}, [\![\boldsymbol{s}_1]\!]_i \rangle + e_{i,1} + z_{i,1}, v_i = \langle \boldsymbol{a}, [\![\boldsymbol{s}_2]\!]_i \rangle + e_{i,2} + z_{i,2}, w_i = \langle \boldsymbol{a} \otimes \boldsymbol{a}, [\![\boldsymbol{s}_1 \otimes \boldsymbol{s}_2]\!]_i \rangle + e_{i,3} + z_{i,3}$.
  - Output $\boldsymbol{y}_i = (u_i, v_i, w_i)$.

Figure 5: Construction of PCF for noisy Beaver triples

Theorem 6.2. *Consider the same parameter settings as in Theorem 5.1, with the following difference:*

- $t(\lambda)$ *is any function in* $[0, N(\lambda) - 1]$.
- *The noise rate* ϵ *of* nPCF *is set to satisfy* $\epsilon \geq 1 - \left(1 - \frac{4\varepsilon}{1+2\varepsilon}\right)^{2N/(N-t)}$.
- PRZS *is a pseudorandom zero-sharing scheme.*

Assume that the $(\varepsilon, 2, k, n, m)$*- sparse LPN assumption holds for all polynomials* $m(\lambda) \in \mathsf{poly}(\lambda)$*. Then, the scheme shown in Figure 5 is a* (N, t)*-noisy pseudorandom correlation function for Beaver triples in* $\mathbb{F}_2$ *with respect to input distribution* $\mathcal{X}$ *with noise rate* ϵ*. Moreover, each* nPCF *evaluation requires* $O(k^2)$ *computational steps, along with a* PRF *evaluation and a* PRZS *evaluation.*

Corollary 6.1. *Assume for* $n \in \mathsf{poly}(\lambda)$*, the* $(\varepsilon, 2, n, m)$-LPN *assumption holds for all polynomials* $m(\lambda)$*. For any polynomials* $N(\lambda), t(\lambda) \in [0, N(\lambda) - 1]$ *and noise rate* $\epsilon(\lambda)$ *satisfying* $\epsilon \geq 1 - \left(1 - \frac{4\varepsilon}{1+2\varepsilon}\right)^{2N/(N-t)}$ *for all* $\lambda \in \mathbb{N}$*, the scheme shown in Figure 2 is a* (N, t)*-noisy pseudorandom correlation function for Beaver triples in* $\mathbb{F}_2$ *with respect to input distribution* $\mathcal{X}$ *with noise rate* ϵ*, where* $\mathcal{X}_\lambda$ *describes the uniform distribution over* $\mathbb{F}_2^n$ *and a bit string* r *uniformly sampled from* $\{0,1\}^\lambda$*. Moreover, each* nPCF *evaluation requires* $O(n^2)$ *computational steps, along with a* PRF *evaluation and a* PRZS *evaluation, which is independent of* $N(\lambda)$ *in the case of constant-fraction corruption.*

Proof. Clearly, *ϵ-error probability* holds for the same reason as in the prior construction.

To prove that the scheme satisfies (N, t)-*Weak pseudorandomness* in Definition 5.1, we rely on the following sequence of hybrid arguments:

- Hyb_0: This hybrid corresponds to the experiment $\mathsf{Exp}^{\mathsf{nPCF}}_{\mathcal{A},N,t,\mathcal{X}}(\lambda)$ with $b = 0$.
- Hyb_1: This hybrid is identical to Hyb_0, except that when computing $y_{i,0} \leftarrow \mathsf{nPCF.Eval}(K_i, x)$ for the honest party, the noise vector $\boldsymbol{e}_i$ is generated using fresh randomness, rather than derived from the PRF evaluation randomness r_i.

  Since the PRF key K_i is hidden from the adversary, and the probability that the adversary queries the same PRF input is negligible, the outputs remain pseudorandom. Hence, by the pseudorandomness property of the PRF, the adversary cannot distinguish Hyb_0 from Hyb_1.
- Hyb_2: This hybrid is the same as Hyb_1 except that for an oracle query, $\mathcal{A}$ would receive x and $\{\boldsymbol{y}'_i\}_{i\notin T}$, instead of $\{\boldsymbol{y}_{i,0}\}_{i\notin T}$. Here, $\boldsymbol{y}'_i$ is sampled uniformly at random, conditioned on $\sum_{i\notin T} \boldsymbol{y}'_i = \sum_{i\notin T} \boldsymbol{y}_{i,0}$.

  For the indistinguishability between Hyb_1 and Hyb_2, we leverage the pseudorandomness of PRZS. Suppose $\mathcal{A}$ can distinguish Hyb_1 and Hyb_2, then the following adversary $\mathcal{A}'$ can break the pseudorandomness of PRZS:
  - Simulate the experiment in Hyb_1 by acting as the challenger with the following modifications:
    - Instead of generating keys for PRZS itself, $\mathcal{A}'$ would receive keys $\{\mathsf{PRZS}.K_i\}_{i\in T}$ from the challenger in $\mathsf{Exp}^{\mathsf{PRZS}}_{\mathcal{A}',N,t}$ by choosing the same index set T as $\mathcal{A}$.
    - When $\mathsf{nPCF.Eval}(K_i, x)$ should be invoked for $i \notin T$, generate $z_{i,j}$ by querying the challenger in $\mathsf{Exp}^{\mathsf{PRZS}}_{\mathcal{A}',N,t}$ with input (x, j).
  - Without loss of generality, assume $\mathcal{A}$ distinguishes $\mathsf{Hyb}_1, \mathsf{Hyb}_2$ by outputting $b' = 1$ with higher probability in Hyb_2. Then $\mathcal{A}'$ outputs $b' = 1$ if and only if $\mathcal{A}$ outputs $b' = 1$ in the simulated experiment.

  One can easily verify that if $b = 0$ in $\mathsf{Exp}^{\mathsf{PRZS}}_{\mathcal{A}',N,t}$, the simulated experiment is the same as Hyb_1, while if $b = 1$ in $\mathsf{Exp}^{\mathsf{PRZS}}_{\mathcal{A}',N,t}$, the simulated experiment is the same as Hyb_2. Thus $\mathcal{A}'$ can break the pseudorandomness of PRZS if $\mathcal{A}$ has non-negligible advantage, which implies Hyb_1 and Hyb_2 are indistinguishable.
- Hyb_3: This hybrid is identical to Hyb_2, except that the oracle query responses are reformulated as follows:
  - Let $u = \langle \boldsymbol{a}, \boldsymbol{s}_1 \rangle$, $v = \langle \boldsymbol{a}, \boldsymbol{s}_2 \rangle$, and $w = \langle \boldsymbol{a} \otimes \boldsymbol{a}, \boldsymbol{s}_1 \otimes \boldsymbol{s}_2 \rangle$. Sample (e_1, e_2, e_3) independently from $\mathcal{D}^3_e(\eta')$, where η' satisfies $\eta' = 1 - (1 - \eta)^{N-|T|}$.
  - Let
$$u' = u - \sum_{i'\in T} \langle \boldsymbol{a}, [\![\boldsymbol{s}_1]\!]_{i'} \rangle + e_1, \quad v' = v - \sum_{i'\in T} \langle \boldsymbol{a}, [\![\boldsymbol{s}_2]\!]_{i'} \rangle + e_2,$$
$$w' = w - \sum_{i'\in T} \langle \boldsymbol{a} \otimes \boldsymbol{a}, [\![\boldsymbol{s}_1 \otimes \boldsymbol{s}_2]\!]_{i'} \rangle + e_3.$$
  - Sample $\boldsymbol{y}'_i$ uniformly at random for honest parties $i \notin T$, conditioned on $\sum_{i\notin T} \boldsymbol{y}'_i = (u', v', w')$.

– Reply to the adversary with $\boldsymbol{y}'_i$ for $i \notin T$.

This change is purely representational, as we only explicitly write out the formula for $\sum_{i\notin T} \boldsymbol{y}_{i,0}$. Thus, Hyb_3 is identical to Hyb_2.

- Hyb_4: This hybrid is identical to Hyb_3, except that (u, v, w) in oracle responses is now sampled directly from $\mathcal{D}_{\mathsf{Beaver}}(\mathbb{F}_2)$.

  If the adversary makes m oracle queries, then distinguishing between Hyb_3 and Hyb_4 reduces to distinguishing between the following two distributions:

$$\left\{\boldsymbol{a}^j,\ \langle \boldsymbol{a}^j, \boldsymbol{s}_1\rangle + e_1^j,\ \langle \boldsymbol{a}^j, \boldsymbol{s}_2\rangle + e_2^j,\ \langle \boldsymbol{a}^j \otimes \boldsymbol{a}^j, \boldsymbol{s}_1 \otimes \boldsymbol{s}_2\rangle + e_3^j\right\}_{j\in[m]}$$
$$\approx_c$$
$$\left\{\boldsymbol{a}^j,\ u^j + e_1^j,\ v^j + e_2^j,\ w^j + e_3^j\right\}_{j\in[m]},$$

  where $\boldsymbol{s}_1, \boldsymbol{s}_2$ are uniformly random vectors, $\boldsymbol{a}^j$ is sampled from the set of k-sparse vectors, (e_1^j, e_2^j, e_3^j) are drawn from $\mathcal{D}_e^3(\eta')$, and (u^j, v^j, w^j) are sampled from $\mathcal{D}_{\mathsf{Beaver}}(\mathbb{F}_2)$.

  By Lemma 4.2, this indistinguishability holds whenever $\frac{\eta'}{4-2\eta'} \geq \varepsilon$ under the $(\varepsilon, 2, k, n, m)$-sLPN assumption. In fact, the left-hand side matches $\mathcal{D}'_{\mathsf{QLPN}_{n,m,\eta',2}}$, and the lemma reduces the claim to proving

$$\mathcal{D}^{\mathsf{rand}}_{\mathsf{QLPN}_{n,m,\eta',2}} \approx_c \left\{\boldsymbol{a}^j,\ u^j + e_1^j,\ v^j + e_2^j,\ w^j + e_3^j\right\}_{j\in[m]},$$

  where $(u^j, v^j, w^j) \leftarrow \mathcal{D}_{\mathsf{Beaver}}(\mathbb{F}_2)$ and $(e_1^j, e_2^j, e_3^j) \leftarrow \mathcal{D}_e^3(\eta')$.

  By the definition of $\mathcal{D}^{\mathsf{rand}}_{\mathsf{QLPN}_{n,m,\eta',2}}$, these two distributions are identical. Moreover, the condition $\frac{\eta'}{4-2\eta'} \geq \varepsilon$ is always satisfied whenever $\epsilon \geq 1 - \left(1 - \frac{4\varepsilon}{1+2\varepsilon}\right)^{2N/(N-t)}$, and we assume the sparse LPN assumption for all polynomials $m \in \mathsf{poly}(\lambda)$. Hence, we conclude that $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_4$.

- Hyb_5: This hybrid corresponds to the experiment $\mathsf{Exp}^{\mathsf{nPCF}}_{\mathcal{A},N,t,\mathcal{X}}(\lambda)$ with $b = 1$.

  Compared with Hyb_4, this is again only a representational change. Thus, Hyb_4 and Hyb_5 are identical, which completes the proof.

□

6.2 Working with Polynomial-Size Fields

We also demonstrate that our scheme generalizes to polynomial-size fields, rather than only binary fields. To achieve this, we claim that we can directly use the prior construction shown in Figure 2, with the only modification being the use of $\mathbb{F}_q$. We formally state this result as follows:

Theorem 6.3. *For $n, k, q \in \mathsf{poly}(\lambda)$, assume the $(\varepsilon, q, k, n, m)$-sLPN assumption holds for all polynomials $m(\lambda)$ and PRZS is a pseudorandom zero-sharing scheme. For any polynomials $N(\lambda)$, $t(\lambda) \in [1, N(\lambda)]$ and noise rate $\epsilon(\lambda)$ satisfying $\epsilon \geq 1 - \left(1 - \frac{q^2}{q-1}\frac{\varepsilon}{1+q\varepsilon}\right)^{\frac{2N}{N-t}}$ for all $\lambda \in \mathbb{N}$, the scheme shown in Figure 2 is a (N, t)-noisy pseudorandom correlation function for Beaver triples in $\mathbb{F}_q$ with respect to input distribution $\mathcal{X}$ with noise rate ϵ, where $\mathcal{X}_\lambda$ describes the distribution that samples vector $\boldsymbol{a}$ uniformly from $\mathbb{F}_q^n$ and bit string r uniformly from $\{0,1\}^\lambda$.*

Remark 6.2. The extension to large fields can be combined with the extension above, which utilizes the sparse LPN assumption to improve performance.

By examining the proof for Theorem 6.2, we can find that the property of $\mathbb{F}_2$ is never used except when invoking Lemma 4.2. Thus, despite the difference in fields, we can still adapt the proof with modifications only in the parameters, provided we can establish a lemma similar to Lemma 4.2. Here, we give a generalized version of this lemma to address the issue.

Lemma 6.2. *Let λ be the security parameter. For polynomial modulus $q = q(\lambda)$, noise rate $\varepsilon = \varepsilon(\lambda)$, dimension $n = n(\lambda) \in \mathsf{poly}(\lambda)$, sparsity parameter $k = k(\lambda) \in \mathsf{poly}(\lambda)$ and number of samples $m = m(\lambda) \in \mathsf{poly}(\lambda)$, assuming that the $\left(\varepsilon' = \frac{(q-1)\varepsilon/q}{q-\varepsilon}, q, n, m\right)$-$\mathsf{LPN}$ assumption holds, then for the same parameters $n = n(\lambda)$ and $m = m(\lambda)$:*

$$\left\{\mathcal{D}_{\mathsf{QLPN}'_{n,m,\varepsilon,q}}\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathcal{D}^{\mathsf{rand}}_{\mathsf{QLPN}'_{n,m,\varepsilon,q}}\right\}_{\lambda\in\mathbb{N}}.$$

The proof of Lemma 6.2 follows the proof of Lemma 4.2. It can be verified that, prior to the proof of Claim 4.3, the proof does not use the properties of the field $\mathbb{F}_2$. Therefore, it suffices to address the issue of reducing the LPN assumption to distinguishing between the following two distributions:

$$\mathcal{D}_1 = \{(\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s}\rangle + e_{1,i}, e_{2,i})\}_{i\in[m]}, \quad \mathcal{D}_2 = \{(\boldsymbol{a}_i, r_i + e_{1,i}, e_{2,i})\}_{i\in[m]},$$

where $\boldsymbol{a}_i, \boldsymbol{s}$ are uniformly sampled from $\mathbb{F}_q^n$, r_i are uniformly random values in $\mathbb{F}_q$, and $(e_{1,i}, e_{2,i})$ are sampled from the distribution $\mathcal{D}^2_\varepsilon(\mathbb{F}_q, \varepsilon)$. Using a technique similar to the one in the original proof, which analyzes the marginal distribution of $e_{1,i}$ conditioned on different values of $e_{2,i}$, we can deduce the following:

- When $e_{2,i} \neq 0$, $e_{1,i}$ is uniformly distributed.
- When $e_{2,i} = 0$, $e_{1,i}$ follows a Bernoulli distribution $\mathsf{Ber}(\mathbb{F}_q, \varepsilon')$.

By constructing the adversary in the same way as in the original proof, we can conclude the desired result.

6.3 Reduced Storage in Honest Majority Setting

We note that our scheme can achieve a significant performance boost in the honest-majority setting, when the field size is large enough to support Shamir secret sharing, i.e., $q > N$. The key idea is to replace additive secret sharing with Shamir secret sharing for the Sparse LPN secrets. This allows us to exploit the multiplicative properties of Shamir sharing to compute inner products of the form

$$\langle \boldsymbol{a} \otimes \boldsymbol{a}, [\![\boldsymbol{s}_1 \otimes \boldsymbol{s}_2]\!]\rangle.$$

Concretely, we first compute $[\![x_1]\!] = \langle \boldsymbol{a}, [\![\boldsymbol{s}_1]\!]\rangle$ and $[\![x_2]\!] = \langle \boldsymbol{a}, [\![\boldsymbol{s}_2]\!]\rangle$. Shamir secret sharing then guarantees that $[\![x_1]\!] \cdot [\![x_2]\!]$ is a valid sharing of $x_1 \cdot x_2 = \langle \boldsymbol{a} \otimes \boldsymbol{a}, \boldsymbol{s}_1 \otimes \boldsymbol{s}_2\rangle$, provided the degree of the sharing polynomial satisfies $d < N/2$.

In this way, when an honest majority is available, we no longer need to explicitly store shares of $\boldsymbol{s}_1 \otimes \boldsymbol{s}_2$, yielding a substantial reduction in storage requirements. Moreover, the computation per evaluation involves only $O(k)$ field operations—together with calls to PRZS and PRF—where k is the sparsity parameter.

If we want to provide Beaver triples for Boolean operations, we may instantiate $\mathbb{F}_q$ as an extension field of the binary field, with $q = 2^{\lceil \log_2 N \rceil}$. The protocol operates over $\mathbb{F}_q$, while shares can be projected back to the base field (e.g., via the constant coefficient of the polynomial representation) to recover Boolean outputs and keep the communication concise. This ensures compatibility with Boolean circuit evaluation while retaining the efficiency benefits of large-field secret sharing.

7 MPC Protocols with Small Super-Constant Overhead

In this section, we introduce two methods that utilize the noisy Beaver triples to achieve scalable multi-party computation with semi-honest security.

MPC Security Definition. Here, we focus on the dishonest majority setting. Let $t < N$ be an integer. Let $\mathcal{F}$ be a secure function evaluation functionality. A semi-honest adversary $\mathcal{A}$ can corrupt at most t parties, learn the internal randomness of corrupted parties, and receive all messages sent to corrupted parties.

Real World Execution. In the real world, corrupted parties interact with honest parties. At the end of the protocol, the output of the real world execution includes the outputs of honest parties and the view of the adversary $\mathcal{A}$.

Ideal World Execution. In the ideal world, a simulator Sim simulates honest parties and interacts with corrupted parties. Furthermore, Sim has one-time access to the functionality $\mathcal{F}$, which gives the evaluation result of $\mathcal{F}$ given the correct inputs from all parties. The output of the ideal world execution includes the outputs of honest parties and the view of the adversary $\mathcal{A}$.

Semi-honest Security. We say that a protocol π computes $\mathcal{F}$ with computational security if for every semi-honest PPT adversary $\mathcal{A}$, there exists a simulator Sim such that the distribution of the output of the real world execution is computationally indistinguishable from the distribution in the ideal world execution.

Functionality $\mathcal{F}_{\mathsf{MPC}}$

Let $C : \{0,1\}^{\ell_{\text{in}}} \to \{0,1\}^{\ell_{\text{out}}}$ be a Boolean circuit. Let $x \in \{0,1\}^{\ell_{\text{in}}}$ be the input, and each party holds the share of x, denoted by $[\![x]\!]_i$ for party i.
Each party sends its share $[\![x]\!]_i$ to $\mathcal{F}_{\mathsf{MPC}}$. $\mathcal{F}_{\mathsf{MPC}}$ evaluates $\boldsymbol{y} \leftarrow C(x)$ and returns the result $\boldsymbol{y} \in \{0,1\}^{\ell_{\text{out}}}$ to each party.

Figure 6: Functionality of secure computation

MPC functionality. We now describe the functionality $\mathcal{F}_{\mathsf{MPC}}$ as shown in Figure 6, which models the task of secure computation and represents the core functionality we aim to implement. The functionality interacts with N parties that hold input shares and carry out the computation.

7.1 MPC Protocol Using Security Amplification

Since our nPCF only generates noisy triples, it is essential to handle the errors to ensure the correctness of the protocol. Here, we present a protocol that first derives correct triples from the noisy ones and then uses them to achieve secure MPC. The protocol is presented in Figures 7, 8. In this protocol, the parties invoke the Correct and SecurityAmplify procedures to generate secure and correct Beaver triples. The triples produced by the Correct procedure are guaranteed to be correct,

though there is a small probability that they may be insecure. Subsequently, the SecurityAmplify procedure enhances the security of triples by leveraging the output of the Correct procedure. The resulting Beaver triples are secure with overwhelming probability. For simplicity, we omit the details of reconstructing a value from its secret shares in the figure. In the semi-honest setting, this can be done by designating a single party, rotated in a round-robin fashion, to aggregate the masked shares and return the result. This ensures that the amortized computation and communication per party remain $O(1)$.

Theorem 7.1. *Let $N, t, \epsilon, \kappa, \beta$ be functions of the security parameter λ, where $N, t \in [0, N-1]$ are bounded by polynomials in λ, $\epsilon \leq \frac{1}{\lambda}$, and $\kappa, \beta \in \omega(1) \cap O(\log \lambda)$. We assume the existence of an (N, t)-noisy PCF for Beaver triples in $\mathbb{F}_2$ with noise rate ϵ, and that* nPCF *keys have been distributed to enable repeated protocol evaluations. Let $\mathcal{X}$ be the input distribution required by* nPCF*, and let $\mathcal{O}$ be a random oracle whose output follows $\mathcal{X}$. The protocol π_{MPC}, as described in Figure 8, with parameters $N(\lambda), t(\lambda), \epsilon(\lambda), \kappa(\lambda), \beta(\lambda)$, securely instantiates the functionality $\mathcal{F}_{\mathsf{MPC}}$ with semi-honest security against adversaries capable of corrupting at most t parties, with per-gate, per-party computation cost $O(\kappa \cdot \beta^2 \cdot \mathsf{Time}(\mathsf{nPCF.Eval}))$ and communication cost $O(\kappa \cdot \beta^2)$.*

Corollary 7.1. *Let $N, t, \epsilon, \kappa, \beta$ be functions of the security parameter λ, where $N, t \in [0, N-1]$ are bounded by polynomials in λ, $\epsilon \leq 1/\lambda$, $\kappa \in \omega(1) \cap O(\log \lambda)$, and $\beta \in \omega(1)$. Under the $(\varepsilon, 2, k, n, m)$-sparse LPN assumption required in Theorem 5.1, we obtain a semi-honest MPC protocol in the random oracle model with once-for-all preprocessing. The protocol is secure against adversaries corrupting up to t parties, and achieves per-gate, per-party computation cost $O(\kappa \cdot \beta^2 \cdot k^2)$ and communication cost $O(\kappa \cdot \beta^2)$.*

Proof. We first argue that our protocol correctly implements the $\mathcal{F}_{\mathsf{MPC}}$ functionality. Specifically, we claim that the Correct procedure generates correct Beaver triples, and thus the same holds for the SecurityAmplify procedure. This ensures the overall correctness of our protocol.

Claim 7.1. *There exists a negligible function* negl *such that for any $\lambda \in \mathbb{N}$, let $(\llbracket u \rrbracket, \llbracket v \rrbracket, \llbracket w \rrbracket)$ be the shares of a triple generated by invoking* Correct*, then with probability $1 - \mathsf{negl}(\lambda)$, w is equal to $u \cdot v$.*

Proof. By examining the Correct procedure and the And procedure, we can observe that e^j is equal to $(w^j - u^j \cdot v^j) - (w^1 - u^1 \cdot v^1)$. Thus, to achieve the condition that $e^j = 0$ for all $j \in [2, \kappa]$, either all triples are correct, or all triples are incorrect.

By the *ϵ-Error Probability* of nPCF, with probability $1 - \mathsf{negl}(\lambda)$, the probability that a triple is incorrect is bounded by $\epsilon(\lambda) + \mathsf{negl}(\lambda)$ for some negligible function negl. Combined with the fact that inputs of nPCF come from random oracle $\mathcal{O}$ and are independent, we conclude that the probability that all triples are incorrect is negligible when $\kappa \in \omega(1), \epsilon \leq \frac{1}{\lambda}$. □

Now we move to the security of the protocol. We will construct a simulator $\mathsf{Sim}_{\mathsf{MPC}}$ to simulate the behavior of honest parties. Before describing the simulator, we first introduce the following notation: Let T represent the corrupted parties and $\bar{T}$ denote the honest parties. We say that a Beaver triple correlation is perfectly simulated if there exists a fresh Beaver triple (u, v, w), and the shares held by the honest parties are uniformly sampled, conditioned on the shared triple being (u, v, w). If we replace a shared triple with a perfectly simulated Beaver triple, we are referring to sampling a fresh Beaver triple (u, v, w), and then sampling the shares of the honest parties conditioned on their summation equaling the corresponding values. Note that the nPCF simulator $\mathsf{Sim}_{\mathsf{PCF}}$ gives perfectly simulated triples when noise indicator $\mathbb{I}_A$ gives $\mathbf{0}$.

The simulator $\mathsf{Sim}_{\mathsf{MPC}}$ works as follows:

- Invoke the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$ to obtain the output $y \leftarrow C(x)$.

Procedures in protocol $\pi_{\mathsf{MPC}}^{\mathsf{CorrectTriple}}$

Parameters: Number of parties N, the upper bound on corrupted parties t, noise rate ϵ, the number of triples for correction κ and the number of shares for security amplification β.
Setup: For $i \in [N]$, party i holds $\mathsf{nPCF}.K_i$, where $\{\mathsf{nPCF}.K_i\}_{i\in[N]}$ are generated by $\mathsf{nPCF}.\mathsf{Gen}(1^\lambda, 1^N, t, \epsilon)$. Each party holds a random oracle $\mathcal{O}$ such that the output of $\mathcal{O}$ follows the input distribution $\mathcal{X}$ of nPCF.

- $\mathsf{Xor}(\llbracket x \rrbracket, \llbracket y \rrbracket)$: For values $x, y \in \mathbb{F}_2$ whose shares the parties hold:
  - For $i \in [N]$, party i locally evaluates $\llbracket z \rrbracket_i = \llbracket x \rrbracket_i + \llbracket y \rrbracket_i$.
  - Let $\llbracket z \rrbracket$ be the result of computation.
- $\mathsf{And}(\llbracket x \rrbracket, \llbracket y \rrbracket)$: For values $x, y \in \mathbb{F}_2$ whose shares the parties hold:
  - Let $(\llbracket u \rrbracket, \llbracket v \rrbracket, \llbracket w \rrbracket)$ be the shares of a triple provided when invoking this procedure. For $i \in [N]$, party i evaluates $\llbracket \Delta_x \rrbracket_i = \llbracket x \rrbracket_i - \llbracket u \rrbracket_i$ and $\llbracket \Delta_y \rrbracket_i = \llbracket y \rrbracket_i - \llbracket v \rrbracket_i$.
  - Parties reconstruct the values Δ_x and Δ_y by collecting the shares.
  - For $i \in [N]$, party i locally evaluates $\llbracket z \rrbracket_i = \Delta_x \cdot \Delta_y + \Delta_x \cdot \llbracket v \rrbracket_i + \Delta_y \cdot \llbracket u \rrbracket_i + \llbracket w \rrbracket_i$, and sets $\llbracket z \rrbracket$ to be the result of computation.
- $\mathsf{Correct}()$:
  - Let cnt be a counter known by all parties, and increase cnt by κ after this procedure.
  - For $i \in [N]$, party i locally evaluates $\mathsf{nPCF}.\mathsf{Eval}(\mathsf{nPCF}.K_i, \mathcal{O}(\mathsf{cnt} + j))$ to obtain the share of triple $(\llbracket u^j \rrbracket_i, \llbracket v^j \rrbracket_i, \llbracket w^j \rrbracket_i)$ for $j \in [\kappa]$.
  - For $j \in [2, \kappa]$, parties invoke $\mathsf{And}(\llbracket u^1 \rrbracket, \llbracket v^1 \rrbracket)$ with Beaver triple $(\llbracket u^j \rrbracket, \llbracket v^j \rrbracket, \llbracket w^j \rrbracket)$ to obtain $\left\llbracket w'^j \right\rrbracket$, and let $\llbracket e^j \rrbracket = \left\llbracket w'^j \right\rrbracket - \llbracket w^1 \rrbracket$.
  - The parties reconstruct e^j for $j \in [2, \kappa]$. If $e^j = 0$ for all $j \in [2, \kappa]$, set $(\llbracket u^1 \rrbracket, \llbracket v^1 \rrbracket, \llbracket w^1 \rrbracket)$ to be the result of this procedure. Otherwise, rerun this procedure from the beginning.
- $\mathsf{SecurityAmplify}()$:
  - Parties locally sample random values to obtain random shares $(\llbracket a^1 \rrbracket, \cdots, \llbracket a^\beta \rrbracket)$, $(\llbracket b^1 \rrbracket, \cdots, \llbracket b^\beta \rrbracket)$.
  - For $i, j \in [\beta]$, parties invoke $\mathsf{Correct}()$ to obtain triple $(\llbracket u^{i,j} \rrbracket, \llbracket v^{i,j} \rrbracket, \llbracket w^{i,j} \rrbracket)$ and invoke $\mathsf{And}(\llbracket a^i \rrbracket, \llbracket b^j \rrbracket)$ with triple $(\llbracket u^{i,j} \rrbracket, \llbracket v^{i,j} \rrbracket, \llbracket w^{i,j} \rrbracket)$ to get $\llbracket c^{i,j} \rrbracket$.
  - Parties locally evaluate $\llbracket a \rrbracket = \sum_{i\in[\beta]} \llbracket a^i \rrbracket$, $\llbracket b \rrbracket = \sum_{i\in[\beta]} \llbracket b^i \rrbracket$, $\llbracket c \rrbracket = \sum_{i,j\in[\beta]} \llbracket c^{i,j} \rrbracket$. Let $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ be the result of this procedure.

Figure 7: Procedures in MPC with noiseless triples

- For Correct procedure, simulate the honest parties by simulating the evaluation of nPCF as described in the simulator $\mathsf{Sim}_{\mathsf{PCF}}$ of nPCF. If all triples from nPCF are noiseless in one procedure, i.e. the noise indicator $\mathbb{I}_A$ of nPCF always outputs $\mathbf{0}$, $\mathsf{Sim}_{\mathsf{MPC}}$ replaces the result

Protocol $\pi_{\mathsf{MPC}}^{\mathsf{CorrectTriple}}$

Parameters: Number of parties N, upper bound on corrupted parties t, noise rate ϵ, the number of triples for correction κ and the number of shares for security amplification β.
Parties evaluate the circuit gate by gate:

- For a Xor gate with inputs x, y, let $[\![x]\!], [\![y]\!]$ be the corresponding shares. Evaluate $[\![z]\!] = \mathsf{Xor}([\![x]\!], [\![y]\!])$ to obtain the result of this Xor gate.
- For an And gate with inputs x, y, let $[\![x]\!], [\![y]\!]$ be the corresponding shares. Parties invoke $\mathsf{SecurityAmplify}()$ to obtain triple $([\![u]\!], [\![v]\!], [\![w]\!])$ and then evaluate $[\![z]\!] = \mathsf{And}([\![x]\!], [\![y]\!])$ with this triple to obtain the result of this And gate.

For each output with corresponding share $[\![x]\!]$, parties directly reconstruct this share to receive the final result of computation.

Figure 8: MPC protocol with noiseless triples

triple with a perfectly simulated Beaver triple correlation.

- For SecurityAmplify procedure, Sim replaces the share of the result triple with a perfectly simulated Beaver triple correlation.
- For And procedure and Xor procedure in the gate-by-gate evaluation, replace the messages from honest parties with random values uniformly sampled in $\mathbb{F}_2$.
- For the final output with the corresponding share $[\![x]\!]$, $\mathsf{Sim}_{\mathsf{MPC}}$ replaces it with a perfectly simulated share, i.e., it samples the honest parties' shares uniformly, conditioned on the value inside this share being x, and allows the parties to reconstruct the share instead.

We prove that the view of the adversary $\mathcal{A}$ in the real world execution is indistinguishable from the view in the ideal world execution by invoking hybrid arguments.

- Hyb_0: This hybrid is identical to the real world execution.
- Hyb_1: This hybrid is identical to Hyb_0, except that the shares of Beaver triples generated by $\mathsf{nPCF.Eval}(K_i, \cdot)$ for $i \in \bar{T}$ are replaced with shares generated by simulator $\mathsf{Sim}_{\mathsf{PCF}}$ in the experiment of nPCF. The indistinguishability between Hyb_0 and Hyb_1 follows from the (N, t)-*weak pseudorandomness* of nPCF.
- Hyb_2: This hybrid is identical to Hyb_1 except that for Correct procedure executions in which all triples sampled from nPCF are noiseless, i.e. the noise indicator $\mathbb{I}_A$ outputs $\mathbf{0}$, we replace the result triples of these Correct procedures with perfectly simulated Beaver triple correlations.

  It is not hard to observe that Hyb_2 is actually the same as Hyb_1 by viewing $([\![u^j]\!], [\![v^j]\!])$ for $j \in [2, \kappa]$ as a one-time pad for the perfect Beaver triple correlation $([\![u^1]\!], [\![v^1]\!], [\![w^1]\!])$.
- Hyb_3: This hybrid follows Hyb_2, except that we replace the result triples of SecurityAmplify procedures with perfectly simulated Beaver triple correlations.

  According to Claim 7.1, the output of SecurityAmplify should be the share of a correct Beaver triple with overwhelming probability. Thus, to show the indistinguishability between Hyb_2

and Hyb_3, we are left to demonstrate that the shares and the values of the resulting triple are randomly sampled from the adversary's perspective, with overwhelming probability.

For each SecurityAmplify procedure, we attempt to find an index i such that the triples involved in the $\mathsf{And}(\llbracket a^i \rrbracket, \llbracket b^j \rrbracket)$ procedure for $j \in [\beta]$ are all perfectly simulated Beaver triple correlations. Similarly, we attempt to find an index j such that the triples involved in the $\mathsf{And}(\llbracket a^i \rrbracket, \llbracket b^j \rrbracket)$ procedure for $i \in [\beta]$ are all perfectly simulated Beaver triple correlations.

Note that a triple correlation from Correct will be a perfectly simulated Beaver triple correlation with probability at least $1 - \kappa \cdot \epsilon \geq 1 - \frac{\kappa}{\lambda}$, so the failure rate in finding such indices i and j is bounded by $2\left(\frac{\kappa \cdot \beta}{\lambda}\right)^{\beta}$, a negligible function in λ.

By the property of perfectly simulated Beaver triple correlations, $a^i, b^j, c^{i,j}$, together with their shares held by the honest parties, are random from the adversary's perspective. This implies that the output $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ appears random to the adversary, thus forming a perfectly simulated Beaver triple correlation.

- Hyb_4: This hybrid is identical to Hyb_3, except that in the And procedure during the gate-by-gate evaluation, the messages from honest parties are replaced with random values uniformly sampled from $\mathbb{F}_2$, and the output phase is simulated as described in the simulator $\mathsf{Sim}_{\mathsf{MPC}}$.

  The Beaver triples used in these And procedures are perfectly simulated Beaver triple correlations generated by the SecurityAmplify procedure. By treating the Beaver triples as one-time pads, it follows that all messages from honest parties are uniformly random. Additionally, by examining the And procedure, we can observe that the shares of the computation result held by honest parties will be uniformly random when using perfectly simulated Beaver triple correlations. Consequently, the shares held by honest parties involved in the Output reconstruction will be uniformly random conditioned on their summation. Therefore, Hyb_4 is indistinguishable from Hyb_3.

  Note that Hyb_4 is the ideal world execution, thus we complete the proof with this hybrid argument.

Finally, the computation and communication costs amount to at most $O(\kappa \cdot \beta^2)$ invocations of the And and Xor procedures, together with $\kappa \cdot \beta^2$ evaluations of nPCF. □

7.2 MPC Protocol via Fault Tolerance

Here we present another solution that leverages majority functions for fault tolerance, as shown in Figure 9, 10. In this protocol, for each value on the wire, the parties hold κ shares corresponding to that value, and we guarantee that the majority of the shares are correct. After each And procedure, a majority function is applied to correct the errors introduced by the noisy triples. The majority function is also implemented using noisy triples; however, as long as the noise rate remains sufficiently low, we can ensure that the majority circuit will still function correctly. As in the construction of Figure 8, secret reconstruction is performed by designating one party to collect all shares and output the result.

Theorem 7.2. *Let $N, t, \epsilon, \kappa, \beta$ be functions of the security parameter λ, where $N, t \in [0, N-1]$ are bounded by polynomials in λ, $\epsilon \leq \frac{1}{\lambda}$, and $\kappa \in \omega(1) \cap O(\log \lambda), \beta \in \omega(1)$. We assume the existence of an (N, t)-noisy PCF for Beaver triples in $\mathbb{F}_2$ with noise rate ϵ, and that* nPCF *keys have been distributed to enable repeated protocol evaluations. Let $\mathcal{X}$ be the input distribution required by* nPCF*, and let $\mathcal{O}$ be a random oracle whose output follows $\mathcal{X}$. The protocol π_{MPC}, as described in Figure 10, with parameters*

Procedures in protocol $\pi_{\mathsf{MPC}}^{\mathsf{FaultTolerance}}$

Parameters: Number of parties N, upper bound on corrupted parties t, noise rate ϵ, the number of repetitions κ and the repetition parameter for the Output procedure β.
Setup: For $i \in [N]$, party i holds $\mathsf{nPCF}.K_i$, where $\{\mathsf{nPCF}.K_i\}_{i\in[N]}$ are generated by $\mathsf{nPCF.Gen}(1^\lambda, 1^N, t, \epsilon)$. Each party holds a random oracle $\mathcal{O}$ whose output follows the input distribution $\mathcal{X}$ of nPCF.

- Init: For the shares of an input $x \in \{0,1\}^{\ell_{\text{in}}}$, parties copy the shares κ times: $[\![x^k]\!]_i = [\![x]\!]_i$ for $k \in [\kappa]$.
- $\mathsf{Xor}([\![x]\!], [\![y]\!])$: For values $x, y \in \mathbb{F}_2$ whose shares the parties hold:
  - For $i \in [N]$, party i locally evaluates $[\![z]\!]_i = [\![x]\!]_i + [\![y]\!]_i$.
  - Let $[\![z]\!]$ be the result of computation.
- $\mathsf{And}([\![x]\!], [\![y]\!])$: For values $x, y \in \mathbb{F}_2$ whose shares the parties hold:
  - Let cnt be a counter known by all parties, and increase cnt by one after this procedure.
  - For $i \in [N]$, party i locally evaluates $\mathsf{nPCF.Eval}(\mathsf{nPCF}.K_i, \mathcal{O}(\mathsf{cnt}))$ to obtain the share of triple $([\![u]\!]_i, [\![v]\!]_i, [\![w]\!]_i)$. Then, party i evaluates $[\![\Delta_x]\!]_i = [\![x]\!]_i - [\![u]\!]_i$ and $[\![\Delta_y]\!]_i = [\![y]\!]_i - [\![v]\!]_i$.
  - Parties reconstruct the values Δ_x and Δ_y by collecting the shares.
  - For $i \in [N]$, party i locally evaluates $[\![z]\!]_i = \Delta_x \cdot \Delta_y + \Delta_x \cdot [\![v]\!]_i + \Delta_y \cdot [\![u]\!]_i + [\![w]\!]_i$, and sets $[\![z]\!]$ to be the result of computation.
- $\mathsf{Reshare}([\![x^1]\!], \cdots, [\![x^\kappa]\!])$: For κ shares held by parties,
  - Let MAJ be a Boolean circuit consisting of And gates and Xor gates that evaluates the majority function of κ inputs.
  - Parties evaluate $\mathsf{MAJ}([\![x^1]\!], \cdots, [\![x^\kappa]\!])$ for κ times, by invoking the And and Xor procedures as above, to obtain $[\![z^1]\!], \cdots, [\![z^\kappa]\!]$.
  - Let $([\![z^1]\!], \cdots, [\![z^\kappa]\!])$ be the κ shares of the computation result.
- $\mathsf{Output}([\![x^1]\!], \cdots, [\![x^\kappa]\!])$: For κ shares held by parties,
  - Iteratively invoke the Reshare procedure on $([\![x^1]\!], \cdots, [\![x^\kappa]\!])$ for β times to obtain $([\![z^1]\!], \cdots, [\![z^\kappa]\!])$.
  - Parties reconstruct the shares and locally compute the majority of κ values, setting it to be the result.

Figure 9: Procedures in fault-tolerant MPC protocol

Protocol $\pi_{\mathsf{MPC}}^{\mathsf{FaultTolerance}}$

Parameters: Number of parties N, threshold t, noise rate ϵ, the number of repetitions κ and the parameter for the Output procedure β.
Parties first invoke the Init procedure. Then parties evaluate the circuit gate by gate:

- For a Xor gate with inputs x, y, let $\left\{\left[\!\left[x^k\right]\!\right]\right\}_{k\in[\kappa]}, \left\{\left[\!\left[y^k\right]\!\right]\right\}_{k\in[\kappa]}$ be the corresponding shares. Evaluate $\left[\!\left[z^k\right]\!\right] = \mathsf{Xor}(\left[\!\left[x^k\right]\!\right], \left[\!\left[y^k\right]\!\right])$ for $k \in [\kappa]$. Then invoke $\mathsf{Reshare}(\left[\!\left[z^1\right]\!\right], \cdots, \left[\!\left[z^\kappa\right]\!\right])$ to obtain the result of this Xor gate.
- For an And gate with inputs x, y, let $\left\{\left[\!\left[x^k\right]\!\right]\right\}_{k\in[\kappa]}, \left\{\left[\!\left[y^k\right]\!\right]\right\}_{k\in[\kappa]}$ be the corresponding shares. Evaluate $\left[\!\left[z^k\right]\!\right] = \mathsf{And}(\left[\!\left[x^k\right]\!\right], \left[\!\left[y^k\right]\!\right])$ for $k \in [\kappa]$. Then invoke $\mathsf{Reshare}(\left[\!\left[z^1\right]\!\right], \cdots, \left[\!\left[z^\kappa\right]\!\right])$ to obtain the result of this And gate.

For the output with corresponding shares $\left\{\left[\!\left[x^k\right]\!\right]\right\}_{k\in[\kappa]}$, parties invoke $\mathsf{Output}(\left[\!\left[x^1\right]\!\right], \cdots, \left[\!\left[x^\kappa\right]\!\right])$ to receive the result of computation.

Figure 10: Fault-tolerant MPC protocol

$N(\lambda), t(\lambda), \epsilon(\lambda), \kappa(\lambda), \beta(\lambda)$, *securely instantiates the functionality* $\mathcal{F}_{\mathsf{MPC}}$ *with semi-honest security against adversaries capable of corrupting at most* t *parties, with per-gate, per-party computation cost* $\mathsf{poly}(\kappa) \cdot \mathsf{Time}(\mathsf{nPCF.Eval})$ *and communication cost* $\mathsf{poly}(\kappa)$.

Proof. We first focus on the correctness of this protocol. Correctness is directly implied by the following claim:

Claim 7.2. *There exists a negligible function* negl *such that for any* $\lambda \in \mathbb{N}$, *with probability* $1 - \mathsf{negl}(\lambda)$, *for any wire in the circuit* C, *let* y *be the value on the wire on input* x *and let* $\left[\!\left[y^1\right]\!\right], \cdots, \left[\!\left[y^\kappa\right]\!\right]$ *be the corresponding shares in the protocol, the number of* $k \in [\kappa]$ *such that* $y^k \neq y$ *is bounded by* $\lceil \kappa/2 \rceil - 1$.

Proof. We prove the claim by induction, proceeding in topological order, gate by gate.

For any gate, let x and y denote the values on the input wires. Assume the corresponding shares $\{\left[\!\left[x^k\right]\!\right]\}_{k\in[\kappa]}$ and $\{\left[\!\left[y^k\right]\!\right]\}_{k\in[\kappa]}$ are correct, except for at most $\left\lceil \frac{\kappa}{6} \right\rceil - 1$ shares on each wire.

Next, we examine the share of the computation result $\left\{\left[\!\left[z^k\right]\!\right]\right\}_{k\in[\kappa]}$ before invoking the Reshare procedure. The value of $\left[\!\left[z^k\right]\!\right]$ will be erroneous only if $\left[\!\left[x^k\right]\!\right]$ or $\left[\!\left[y^k\right]\!\right]$ is incorrect, or if the gate is an And and at least one noisy Beaver triple is involved in the computation. Since the noise rate of nPCF is $\epsilon \leq \frac{1}{\lambda}$ and $\kappa \in \omega(1)$ by the choice of parameters, the probability that more than $\left\lceil \frac{\kappa}{6} \right\rceil - 1$ And procedures are affected by noisy Beaver triples is negligible, by the Chernoff bound.

Combining this with the assumption that the shares corresponding to x and y have at most $\left\lceil \frac{\kappa}{6} \right\rceil - 1$ errors, we conclude that with probability $1 - \mathsf{negl}$, the result of either the Xor or And procedure will contain no more than $\left\lceil \frac{\kappa}{2} \right\rceil - 1$ erroneous shares.

We now consider the Reshare procedure. If no more than $\left\lceil \frac{\kappa}{2} \right\rceil - 1$ shares are incorrect, the majority gate will yield a correct share, provided that all Beaver triples involved are noiseless. Since $\kappa \in O(\log \lambda)$ by the choice of parameters, there exists a construction for the majority function such that the number of And procedures involved is in $\mathsf{poly}(\kappa) \in \mathsf{poly}\log(\lambda)$, and the number of Beaver triples involved is similarly polynomial, as shown in [Val84]. Again, by the Chernoff bound, the probability that more than $\left\lceil \frac{\kappa}{6} \right\rceil - 1$ And procedures are affected by noisy Beaver triples is negligible. This completes the induction step.

□

Thus, our protocol correctly implements the functionality $\mathcal{F}_{\mathsf{MPC}}$.

Then, we construct a simulator $\mathsf{Sim}_{\mathsf{MPC}}$ to simulate the behavior of honest parties. Recall the notation used in the proof of Theorem 7.1: Let T represent the corrupted parties and $\bar{T}$ denote the honest parties. We say that a Beaver triple correlation is perfectly simulated if there exists a fresh Beaver triple (u, v, w), and the shares held by the honest parties are uniformly sampled, conditioned on the shared triple being (u, v, w).

The simulator $\mathsf{Sim}_{\mathsf{MPC}}$ works as follows:

- Invoke the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$ to obtain the output $C(x)$.
- For procedures before Output, Sim simulates the behavior of the honest parties by replacing all the messages from the honest parties with uniformly random values in $\mathbb{F}_2$.
- For the Output procedure, $\mathsf{Sim}_{\mathsf{MPC}}$ first simulates the shares of Beaver triples held by honest parties by invoking the simulator $\mathsf{Sim}_{\mathsf{PCF}}$ in the (N, t)-*weak pseudorandomness* experiment of nPCF.

  Next, among the β layers of the Reshare procedure, $\mathsf{Sim}_{\mathsf{MPC}}$ attempts to identify a Reshare procedure such that all Beaver triples used in the procedure are perfectly simulated. The result of this Reshare procedure is set to be κ perfectly simulated shares of $C(x)$, i.e., the shares held by the honest parties are randomly sampled, conditioned on their sum with the shares held by the corrupted parties equaling $C(x)$.

  For the part of the Output procedure before the specific Reshare procedure, Sim simulates the behavior of the honest parties simply by replacing all messages from the honest parties with random values. For the part of the Output procedure after that specific Reshare procedure, Sim follows the protocol honestly, starting with the perfectly simulated shares of $C(x)$ as described above, except that the Beaver triples are simulated using $\mathsf{Sim}_{\mathsf{PCF}}$.

We prove that the view of the adversary $\mathcal{A}$ in the real world execution is indistinguishable from its view in the ideal world execution by invoking hybrid arguments.

- Hyb_0: This hybrid is identical to the real world execution.
- Hyb_1: This hybrid is identical to Hyb_0, except that shares of Beaver triples generated by $\mathsf{nPCF.Eval}(K_i, \cdot)$ for $i \in \bar{T}$ are replaced with Beaver triples simulated by $\mathsf{Sim}_{\mathsf{PCF}}$. The indistinguishability between Hyb_0 and Hyb_1 is implied by the weak pseudorandomness of nPCF.
- Hyb_2: In this hybrid, we will try to find the specific Reshare procedure in which all Beaver triples involved are perfectly simulated, i.e., the noise indicator $\mathbb{I}_A$ outputs $\mathbf{0}$. The hybrid Hyb_2 is identical to Hyb_1, except that the result of this specific Reshare procedure is replaced with perfectly simulated shares of $C(x)$, as described in the simulator $\mathsf{Sim}_{\mathsf{MPC}}$.

  The noise rate is $\epsilon \leq \frac{1}{\lambda}$, and the total number of Beaver triples involved in any given Reshare procedure is $\mathsf{poly}(\log \lambda)$, as discussed in the proof of Claim 7.2. Since there are $\beta \in \omega(1)$ Reshare procedures in the Output procedure, the probability that noisy Beaver triples are involved in every Reshare procedure is negligible. By Claim 7.2, with probability $1 - \mathsf{negl}$, the correct result of a majority function should be $C(x)$. Therefore, as long as the Beaver triples are noiseless, with probability $1 - \mathsf{negl}$, the result of a Reshare procedure will be perfectly simulated shares of $C(x)$, due to the property of those perfectly simulated triples. This implies that Hyb_2 is statistically close to Hyb_1.

- Hyb_3: This hybrid is identical to Hyb_2 except that all messages from honest parties before the specific Reshare procedure are replaced with random values uniformly sampled in $\mathbb{F}_2$. Note that this hybrid is identical to the ideal world execution.

  To show that Hyb_3 is indistinguishable from Hyb_2, we introduce a series of more detailed hybrids. Let m denote the number of And procedures invoked throughout the protocol, and let $\mathsf{Hyb}_{2,j}$ be identical to Hyb_2, except that the messages from honest parties in the last j And procedures are replaced with random values for $j \in [0, m]$. Clearly, $\mathsf{Hyb}_{2,0}$ is identical to Hyb_2, and since all messages from honest parties before the reconstruction of the final output are within And procedures, it follows that $\mathsf{Hyb}_{2,m}$ is equivalent to Hyb_3. Therefore, to prove that Hyb_3 is indistinguishable from Hyb_2, it suffices to demonstrate the indistinguishability between $\mathsf{Hyb}_{2,j}$ and $\mathsf{Hyb}_{2,j-1}$ for each j.

  We now show that $\mathsf{Hyb}_{2,j}$ and $\mathsf{Hyb}_{2,j-1}$ are indeed identical. For the j-th-from-last And procedure, since all messages from honest parties after this procedure, but before the specific Reshare procedure, are random, and the messages after the specific Reshare procedure are completely independent of previous information, we conclude that the information about this Beaver triple (u, v, w) in the current And procedure remains concealed. Since u and v are uniformly distributed, independent of whether the triple contains noise, the messages from honest parties will also be uniformly distributed. Thus, $\mathsf{Hyb}_{2,j} = \mathsf{Hyb}_{2,j-1}$.

With these hybrid arguments, we have shown that the real world execution is indistinguishable from the ideal world execution, which completes the proof. □

8 Concrete Efficiency

To better illustrate the concrete efficiency of our nPCF scheme and the resulting MPC protocol, we analyze in this section the number of bit operations required for a single nPCF evaluation and, subsequently, for a single-gate MPC evaluation.

Recall that in the nPCF construction (Figure 2), each evaluation involves the following steps:

- Two inner products between a k-sparse vector and the secret vector, and one inner product between a k^2-sparse vector and the secret vector.
- Noise sampling from $\mathcal{D}_e^3(\eta)$.
- A PRF evaluation.

In the most straightforward implementation, computing the inner products requires $k^2 + 2k$ bitwise AND operations, along with the same number of bitwise XOR operations. Moreover, in the honest-majority setting, the expensive inner product involving a k^2-sparse vector and the secret can be avoided altogether. In this case, the computation reduces to just $2k + 1$ operations over the extension field, yielding a substantial efficiency improvement.

As discussed in Section 5.1, noise sampling only requires $\mathsf{poly}\log(\lambda)$ random bits and the same order of operations to sample λ error terms, resulting in an amortized $\widetilde{O}(1/\lambda)$ overhead per evaluation. Hence, this cost can be ignored in the overall operation count.

For the PRF evaluation, we leave its cost unspecified for two reasons. First, the efficiency varies significantly depending on the chosen implementation. Second, as noted earlier, in the semi-honest MPC setting the PRF call can be entirely avoided by letting each party directly sample the required random bits.

$\epsilon_{\mathsf{MPC}} \backslash \epsilon$	2^{-10}	2^{-12}	2^{-14}	2^{-16}
2^{-30}	$100(\kappa = 4, \beta = 5)$	$48(\kappa = 3, \beta = 4)$	$27(\kappa = 3, \beta = 3)$	$18(\kappa = 2, \beta = 3)$
2^{-40}	$245(\kappa = 5, \beta = 7)$	$100(\kappa = 4, \beta = 5)$	$48(\kappa = 3, \beta = 4)$	$27(\kappa = 3, \beta = 3)$
2^{-50}	$726(\kappa = 6, \beta = 11)$	$245(\kappa = 5, \beta = 7)$	$100(\kappa = 4, \beta = 5)$	$64(\kappa = 4, \beta = 4)$
2^{-60}	$1372(\kappa = 7, \beta = 14)$	$486(\kappa = 6, \beta = 9)$	$245(\kappa = 5, \beta = 7)$	$100(\kappa = 4, \beta = 5)$

Table 2: Estimated values of $\beta^2 \cdot \kappa$ for various target error rates ϵ_{MPC} (rows) and noisy PCF error probabilities ϵ (columns).

For the MPC protocol $\pi_{\mathsf{MPC}}^{\mathsf{CorrectTriple}}$ (Figure 8), the following operations are required for evaluating a single AND gate:

- $\beta^2 + \beta^2 \cdot \kappa + 1$ *AND procedures*, arising from the invocations of the SecurityAmplify and Correct subroutines. Each AND procedure consists of two share reconstructions and three local AND operations.
- $\beta^2 \cdot \kappa$ calls to nPCF evaluations. Each nPCF evaluation further requires k calls to the random oracle in order to generate its input.

Recall that β denotes the number of shares used in SecurityAmplify, while κ is the number of triples used in Correct procedures. Thus, ignoring lower-order terms, the total computation per gate per party is $k^2 \cdot \beta^2 \cdot \kappa$ bitwise operations and $\beta^2 \cdot \kappa$ random oracle calls. Let ϵ denote the noise rate of nPCF. To ensure that the per-gate error rate does not exceed ϵ_{MPC}, the parameters β, κ, and ϵ must satisfy the following conditions:

- $(\epsilon/2)^\kappa \cdot \beta^2 \leq \epsilon_{\mathsf{MPC}}$, guaranteeing correctness of the β^2 triples employed in SecurityAmplify.
- $2 \cdot (\epsilon/2 \cdot \kappa \cdot \beta)^\beta \leq \epsilon_{\mathsf{MPC}}$, ensuring the security of the procedure.

The additional factor of $1/2$ in front of ϵ reflects a property of our construction: a noisy Beaver triple is incorrect with probability $\epsilon/2$, whereas this need not hold in general.

Table 2 gives concrete operation counts for the targeted error rate ϵ_{MPC} and nPCF noise rate.

The focus of this work is primarily theoretical, and we leave a careful concrete security analysis of Sparse LPN—together with a systematic implementation study of our protocol—for future work in order to fully assess its practical efficiency. In this paper, our goal is not to provide precise benchmarks, but rather to offer intuition about the potential efficiency of our approach. To that end, we consider two natural settings of interest and provide rough ballpark estimates of the storage requirements and the number of bitwise operations per gate. These estimates are illustrative only, highlighting how the asymptotic improvements in our design could plausibly translate into meaningful gains in practice once concrete implementations are developed. In particular, the examples below use 80-bit Sparse-LPN security only as an illustrative target, rather than as recommended deployment parameters; targeting 128-bit security would require larger Sparse-LPN parameters and correspondingly higher storage and computation.

Case Study 1: 5-Party MPC with Target Per-Gate Error 2^{-30}. As a first illustrative example, we consider a small-scale deployment with 5 parties. We target a per-gate error probability of 2^{-30}, which ensures a negligible overall failure rate for circuits of moderate size. Using the parameter analysis in Table 2, together with the attack landscape surveyed in the concurrent

work [BCM+25], we instantiate the parameters $(k, n, \varepsilon, \beta, \kappa, \epsilon)$ to satisfy both correctness and 80-bit security requirements while keeping the computational overhead within a practical range.

In this setting, the chosen parameters are $k = 30$, $n \approx 3.1 \times 10^6$, $\varepsilon = 2^{-10}$, $\beta = 5$, and $\kappa = 4$. These values yield approximately 100 noisy PCF evaluations per multiplication gate and allow the noisy PCF to support at least 2^{60} independent evaluations. Each evaluation requires about 960 operations and 690 random bits for sampling coefficient vectors, which can be generated using random oracle calls or shared random tapes. Taken together, this leads to a per-gate cost of roughly 96,000 operations, alongside about 69,000 random bits for the corresponding randomness generation. The communication is 2 bits per underlying multiplication per party. Since each multiplication gate uses $\beta^2 + \kappa\beta^2 + 1 = 126$ such multiplications, the total communication is 252 bits per party per gate.

The dominant overhead in this setting, however, is storage. Each party must store approximately n^2 tensor-product terms, leading to a storage requirement of about 1.118 terabytes per party.

The preceding estimates are for the 5-party setting. Separately, the same target error rate can also illustrate the constant-fraction extension: in a 1000-party setting with at most 800 corruptions, as shown in Section 6.1 , we can incorporate a PRZS layer into the outputs of nPCF so that the seed size no longer grows with N. In this 1000-party constant-fraction setting, our PRZS construction requires approximately 130 PRF calls (yielding 3 bits per call) per nPCF evaluation, which amounts to an additional 39,000 random bits generated via random oracle queries.

If we further assume that the circuit is resilient to random leakages occurring with probability 2^{-9}, the performance improves by a factor of β^2, reducing the cost to just 3,840 operations and 4,320 random bits per gate. In this leakage-resilient setting, we can set $\beta = 1$, so the communication drops to 12 bits per party per gate.

Although these figures are only rough ballpark estimates, they indicate that our construction can achieve practically reasonable computational performance in small-party scenarios, though at the expense of substantial storage requirements.

Case Study 2: 1000-Party MPC with Honest Majority and Target Per-Gate Error 2^{-30}. As a second example, we examine a large-scale setting with 1000 parties under an honest-majority assumption. We again fix the per-gate error probability to 2^{-30}, which guarantees negligible error accumulation for circuits of substantial size. Following the same parameter analysis, we instantiate $(k, n, \varepsilon, \beta, \kappa, \epsilon)$ to balance correctness, security, and efficiency.

Here, we obtain parameter values of $k = 20$, $n \approx 10^8$, $\varepsilon = 2^{-16}$, $\beta = 3$, and $\kappa = 2$. These parameters yield 18 noisy PCF evaluations per multiplication gate, while still supporting at least 2^{60} independent evaluations. Each evaluation requires about 40 field operations in the extension field $\mathbb{F}_{2^{10}}$, 740 random bits from random-oracle calls, and one PRZS evaluation. For the PRZS construction, each party performs about 28 PRF calls (each of length 30 bits). Aggregating these costs, the total per-gate requirement is about 720 field operations and 28,440 random bits from random oracle queries. If the evaluated circuit is resilient to leakages with probability 2^{-16}, we can set $\beta = 1$, yielding a $9\times$ performance improvement and reducing the cost to just 80 operations and 3,160 random bits per gate. The communication is again 2 bits per underlying multiplication per party. Since each multiplication gate uses $\beta^2 + \kappa\beta^2 + 1 = 28$ such multiplications, the total communication is 56 bits per party per gate. In the leakage-resilient setting with $\beta = 1$, this drops to 8 bits per party per gate.

The per-party storage in this case is about 240 MB, representing a substantial improvement compared to the terabyte-scale overhead of the 5-party setting. This reduction is primarily due to the honest-majority optimizations of Section 6.3 .

While still approximate, these estimates suggest that our approach scales favorably even in the thousand-party regime. The honest-majority assumption plays a key role in keeping both computation and storage within feasible bounds, highlighting the potential practicality of our protocol in such settings.

Acknowledgments. Huijia Lin was supported in part by the Simons Collaboration on the Theory of Algorithmic Fairness and an Amazon Research Award. The work of Aayush Jain and Nuozhou Sun was supported by an NSF CAREER Award, an Alfred P. Sloan Fellowship, QCIT Center funding at CMU, a Stellar Foundation Grant, and an Amazon Research Award. Aayush Jain was additionally supported by national funds through FCT — Fundação para a Ciência e a Tecnologia, I.P., under project 2024.12595.CMU and the LASIGE Research Unit (UID/00408/2025). We thank the anonymous TCC reviewers for their helpful comments and suggestions.

9 References

[ABW10] Benny Applebaum, Boaz Barak, and Avi Wigderson. Public-key cryptography from different assumptions. In Leonard J. Schulman, editor, *42nd ACM STOC*, pages 171–180. ACM Press, June 2010.

[AIK08] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. On pseudorandom generators with linear stretch in nc^0. *Comput. Complex.*, 17(1):38–69, 2008.

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *44th FOCS*, pages 298–307. IEEE Computer Society Press, October 2003.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 535–548. ACM Press, November 2013.

[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, Heidelberg, March 2022.

[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, Heidelberg, August 2023.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Heidelberg, August 2019.

[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.

[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, Heidelberg, August 2020.

[BCG+22] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 603–633. Springer, Heidelberg, August 2022.

[BCG+23] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Oblivious transfer with constant computational overhead. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 271–302. Springer, Heidelberg, April 2023.

[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.

[BCM+24] Dung Bui, Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Fast public-key silent OT and more from constrained Naor-Reingold. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 88–118. Springer, Heidelberg, June 2024.

[BCM+25] Lennart Braun, Geoffroy Couteau, Kelsey Melissaris, Mahshid Riahinia, and Elahe Sadeghi. Fast pseudorandom correlation functions from sparse LPN. Cryptology ePrint Archive, Paper 2025/1644, 2025.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.

[BIK+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1175–1191. ACM Press, October / November 2017.

[CD23] Geoffroy Couteau and Clément Ducros. Pseudorandom correlation functions from variable-density LPN, revisited. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part II*, volume 13941 of *LNCS*, pages 221–250. Springer, Heidelberg, May 2023.

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 342–362. Springer, Heidelberg, February 2005.

[DIJL23] Quang Dao, Yuval Ishai, Aayush Jain, and Huijia Lin. Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 315–348. Springer, Heidelberg, August 2023.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

[EGP+23] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, Yifan Song, and Chenkai Weng. SuperPack: Dishonest majority MPC with constant online communication. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2023.

[EKM17] Andre Esser, Robert Kübler, and Alexander May. LPN decoded. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 486–514. Springer, Heidelberg, August 2017.

[ER60] Paul Erd6s and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5:17–61, 1960.

[Fei02] Uriel Feige. Relations between average case complexity and approximation complexity. In *34th ACM STOC*, pages 534–543. ACM Press, May 2002.

[FLY22] Zhiyuan Fan, Jiatu Li, and Tianqi Yang. The exact complexity of pseudorandom functions and the black-box natural proof barrier for bootstrapping results in computational complexity. In Stefano Leonardi and Anupam Gupta, editors, *54th ACM STOC*, pages 962–975. ACM Press, June 2022.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 465–482. Springer, Heidelberg, April 2012.

[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, Heidelberg, May 2014.

[GLO+21] Vipul Goyal, Hanjun Li, Rafail Ostrovsky, Antigoni Polychroniadou, and Yifan Song. ATLAS: Efficient and scalable MPC in the honest majority setting. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 244–274, Virtual Event, August 2021. Springer, Heidelberg.

[GMPW20] Nicholas Genise, Daniele Micciancio, Chris Peikert, and Michael Walter. Improved discrete gaussian and subgaussian analysis for lattice cryptography. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 623–651. Springer, Heidelberg, May 2020.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GR12] Shafi Goldwasser and Guy N. Rothblum. How to compute in the presence of leakage. In *53rd FOCS*, pages 31–40. IEEE Computer Society Press, October 2012.

[HOSS18] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. TinyKeys: A new approach to efficient multi-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 3–33. Springer, Heidelberg, August 2018.

[HR22] Justin Holmgren and Ron D. Rothblum. Faster sounder succinct arguments and IOPs. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 474–503. Springer, Heidelberg, August 2022.

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 433–442. ACM Press, May 2008.

[IS24] Yuval Ishai and Yifan Song. Leakage-tolerant circuits. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part IV*, volume 14654 of *LNCS*, pages 196–225. Springer, Heidelberg, June 2024.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, August 2003.

[KMA+21] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary

Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zaixiang He, Zhouyuan Huo, Pooria Javidbakht, Tara Javidi, Gauri Joshi, Mohammad Mahdi Kamani, Cédric Koumchatzky, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Liwei Wang, Zhiwei Steven Wu, Felix X. Yu, Han Zhao, Zheng Xu, Michael Zhang, Kai Zheng, and Chaochao Zhou. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1–2):1–210, 2021.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 158–189. Springer, Heidelberg, April / May 2018.

[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, *Advances in Cryptology - ASIACRYPT 2025 - 31st International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, VIC, Australia, December 8-12, 2025, Proceedings, Part VII*, Lecture Notes in Computer Science, pages 474–504. Springer, 2025.

[OMPR05] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A side-channel analysis resistant description of the AES s-box. In *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 413–423. Springer, 2005.

[Pip85] Nicholas Pippenger. On networks of noisy gates. In *26th FOCS*, pages 30–38. IEEE Computer Society Press, October 1985.

[Val84] L.G Valiant. Short monotone formulae for the majority function. *Journal of Algorithms*, 5(3):363–366, 1984.

[VN56a] John Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Bródy & Vámos*, 1956.

[VN56b] John Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, 34(34):43–98, 1956.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.