CAVERN: Efficient Honest-Majority Maliciously Secure $(2+1)$-PC for $\mathbb{Z}_{2^n}$ via DPF

Yang Liu
ShanghaiTech University

Liang Feng Zhang
ShanghaiTech University

***Abstract*—We introduce CAVERN, a new maliciously secure $(2+1)$-PC protocol for efficient piecewise polynomial (i.e., spline) evaluation on additively secret shared inputs over the ring $\mathbb{Z}_{2^n}$ in the preprocessing model, where parties obtain input-independent correlated randomness in an offline phase, which they then use to run an efficient protocol in the input-dependent online phase. This $(2+1)$ party structure can alternatively be instantiated between two parties with the aid of a (possibly untrusted) dealer. At the technical level, we introduce a new primitive called verifiable incremental distributed point function (VIDPF) and build on a novel combination of the VIDPF and authenticated secret sharing, providing an efficient method to detect the malicious behavior of the dealer or one of the parties. We implement and benchmark our protocol against the state-of-the-art semi-honest protocol GROTTO (CCS 2023), and the trusted-dealer-based maliciously secure 2PC protocol SHARK (S&P 2025). The results indicate that CAVERN only imposes a constant factor overhead on the top of GROTTO and SHARK, while providing stronger security guarantees.**

1. Introduction

Secure Multi-Party Computation (MPC) [1], [2] has solidified its position as a pivotal cryptographic paradigm, enabling collaborative computations on sensitive data without necessitating the revelation of individual private inputs. MPC is applicable in domains that routinely handle sensitive information, such as privacy-preserving machine learning (PPML) [3], secure financial computations [4] and collaborative data analytics [5]; see [6] for more current real-world use cases.

A major persistent challenge in the practical application of MPC, especially for protocols grounded in linear schemes, is the efficient evaluation of non-linear functions. Such non-linear functions are integral to many applications, including the activation functions (e.g., ReLU, sigmoid, tanh) that are interspersed between linear layers in neural networks, fundamental comparison operations, and basic mathematical functions like square root and logarithm.

To facilitate the need to evaluate non-linear functions in a linear setting, a variety of prior works [7], [8], [9], [10], [11], [12], [13], [14], [15] uses *piecewise-polynomial functions* (or *splines*) to approximate arbitrary non-linear functions on input additively shared secrets with a high degree of accuracy. However, almost all of these works achieve only *semi-honest security*, where the parties are assumed to follow the protocol steps but may attempt to learn additional information from the messages they observe, and are not *maliciously secure*, i.e., they do not guarantee security against *malicious* adversaries who may deviate arbitrarily from the protocol.

The two exceptions are PIKA [15] and SHARK [9]. PIKA provides an efficient approach to evaluate splines in the maliciously secure setting based in Distributed Point Function (DPF) [16]. However, PIKA's approach relies on the full-domain evaluation of DPFs, which incurs exponential computational cost and is only practical for small domains. SHARK is a maliciously secure scheme that uses Distributed Comparison Function [8] to evaluate splines and authenticated values to provide security against malicious behaviors. Unfortunately, SHARK still relies on a trusted dealer to provide input-independent preprocessing material, which is a significant limitation for practical applications.

Therefore, the above discovery urges us to propose efficient maliciously secure MPC protocols for evaluating splines without relying on the inefficient full-domain evaluation or a trusted dealer.

1.1. Our Contributions

Malicious Secure Protocols for Evaluating Splines. We present CAVERN, a new maliciously secure MPC protocol for evaluating splines over the ring $\mathbb{Z}_{2^n}$ in the $(2+1)$ party setting, which is a special subclass of 3-party computation. More precisely, in the $(2+1)$ party setting, one of the three parties is a dealer that only active in a data-independent preprocessing phase. To defend against the malicious adversary that can corrupt the dealer or one of the two online parties, we introduce a new primitive called verifiable incremental distributed point function (VIDPF) and proposed a novel combination of the VIDPF and authenticated secret sharing. Additionally, the online computation complexity of CAVERN is polynomial in the input size, where the prior maliciously secure $(2+1)$-PC protocols suffer from exponential complexity (cf. **TABLE 1**).

Implementing and Benchmarking. We present an implementation of CAVERN to show the practicality of our protocol and benchmark it against the state-of-the-art semi-honest protocol GROTTO [13] and the trusted-dealer-based malicious 2PC protocol SHARK [9]. As anticipated, the

TABLE 1: Comparing CAVERN to prior works.

Schemes	Model	Security	Core Primitives	Online Computation[a]	Online Communication[b]	Preprocessing Size[c]
GROTTO[13]	$(2+1)$-PC	Semi-honest	DPF	$O(\ell n)$	$O(dn)$	$O(n(n+d+\lambda))$
PIKA[15]	$(2+1)$-PC	Malicious	VDPF	$O(2^n)$	$O(n)$	$O(2^n)$
SHARK[9]	2PC with trusted dealer	Malicious	DCF&DPF	$O(\ell n(n+s)/\lambda)$	$O(d(n+s))$	$O(n(n+d+\lambda))$
CAVERN	$(2+1)$-PC	Malicious	VIDPF	$O(\ell n(n+s)/\lambda)$	$O(d(n+s))$	$O(n(n+d+\lambda))$

[a] λ - the computational security parameter, n - the input length, s - the SPD$\mathbb{Z}_{2^n}$ slack, ℓ - the number of intervals of the spline.
[b] d - the degree of polynomials of the spline.
[c] Preprocessing size represents the size of the preprocessing data required for each online evaluation of a spline.

results indicate that CAVERN only imposes a constant factor overhead on the top of GROTTO and SHARK due to the additional cryptographic machinery required to ensure stronger security guarantees. In real-world deployments, these trade-offs are entirely acceptable. Specifically, during the online phase, CAVERN incurs computational and communication overheads amounting to less than three times those incurred by GROTTO and SHARK.

1.2. Technique Overview

To obtain the aforementioned maliciously secure protocol, we build on the ideas of the semi-honest protocol GROTTO [13], and extend it to the maliciously secure setting based on the incremental verifiable distributed point function (VIDPF) as presented in Section 3, which is a combination of the verifiable distributed point function (VDPF) [17] and the incremental distributed point function (IDPF) [18], and the SPD$\mathbb{Z}_{2^n}$ authenticated secret sharing [19].

Spline. An (ℓ, d)-spline splits the input domain into ℓ intervals and approximates the target function with a polynomial of degree d on each interval. Concretely, for degree-d polynomials $P = (p_1, \ldots, p_\ell) \in (\mathbb{Z}_{2^n}[X])^\ell$, a sorted list of endpoints $E = (e_1, \ldots, e_{\ell+1}) \in \mathbb{Z}_{2^n}^{\ell+1}$ with $e_1 = 0, e_{\ell+1} = 2^n$ and an input $x \in \mathbb{Z}_{2^n}$, define

$$\mathsf{spline}_{n,\ell,d,P,E}(x) = \sum_{i\in[\ell]} \mathcal{I}_{x\in[e_i,e_{i+1})} \cdot p_i(x),$$

where $\mathcal{I}_{\mathsf{pred}}$ is an indicator that outputs 1 if the predicate pred is true and 0 otherwise. Commonly used functions such as Rectified Linear Unit (ReLU) and Absolute value are special cases of splines. Moreover, splines have been used to approximate transcendental functions such as natural logarithm, sigmoid and square root [20], [3].

Evaluating Splines via DPF. In this paper, the evaluation of splines draws upon GROTTO [13], which employs the Distribution Point Function (DPF) [21] for their computation. Specifically, GROTTO designs an efficient algorithm called *prefix-parity algorithm* based on the tree structure of DPF keys to compute the indicators $\{\mathcal{I}_{x\in[e_i,e_{i+1})}\}_{i\in[\ell]}$ without any online communication. However, GROTTO's prefix-parity algorithm is limited to DPFs with one bit output, necessitating binary-to-arithmetic conversions to lift the output to the elements in $\mathbb{Z}_{2^n}$, and it falls short of enabling extensions to the maliciously secure setting. To address this, we employ IDPF, a superset of DPF, which additionally supports outputs at each level of the DPF tree structure, with output group encompassing any Abelian group, such as $\mathbb{Z}_{2^n}$. Based on the idea of prefix-parity algorithm in GROTTO, we design a new method to compute the indicators $\{\mathcal{I}_{x\in[e_i,e_{i+1})}\}_{i\in[\ell]}$ with IDPF keys, which allows us to directly compute the additive shares of the indicators $\{\mathcal{I}_{x\in[e_i,e_{i+1})}\}_{i\in[\ell]}$ in the arithmetic domain, and can be extended to the maliciously secure setting. We refer to Section 4.2 for more details.

Defending Against Malicious Behaviors. To safeguard our spline evaluation method against malicious adversary, we incorporate specific defenses targeting both the dealer and online parties. Recap that in our setting, only one of the three parties can be malicious, either the dealer or one of the two online parties. *To defend against a malicious dealer* $\mathcal{D}$, we employ VIDPF, enabling the online parties to cryptographically verify the well-formedness of the generated keys. Additionally, to ensure the dealer $\mathcal{D}$ generates desired VIDPF keys that determined by both the online parties, we require the dealer $\mathcal{D}$ to additionally generate a pair of VIDPF keys as tags. The online parties can then perform a consistency check to verify that the dealer-generated VIDPF keys are consistent with the desired function. *To defend against a malicious online party*, we utilize the authenticated secret sharing [19] to ensure that the online parties can check the integrity of any shared values among the two parties. In addition, a batch check protocol, adapted from [19], is employed to ensure the integrity of all the opened values.

1.3. Related Work

CAVERN builds upon and distinguishes itself from several key lines of research in secure multi-party computation for non-linear functions, particularly splines. We categorize these into semi-honest and maliciously secure protocols. **TABLE 1** summarizes the key characteristics of CAVERN in comparison to the most relevant existing literature.

Semi-Honest Schemes for Spline. Most of the prior works for evaluating splines in the semi-honest $(2+1)$ party setting employ one of two competing strategies: the Distributed Point Function (DPF) [16] based strategy and the Distributed Comparison Function (DCF) [8], [21] based strategy. The DPF based strategy, introduced by Vadapalli et al. [14] for their PIRSONA scheme, uses DPFs with a full-domain

evaluation procedure to evaluate splines. However, their approach incurs exponential computational cost, restricting its practical use to small domains. The DCF based strategy, introduced by Boyle et al. [8], uses DCFs to avoid the need for costly full-domain evaluations. The follow-up schemes such as [7], [10], [11], [12] further optimize the DCF-based approach, achieving significant efficiency improvements. However, substituting DCFs with DPFs markedly enhances computational scaling at the expense of larger "hidden constants" and substantially increased communication and storage overhead, which escalates rapidly as the approximation precision intensifies. The state-of-the-art semi-honest scheme GROTTO [13] presents a highly efficient scheme by the "prefix-parity algorithm" that leverages the tree structure of DPFs, allowing it to evaluate spline with a single DPF without suffering from the exponential computational cost full-domain evaluation incurs and the large communication and storage overhead incurred by DCFs. While GROTTO achieves high efficiency, it only considers the semi-honest setting, which is insufficient for many practical applications.

Maliciously Secure Schemes for Spline. The first maliciously secure scheme in this setting is PIKA [15], it generalized the semi-honest scheme PIRSONA [14] to arbitrary functions by lookup style secure computation via DPF and leveraging a novel extension of the Schwartz-Zippel lemma [22], [23] to rings to verify the well-formedness of DPF keys. However, PIKA still relies on the DPF-plus-full-domain-evaluation approach, which has low communication cost and concretely efficient running times for "short" inputs (say, 16-24 bits), but its exponential computation cost quickly grows untenable as inputs expand. The DCF based scheme SHARK [9] also gives a maliciously secure protocol for evaluating splines by utilizing SPD$\mathbb{Z}_{2^n}$ authenticated secret sharing within the Function Secret Sharing (FSS) based 2PC model [8]. Beyond inheriting the limitations inherent to the DCF based semi-honest scheme, SHARK necessitates a trusted dealer to provide input-independent preprocessing material, posing a significant constraint for practical deployment. Additionally, both PIKA and SHARK requires a large storage overhead, which is a significant bottleneck for practical applications.

Defending Against Malicious Dealer. One could alternatively use 2PC protocols to instantiate the trusted dealer of SHARK in the malicious setting. However, the state-of-the-art 2PC protocol [24] that can securely realize the dealer functionality incurs an offline computation complexity of $O(N)$, where $N = 2^n$ is the input domain size. This becomes impractical even for moderate values of n (e.g., $n = 32$). One could also alternatively use SNARKs to enforce honest dealer behavior of SHARK; but, disregarding technical issues that may arise by doing so, even the state-of-the-art general purpose SNARKs (see, e.g., [25], [26]) would impose additional $O((n(n+s)/\lambda)\log(n(n+s)/\lambda))$ computation complexity on the dealer. In comparison, compared to these two alternatives, CAVERN maintains an offline computation complexity of just $O(n(n+s)/\lambda)$, introducing only a constant factor overhead over the basic SHARK design. Although a generic compilation is an option for lifting SHARK to malicious security, this paper gives a custom solution that is significantly more efficient.

1.4. Organization

In Section 2 we introduce preliminaries, including notations, security model, and background concepts. In Section 3, we present the definition and our construction of verifiable incremental distributed point function (VIDPF), which is the key primitive of our protocol. In Section 4, we give our maliciously secure spline evaluation protocol CAVERN and prove its security. In Section 5, we present our implementation of CAVERN and benchmark it against the state-of-the-art semi-honest protocol GROTTO and the trusted-dealer-based 2PC maliciously secure protocol SHARK. Finally, we conclude our work in Section 6.

2. Preliminaries

2.1. Notation

Let $\mathbb{N}$ and $\mathbb{Z}$ be the set of *natural numbers* and the set of *integers*, respectively. We denote by PPT *probabilistic polynomial time*, and by $\mathsf{negl}(\lambda)$ a *negligible* function in λ. For any finite set S, we denote by $s \leftarrow S$ the process of sampling an element s uniformly at random from S. For any integer $n > 0$, we use $[n]$ to denote the set $\{1, \ldots, n\}$, and use $\mathbb{Z}_{2^n}$ to denote the ring of integers modulo 2^n. Depending on the context, we use $\{0,1\}^n$ and $\mathbb{Z}_{2^n}$ interchangeably and omit the conversion between them. We say that two values a and b are equal modulo n and denote $a \equiv_n b$, if $a \bmod n = b \bmod n$. We use $\|$ for bit-string concatenation. For any bit-string $x \in \{0,1\}^n$ and any $i \in [n]$, let x_i denote its i-th most significant bit, and $x_{[i]}$ denote the i-bit prefix of x, i.e., $x_{[i]} = x_1 \cdots x_i \in \{0,1\}^i$. For a single bit $x \in \{0,1\}$, we use $\bar{x}$ to denote the negation of x. Let $\mathcal{I}_{\mathsf{pred}}$ represents the indicator function that outputs 1 when the predicate pred is true and 0 otherwise. For any numbers $a, b \in \mathbb{Z}_{2^n}$ and $a < b$, we use $[a, b)$ to denote the interval $\{x \in \mathbb{Z}_{2^n} \mid a \le x < b\}$, and for any $x \in \mathbb{Z}_{2^n}$, we use $[a, b) \ggg x$ to denote cyclically shifting the interval $[a, b)$ to right by x, and use $[a, b) \lll x$ to denote cyclically shifting the interval $[a, b)$ to left by x. We use $\oplus$ to denote the bitwise XOR operation, and use $\wedge$ to denote the bitwise AND operation. We denote by $\{v\}^n$ the multiset consisting of the value v repeated n times.

2.2. Security Model

Boyle et al. [27] introduced a $(2+1)$ model for MPC, where an offline party $\mathcal{D}$ (called dealer) prepares the necessary randomness used by two online parties P_0 and P_1 and the security was defined by adapting the standard security definition based on the ideal/real model paradigm from [28], [29]. In this work, our protocols will be constructed in this $(2+1)$ model. For simplicity, we consider the case of evaluating a single function f over $\mathbb{Z}_{2^n}$ (i.e., a function with

all computations in $\mathbb{Z}_{2^n}$), but our definition can be naturally extended to multiple functions.

Definition 1 (MPC in the $(2+1)$ Model)**.** *Let $\mathcal{F}$ be the class of 2 party ideal functionality over $\mathbb{Z}_{2^n}$ represented by arithmetic functions f. Let κ be a security parameter. Let P_0 and P_1 be the two online parties and let $\mathcal{D}$ be an offline party (also called dealer). A protocol in the $(2+1)$ model for $\mathcal{F}$ is defined by a pair of protocols $(\Pi_{\mathsf{off}}, \Pi_{\mathsf{on}})$ such that:*

- Π_{off} *involves $\mathcal{D}$ and P_0, P_1. It starts with each party holding a public input 1^κ, and ends with each online party P_σ ($\sigma \in \{0,1\}$) outputting a local random string o_σ and the offline party $\mathcal{D}$ outputting nothing.*
- Π_{on} *involves P_0, P_1. It starts with each online party P_σ ($\sigma \in \{0,1\}$) holding a public input f and a local input (o_σ, x_σ), and ends with each P_σ holding an output y_σ.*

In the ideal-world *execution, three parties P_0, P_1 and $\mathcal{D}$ interact with an ideal functionality $\mathcal{F}$, and one of them may be corrupted by an* ideal-world adversary *(a.k.a.,* simulator*) $\mathcal{S}$. In the* real-world *execution, the three parties P_0, P_1 and $\mathcal{D}$ interacts with each other via executing a protocol Π, and one of them may be corrupted by a* real-world adversary *$\mathcal{A}$. We say that a protocol Π, obtained by first running Π_{off} and then Π_{on}, is said to be a* secure *protocol for $\mathcal{F}$ in the $(2+1)$ model if the real-world execution of Π is computationally indistinguishable from the ideal-world execution of $\mathcal{F}$.*

By default, we consider security against malicious adversaries, which means that the corrupted party may deviate arbitrarily from the protocol specifications (e.g., send incorrect values).

2.3. Commitment and Coin Tossing Functionalities

Our protocol relies on two standard two-party functionalities: the *commitment* functionality $\mathcal{F}_{\mathsf{com}}$ and the *coin-tossing* functionality $\mathcal{F}_{\mathsf{coin}}$, which can be securely realized using a random oracle [19], [30].

The commitment functionality $\mathcal{F}_{\mathsf{com}}$ allows a party to commit to a value and later open it. Informally, to commit to a value x, the party P_σ ($\sigma \in \{0,1\}$) chooses a random r and sends $c = H(x\|r)$ to $P_{1-\sigma}$, where H is a hash function, modelled as a random oracle. To open the commitment, the party P_σ sends (x, r) to $P_{1-\sigma}$, which verifies that $c = H(x\|r)$.

The coin tossing functionality $\mathcal{F}_{\mathsf{coin}}$ samples an element from a set S uniformly at random and hands it to both parties. This can be realized by having each party P_σ ($\sigma \in \{0,1\}$) first commit to a random seed s_σ, and then both parties open all commitments and use $s_0 \oplus s_1$ as a random seed to sample a random value $r \in S$.

2.4. Additive Secret Sharing

For any integer $n > 0$, we use $\langle x \rangle^{2^n} = (\langle x \rangle_0^{2^n}, \langle x \rangle_1^{2^n})$ to denote an additive sharing of $x \in \mathbb{Z}_{2^n}$, where $\langle x \rangle_\sigma^{2^n} \in \mathbb{Z}_{2^n}$ ($\sigma \in \{0,1\}$) and $\langle x \rangle_0^{2^n} + \langle x \rangle_1^{2^n} = x$. We abbreviate $\langle x \rangle^{2^n}$ as $\langle x \rangle$ when $n > 1$ is clear from the context. To open $\langle x \rangle$ such that both parties learn x in clear, each party P_σ ($\sigma \in \{0,1\}$) sends its own $\langle x \rangle_\sigma^{2^n}$ to $P_{1-\sigma}$. Upon receiving the value $\langle x \rangle_{1-\sigma}^{2^n}$, P_σ computes $x = \langle x \rangle_0^{2^n} + \langle x \rangle_1^{2^n} \bmod 2^n$.

2.5. Authenticated Secret Sharing

We consider the SPD$\mathbb{Z}_{2^n}$ authenticated secret sharing [19] in two-party setting, which enables two parties to compute over the ring $\mathbb{Z}_{2^n}$ for any integer $n > 0$. The main idea behind this authenticated secret sharing scheme is that, to perform a secure computation on additive shares modulo 2^n with malicious security, the parties can run a computation over a larger ring modulo 2^{n+s}, where the statistical security parameter is denoted as $\kappa = s - \log s$, but correctness is only guaranteed modulo 2^n. The reason for this is that in a ring with many zero divisors, traditional information-theoretic MACs cannot protect the integrity of a ring element $x' \in \mathbb{Z}_{2^{n+s}}$, but can offer integrity on its lower-order n bits, namely $x = x' \bmod 2^n$.

We use $[\![x]\!]^{2^n} = ([\![x]\!]_0^{2^n}, [\![x]\!]_1^{2^n})$ to denote an SPD$\mathbb{Z}_{2^n}$ authenticated secret sharing of $x \in \mathbb{Z}_{2^n}$ in which the party P_σ ($\sigma \in \{0,1\}$) holds $[\![x]\!]_\sigma^{2^n} = (x_\sigma, \mathsf{M}_\sigma[x]) \in \mathbb{Z}_{2^{n+s}}^2$ such that $x \equiv_{2^n} x_0 + x_1$ and $\mathsf{M}_0[x] + \mathsf{M}_1[x] \equiv_{2^{n+s}} (\Delta_0 + \Delta_1) \cdot (x_0 + x_1)$, where Δ_σ is a global MAC key that is selected uniformly at random from $\mathbb{Z}_{2^s}$ by each party P_σ ($\sigma \in \{0,1\}$). We abbreviate $[\![x]\!]^{2^n}$ as $[\![x]\!]$ when $n > 1$ is clear from the context.

Note that SPD$\mathbb{Z}_{2^n}$ authenticated secret sharing are *additively homomorphic*. For instance, given secret shared values $[\![x]\!]$, $[\![y]\!]$ and public constants $a, b \in \mathbb{Z}_{2^n}$, two parties can locally compute $[\![a \cdot x + b \cdot y]\!] = a \cdot [\![x]\!] + b \cdot [\![y]\!]$. Besides, for any public constant $c \in \mathbb{Z}_{2^n}$, both parties can locally compute $[\![c]\!]$ by setting $c_0 := c, c_1 := 0, \mathsf{M}_0[c] = c \cdot \Delta_0$ and $\mathsf{M}_1[c] = c \cdot \Delta_1$.

Protocol $\Pi_{\mathsf{BatchCheck}}$

Input: Two parties P_0 and P_1 hold ℓ SPD$\mathbb{Z}_{2^n}$ authenticated sharing $[\![x^{(1)}]\!], \ldots, [\![x^{(\ell)}]\!]$ along with their opened value $x^{(i)} \in \mathbb{Z}_{2^n}$ for each $i \in [\ell]$.

Batch Check: Two parties P_0 and P_1 do the following.

1) Two parties call $\mathcal{F}_{\mathsf{coin}}$ to sample a public random value $\gamma \in \mathbb{Z}_{2^n}$.
2) Two parties locally compute $[\![z]\!] := \sum_{i \in [\ell]} \gamma^i \cdot [\![x^{(i)}]\!]$ and $z' := \sum_{i \in [\ell]} \gamma^i \cdot x^{(i)} \in \mathbb{Z}_{2^n}$.
3) For each $\sigma \in \{0,1\}$, P_σ computes $V_\sigma := \mathsf{M}_\sigma[z] - z' \cdot \Delta_\sigma$ and calls $\mathcal{F}_{\mathsf{com}}$ to commit to V_σ.
4) Two parties open their commitments and verify that $V_0 + V_1 = 0$. If the check fails then parties abort.

Figure 1: Protocol for batch check.

To open $[\![x]\!]$ such that both parties learn x in clear, each party P_σ ($\sigma \in \{0,1\}$) uses a random shared value $[\![r]\!]$ to mask the upper s bits of x, getting $[\![\tilde{x}]\!] := [\![x]\!] + 2^n \cdot [\![r]\!]$. Each party P_σ sends its own $\tilde{x}_\sigma$ (but not $\mathsf{M}_\sigma[\tilde{x}]$) to $P_{1-\sigma}$. Upon receiving the value $\tilde{x}_{1-\sigma}$, P_σ computes $x = \tilde{x}_\sigma + \tilde{x}_{1-\sigma} \bmod 2^n$. To ensure correctness, the MAC

$\mathsf{M}[\tilde{x}] = \mathsf{M}_0[\tilde{x}] + \mathsf{M}_1[\tilde{x}] \bmod 2^n$ should be checked, but can be deferred and batched with other opened values. The protocol $\Pi_{\mathsf{BatchCheck}}$ (see **Figure 1**) of [19] takes as input a set of opened values and verifies the MACs of all the opened values, with failure probability negligible in κ.

$\mathcal{F}_{\mathsf{edaBits}}$

On input $(\mathsf{edaBits}, \ell)$ from P_0 and P_1, samples uniformly at random values $(r_1, \ldots, r_\ell) \in \mathbb{Z}_2^\ell$, compute $r = \sum_{i=1}^{\ell} 2^{i-1} r_i$, generates $\{[\![r_i]\!]^2\}_{i\in[\ell]}$ and $[\![r]\!]^{2^\ell}$, then distributes the sharings to all parties.

Figure 2: Functionality $\mathcal{F}_{\mathsf{edaBits}}$.

Remark 1. *All the operations above work over a Boolean ring $\mathbb{Z}_{2^1}$ (i.e., $n = 1$) as well. In this case, the additions (or subtractions) and multiplications are replaced by XORs and ANDs, respectively.*

2.6. Generating edaBits

Recently, Escudero et al. [31] introduced a new primitive called edaBits (extended doubly authenticated bits) that generates shares of a random value and its bit decomposition. More formally, an edaBits is a set of shares $([\![r]\!]^{2^\ell}, \{[\![r_i]\!]^2\}_{i\in[\ell]})$ such that $r = \sum_{i=1}^{\ell} 2^{i-1} \cdot r_i$, where r_i is random bit. We use $\mathcal{F}_{\mathsf{edaBits}}$ to denote the ideal functionality that generates edaBits, which is defined in **Figure 2**. A maliciously secure realization of this functionality can be found in [31].

2.7. XOR-Collision Resistant Hash Functions

Following the definition in [17], we say a function family $\mathcal{H}$ is *XOR-Collision resistant* if given a randomly chosen $H \leftarrow \mathcal{H}$ no PPT adversary can find four values $x_0, x_1, x_2, x_3 \in \{0,1\}^\lambda$ such that $(x_0, x_1) \neq (x_2, x_3)$, $(x_0, x_1) \neq (x_3, x_2)$ and $H(x_0) \oplus H(x_1) = H(x_2) \oplus H(x_3) \neq 0$ with non-negligible probability in λ.

2.8. Distributed Point Function

A point function $f_{\alpha,\beta}$ for $\alpha \in \{0,1\}^n$ and $\beta \in \mathbb{G}$, is a function such that $f_{\alpha,\beta}(\alpha) = \beta$ and $f_{\alpha,\beta}(x) = 0$ for all $x \in \{0,1\}^n \setminus \{\alpha\}$. A two-party Distributed Point Function (DPF) [16] allows a dealer to additively secret share a point function $f_{\alpha,\beta}$ among two evaluators such that each evaluator can use its share to locally compute an additive share of $f_{\alpha,\beta}(x)$ for any $x \in \{0,1\}^n$.

Definition 2 (Distributed Point Function)**.** *A (2-party) DPF scheme for a family $\mathcal{F} = \{f_{\alpha,\beta}\}_{\alpha\in\{0,1\}^n,\beta\in\mathbb{G}}$ of point functions is a set of PPT algorithms* $(\mathsf{Gen}, \mathsf{Eval})$ *such that:*

- $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, f)$ *is a key generation algorithm that, given a security parameter λ and a point function $f \in \mathcal{F}$, outputs a pair of DPF keys.*
- $y_\sigma \leftarrow \mathsf{Eval}(\sigma, k_\sigma, x)$ *is an evaluation algorithm that, given an index $\sigma \in \{0,1\}$, a DPF key k_σ, and a point $x \in \{0,1\}^n$, outputs a share y_σ.*

A secure DPF must satisfy two properties as follows:

- **Correctness.** *For any point function $f \in \mathcal{F}$ and any $x \in \{0,1\}^n$, it holds that*

$$\Pr\left[\begin{array}{l}(k_0,k_1) \leftarrow \mathsf{Gen}(1^\lambda, f): \\ \mathsf{Eval}(0,k_0,x) + \mathsf{Eval}(1,k_1,x) = f(x)\end{array}\right] = 1.$$

- **Privacy.** *For any function $f \in \mathcal{F}$, any party index $\sigma \in \{0,1\}$, and any PPT adversary $\mathcal{A}$, there exists a PPT simulator* Sim *such that*

$$\left|\Pr\left[\mathcal{A}(k_\sigma) = 1 \mid (k_0,k_1) \leftarrow \mathsf{Gen}(1^\lambda, f)\right] - \Pr\left[\mathcal{A}(k_\sigma) = 1 \mid k_\sigma \leftarrow \mathsf{Sim}(\sigma)\right]\right| \leq \mathsf{negl}(\lambda).$$

Incremental DPF. Incremental DPF (IDPF) [18] is a generalization of DPF that is designed for the function family of all-prefix point functions. An all-prefix point function represented by a tuple $(\alpha, (\mathbb{G}_1, \beta_1), \ldots, (\mathbb{G}_n, \beta_n))$ ($(\alpha, \vec{\beta})$ for short), where $\alpha \in \{0,1\}^n$, and for each $i \in [n]$, $\mathbb{G}_i$ is the description of an Abelian group and $\beta_i \in \mathbb{G}_i$, is defined as $f_{\alpha,\vec{\beta}} : \cup_{i\in[n]}\{0,1\}^i \to \cup_{i\in[n]}\mathbb{G}_i$, where for each $i \in [n]$ and each $x \in \{0,1\}^i$

$$f_{\alpha,\vec{\beta}}(x) = \begin{cases} \beta_i & \text{if } x = \alpha_{[i]}, \\ 0 & \text{otherwise.} \end{cases}$$

Compared with DPF, the key generation algorithm Gen of IDPF takes an all-prefix point function $f_{\alpha,\vec{\beta}}$ as input and outputs two shares (k_0, k_1) which define $f_{\alpha,\vec{\beta}}$. The evaluation algorithm Eval of IDPF evaluates on $x \in \cup_{i\in[n]}\{0,1\}^i$ and outputs a share in $\mathbb{G}_{|x|}$ such that:

$$\mathsf{Eval}(0,k_0,x) + \mathsf{Eval}(1,k_1,x) = \begin{cases} \beta_i & \text{if } x = \alpha_{[|x|]}, \\ 0 & \text{otherwise.} \end{cases}$$

Informally, the IDPF can be viewed as a combination of n DPFs, where the i-th DPF corresponds to a point function $f_{\alpha_{[i]},\beta_i}$ for $i \in [n]$.

Verifiable DPF. Recently, a new notion, called *verifiable* DPF (VDPF), has been proposed to ensure the well-formedness of the shared point function against malicious dealer [17], [18]. Beyond the conventional security guarantees of DPF, VDPF further enables the evaluators to check whether the shares provided by the dealer define a well-formed point function. A VDPF consists of three PPT algorithms $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Ver})$. Compared with DPF, the evaluation algorithm Eval of VDPF focuses on processing a set of inputs $X \subseteq \{0,1\}^n$, in addition to producing a share of the function output $f(x)$ for each input $x \in X$, it also generates an audit token. The verification algorithm Ver takes the audit tokens as input and ensures rejection if for two or more inputs $x \in X$ the function output shares reconstruct a nonzero value.

• **Parameters:** Domain size $N = 2^n$ for $n \in \mathbb{N}$, Abelian groups $\{\mathbb{G}_i\}_{i\in[n]}$, pseudorandom generators $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$, $G_i : \{0,1\}^\lambda \to \{0,1\}^\lambda \times \mathbb{G}_i$ for $i \in [n]$, collision-resistant and XOR-collision-resistant hash function $\mathsf{H}_i : \{0,1\}^{\lambda+1+i} \to \{0,1\}^{4\lambda}$ for $i \in [n]$, collision-resistant hash function $\mathsf{H}' : \{0,1\}^{4\lambda} \to \{0,1\}^{2\lambda}$.

• $\mathsf{Gen}(1^\lambda, \alpha, \{\beta_i \in \mathbb{G}_i\}_{i\in[n]})$:
1) Let $\alpha_1, \ldots, \alpha_n$ be the bit decomposition of α
2) $s_0^{(0)}, s_1^{(0)} \leftarrow \{0,1\}^\lambda$
3) $t_0^{(0)} := 0, t_1^{(0)} := 1$
4) **for** $i \in [n]$:
5) $\quad s_\sigma^L \| s_\sigma^R \| t_\sigma^L \| t_\sigma^R \leftarrow G(s_\sigma^{(i-1)})$ for $\sigma \in \{0,1\}$
6) $\quad$ **if** $\alpha_i = 0$: $\mathsf{keep} := L, \mathsf{lose} := R$
7) $\quad$ **else** $\mathsf{keep} := R, \mathsf{lose} := L$
8) $\quad t_{\mathsf{cw}}^L := t_0^L \oplus t_1^L \oplus \bar{\alpha}_i$, $t_{\mathsf{cw}}^R := t_0^R \oplus t_1^R \oplus \alpha_i$
9) $\quad s_{\mathsf{cw}} := s_0^{\mathsf{lose}} \oplus s_1^{\mathsf{lose}}$, $\mathsf{cw}_i := (s_{\mathsf{cw}}, t_{\mathsf{cw}}^L, t_{\mathsf{cw}}^R)$
10) $\quad \tilde{s}_\sigma^{(i)} := s_\sigma^{\mathsf{keep}} \oplus t_\sigma^{(i-1)} \cdot s_{\mathsf{cw}}$ for $\sigma \in \{0,1\}$
11) $\quad t_\sigma^{(i)} := t_\sigma^{\mathsf{keep}} \oplus t_\sigma^{(i-1)} \cdot t_{\mathsf{cw}}^{\mathsf{keep}}$ for $\sigma \in \{0,1\}$
12) $\quad \mathsf{cs}_i := \mathsf{H}_i(\tilde{s}_0^{(i)} \| t_0^{(i)} \| \alpha_{[i]}) \oplus \mathsf{H}_i(\tilde{s}_1^{(i)} \| t_1^{(i)} \| \alpha_{[i]})$
13) $\quad s_\sigma^{(i)} \| w_\sigma^{(i)} \leftarrow G_i(\tilde{s}_\sigma^{(i)})$ for $\sigma \in \{0,1\}$
14) $\quad \mathsf{ocw}_i := (-1)^{t_1^{(i)}} \cdot (w_1^{(i)} - w_0^{(i)} + \beta_i)$
15) $k_0 = (s_0^{(0)}, \{\mathsf{cw}_i, \mathsf{ocw}_i, \mathsf{cs}_i\}_{i\in[n]})$
16) $k_1 = (s_1^{(0)}, \{\mathsf{cw}_i, \mathsf{ocw}_i, \mathsf{cs}_i\}_{i\in[n]})$
17) **return** (k_0, k_1)

• $\mathsf{Eval}(\sigma, k_\sigma, X \subseteq \bigcup_{i=1}^n \{0,1\}^i)$:
1) Parse $k_\sigma = (s_\sigma^{(0)}, \{\mathsf{cw}_i, \mathsf{ocw}_i, \mathsf{cs}_i\}_{i\in[n]})$
2) $t_\sigma^{(0)} = \sigma, \mu_\sigma := 0^{4\lambda}$
3) **for** $i \in [n]$: $\mu_\sigma^{(i)} := \mathsf{cs}_i$
4) **for** $x \in X$:
5) $\quad$ Let $x_1, \ldots, x_\ell$ be the bit decomposition of x, $\ell = |x|$
6) $\quad$ **for** $i \in [\ell]$:
7) $\quad\quad s_\sigma^L \| s_\sigma^R \| t_\sigma^L \| t_\sigma^R \leftarrow G(s_\sigma^{(i-1)})$
8) $\quad\quad$ Parse $\mathsf{cw}_i = (s_{\mathsf{cw}}, t_{\mathsf{cw}}^L, t_{\mathsf{cw}}^R)$
9) $\quad\quad$ **if** $x_i = 0$: $\mathsf{keep} := L$ **else**: $\mathsf{keep} := R$
10) $\quad\quad \tilde{s}_\sigma^{(i)} \| t_\sigma^{(i)} := (s_\sigma^{\mathsf{keep}} \| t_\sigma^{\mathsf{keep}}) \oplus t_\sigma^{(i-1)} \cdot (s_{\mathsf{cw}} \| t_{\mathsf{cw}}^{\mathsf{keep}})$
11) $\quad\quad s_\sigma^{(i)} \| w_\sigma^{(i)} \leftarrow G_i(\tilde{s}_\sigma^{(i)})$
12) $\quad \mu_\sigma^{(\ell)} := \mu_\sigma^{(\ell)} \oplus \mathsf{H}'(\mu_\sigma^{(\ell)} \oplus \mathsf{H}_\ell(\tilde{s}_\sigma^{(\ell)} \| t_\sigma^{(\ell)} \| x) \oplus t_\sigma^{(\ell)} \cdot \mathsf{cs}_\ell)$
13) $\quad y_\sigma^{(x)} := (-1)^\sigma \cdot (w_\sigma^{(\ell)} + t_\sigma^{(\ell)} \cdot \mathsf{ocw}_\ell)$
14) $\quad$ **for** $i \in [n]$: $\mu_\sigma := \mu_\sigma \oplus \mathsf{H}'(\mu_\sigma \oplus \mu_\sigma^{(i)})$
15) **return** $(\{y_\sigma^{(x)}\}_{x\in X}, \mu_\sigma)$

• $\mathsf{Ver}(\mu_0, \mu_1)$:
1) **return** $\mu_0 = \mu_1$

Figure 3: The construction of VIDPF.

3. Verifiable Incremental DPF

3.1. Definition

In this section, we present the definition and construction of verifiable incremental DPF (VIDPF). VIDPF extends incremental DPF to ensure well-formedness against malicious dealers, defined formally below:

Definition 3 (Verifiable Incremental Distributed Point Function)**.** *A (2-party) VIDPF scheme for a family $\mathcal{F}$ of all-prefix point functions $f_{\alpha,\vec{\beta}} : \cup_{i\in[n]}\{0,1\}^i \to \cup_{i\in[n]}\mathbb{G}_i$, where $\mathbb{G}_i$ is an Abelian group for each $i \in [n]$, is a set of PPT algorithms* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Ver})$ *such that:*

- $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, f)$ *is the key generation algorithm that, given a security parameter λ and an all-prefix point function $f \in \mathcal{F}$, outputs a pair of VIDPF keys (k_0, k_1).*
- $(\{y_\sigma^{(x)}\}_{x\in X}, \mu_\sigma) \leftarrow \mathsf{Eval}(\sigma, k_\sigma, X)$ *is the evaluation algorithm that, given an index $\sigma \in \{0,1\}$, a VIDPF key k_σ, and a set of inputs $X \subseteq \cup_{i\in[n]}\{0,1\}^i$, outputs a tuple of values. The first element of the tuple is a set $\{y_\sigma^{(x)}\}_{x\in X}$ of VIDPF outputs, which are additive shares of $\{f(x)\}_{x\in X}$. The second element of the tuple is an audit token μ_σ that is used to verify the well-formedness of the shared function.*
- $1/0 \leftarrow \mathsf{Ver}(\mu_0, \mu_1)$ *is the verification algorithm that, given the two audit tokens μ_0 and μ_1, outputs 1 for accept and 0 for reject.*

A secure VIDPF must satisfy the following three properties:

- **Completeness.** *For any all-prefix point function $f \in \mathcal{F}$ and any $X \subseteq \cup_{i\in[n]}\{0,1\}^i$, it holds that*

$$\Pr\left[\begin{array}{c}(k_0,k_1) \leftarrow \mathsf{Gen}(1^\lambda, f);\\ (\{y_0^{(x)}\}_{x\in X}, \mu_0) \leftarrow \mathsf{Eval}(0, k_0, X);\\ (\{y_1^{(x)}\}_{x\in X}, \mu_1) \leftarrow \mathsf{Eval}(1, k_1, X) :\\ (\forall x \in X, y_0^{(x)} + y_1^{(x)} = f(x)) \wedge\\ (\mathsf{Ver}(\mu_0, \mu_1) = 1)\end{array}\right] = 1.$$

- **Privacy.** *For an all-prefix point function $f \in \mathcal{F}$ and a set of inputs $X \subseteq \cup_{i\in[n]}\{0,1\}^i$, define the view of evaluator P_σ as $\mathsf{View}_{\mathsf{VIDPF}}(\sigma, f, X) := \{(k_\sigma, \mu_{1-\sigma})\}_\lambda$, where the VIDPF keys (k_0, k_1) are generated by $\mathsf{Gen}(1^\lambda, f)$, and the audit token $\mu_{1-\sigma}$ is generated by $\mathsf{Eval}(1-\sigma, k_{1-\sigma}, X)$. There exists a PPT simulator Sim such that for all $f \in \mathcal{F}$ and $X \subseteq \cup_{i\in[n]}\{0,1\}^i$, the following two distributions are computationally indistinguishable:*

$$\mathsf{View}_{\mathsf{VIDPF}}(\sigma, f, X) \approx_c \mathsf{Sim}(1^\lambda, \sigma, X).$$

- **Soundness.** *For any $X \subseteq \cup_{i\in[n]}\{0,1\}^i$, any PPT adversary $\mathcal{A}$, it holds that*

$$\Pr\left[\begin{array}{c}(k_0,k_1) \leftarrow \mathcal{A}(1^\lambda);\\ (\{y_0^{(x)}\}_{x\in X}, \mu_0) \leftarrow \mathsf{Eval}(0, k_0, X);\\ (\{y_1^{(x)}\}_{x\in X}, \mu_1) \leftarrow \mathsf{Eval}(1, k_1, X) :\\ (\mathsf{Ver}(\mu_0, \mu_1) = 1) \wedge (\exists i \in [n],\\ \left|\{x \in X_i \mid y_0^{(x)} + y_1^{(x)} \neq 0\}\right| > 1)\end{array}\right] < \mathsf{negl}(\lambda),$$

where $X_i := \{x \in X \mid |x| = i\}$ for $i \in [n]$.

Informally, the completeness property ensures that for any all-prefix point function $f_{\alpha,\vec{\beta}}$, the VIDPF keys, generated by the key generation algorithm, correctly define $f_{\alpha,\vec{\beta}}$ and the evaluation algorithm will output the additive shares of the function outputs for any inputs. And the privacy property guarantees that an adversary that learns k_σ ($\sigma \in \{0,1\}$) and the audit token $\mu_{1-\sigma}$ learns no information about the all-prefix point function $f_{\alpha,\vec{\beta}}$. The soundness property ensures that the not well-formed VIDPF keys will be rejected by the verification algorithm with overwhelming probability. In particular, for any $i \in [n]$, if there are more than one input $x \in X_i$ such that $y_0^{(x)} + y_1^{(x)} \neq 0$, then the verification algorithm will reject with overwhelming probability.

3.2. Construction

Recap the DPF representation in [21], for the point function $f_{\alpha,\beta}$, each of the two DPF keys defines a GGM-style tree [32] with 2^n leaves, and every node in the tree is labeled by a pseudorandom value, where (1) in both trees the labels of each node on the path from root to the α-th leaf are different and the XOR of the two labels has LSB one, (2) the labels corresponding to other nodes are identical. Therefore, we can instruct the i-th output by making the labels of i-th node on the path from root to the α-th leaf reconstruct β_i, while correcting others to 0.

More specifically, our VIDPF construction is depicted in **Figure 3**. In particular, the label of each node includes a pseudo-random seed $\tilde{s}_\sigma$ and a control bit t_σ. They generate the labels of child nodes with the help of the corresponding correction word cw. As observed in [18], using the seed $\tilde{s}_\sigma$ directly in the generation of both children and layer output would compromise its pseudo-randomness, which is required for privacy. Therefore, at each layer $i \in [n]$, an extra PRG $G_i : \{0,1\}^\lambda \to \{0,1\}^\lambda \times \mathbb{G}_i$ is employed to expand $\tilde{s}_\sigma$ to a new pseudo-random seed s_σ for children and an element w_σ for the layer output. Here, we give the pseudocode of G_i in **Figure 4**. For more information, we refer readers to [18].

$G_i(s)$:
1) Let $u := |\mathbb{G}_i|$
2) **if** $u = 2^m$ for an integer m:
3) Let $G' : \{0,1\}^\lambda \to \{0,1\}^{\lambda+m}$ be a PRG
4) **else**:
5) $\ell := \lceil \log_2(u) \rceil + \lambda$
6) Let $G' : \{0,1\}^\lambda \to \{0,1\}^{\lambda+\ell}$ be a PRG
7) $\tilde{s} \| w \leftarrow G'(s)$
8) Let $g \in \mathbb{G}_i$ be the group element represented by $w \bmod u$
9) **return** $(\tilde{s}, g)$

Figure 4: Pseudocode of the PRG $G_i : \{0,1\}^\lambda \to \{0,1\}^\lambda \times \mathbb{G}_i$.

To defend against a malicious dealer, we follow the verification method of the standard VDPF scheme in [17] and extend it to the incremental DPF setting. Specifically, we introduce a correction seed cs_i for each layer $i \in [n]$, which enable evaluators to verify that two trees defined by a pair of VIDPF keys only differ at one node per level. During the key generation, the correct seed cs_i is used to correct the hash values of the different node labels in the two trees, ensuring they are identical. As the correction seed can only correct at most one difference (guaranteed by the XOR-collision-resistance of the hash function), the evaluators can verify whether the two trees are well-formed by checking that all the corrected hash values of the node labels in the two trees are equal.

3.3. Security analysis

Theorem 1. *Let $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$, $G_i : \{0,1\}^\lambda \to \{0,1\}^\lambda \times \mathbb{G}_i$ for $i \in [n]$ be pseudorandom generators. Let $\mathsf{H}_i : \{0,1\}^{\lambda+1+i} \to \{0,1\}^{4\lambda}$ be a collision resistant for $i \in [n]$ and XOR-collision resistant hash function. Let $\mathsf{H}' : \{0,1\}^{4\lambda} \to \{0,1\}^{2\lambda}$ be a collision resistant hash function. The VIDPF construction in* **Figure 3** *is a secure VIDPF as described in* **Definition 3**.

Proof. We prove that the VIDPF construction in **Figure 3** satisfies the properties in **Definition 3**.

Completeness. If we ignore the correction seeds that are used to verify the well-formedness of the shared function, our VIDPF construction is the same as the incremental DPF construction of Boneh et al. [18]. As the computation of the correction seeds are independent of the computation of the VIDPF outputs, we can follow the same proof in [18] to show that the correctness of our VIDPF construction holds, which is omitted here.

Privacy. We construct an efficient simulator Sim for the view of any evaluator P_σ. On input $(1^\lambda, \sigma, X)$, the simulator Sim proceeds as follows:

1) $s_\sigma^{(0)} \leftarrow \{0,1\}^\lambda$
2) **for** $i \in [n]$:
3) $\mathsf{cw}_i \leftarrow \{0,1\}^{\lambda+2}$, $\mathsf{cs}_i \leftarrow \{0,1\}^{4\lambda}$, $\mathsf{ocw}_i \leftarrow \mathbb{G}_i$
4) $k^* = (s_\sigma^{(0)}, \{\mathsf{cw}_i, \mathsf{ocw}_i, \mathsf{cs}_i\}_{i\in[n]})$
5) $(\{y_*^{(x)}\}_{x\in X}, \mu^*) \leftarrow \mathsf{Eval}(\sigma, k^*, X)$
6) Output (k^*, μ^*)

In the real view of P_σ, the VIDPF key k_σ consists of (1) random seed $s_\sigma^{(0)}$ which are distributed identically to the output of Sim, (2) the correction words $\{\mathsf{cw}_i\}_{i\in[n]}$, where each one is XOR'd with the output of the PRG G, (3) the correction seed $\{\mathsf{cs}_i\}_{i\in[n]}$, where each one is XOR'd with the output of the XOR-collision-resistant hash function H_i, and (4) the output correction word $\{\mathsf{ocw}_i\}_{i\in[n]}$, where each one is added with the pseudorandom output of G_i. Since P_σ would never know the initial random seed $s_{1-\sigma}^{(0)}$ of the other evaluator, (2), (3) and (4) are computational indistinguishable from random elements. In addition, the audit token $\mu_{1-\sigma}$ is equal to μ_σ, and the evaluator holding k_σ can locally compute the value of μ_σ. Therefore, we conclude that the output of the simulator Sim is computationally indistinguishable from the real view of P_σ.

Soundness. Given the input set $X \subseteq \cup_{i\in[n]}\{0,1\}^i$ and the VIDPF key k_σ, the evaluator P_σ has $(\{y_\sigma^{(x)}\}_{x\in X}, \mu_\sigma) \leftarrow \mathsf{Eval}(\sigma, k_\sigma, X)$. In addition, for each layer $i \in [n]$, each evaluator has a set of seeds $\{\tilde{s}_\sigma^{(i,v)}\}_{v\in X_i}$ and control bits $\{t_\sigma^{(i,v)}\}_{v\in X_i}$, where $X_i := \{x \in X \mid |x| = i\}$. With the same correction seeds cs_i, we denote $\tilde{\mu}_\sigma^{(i,v)} := \mathsf{H}_i(\tilde{s}_\sigma^{(i,v)}\|t_\sigma^{(i,v)}\|v)$ and $\mu_\sigma^{(i,v)} := \tilde{\mu}_\sigma^{(i,v)} \oplus t_\sigma^{(i,v)}$ for each $i \in [n]$, $v \in X_i$, and $\sigma \in \{0,1\}$. We prove that no PPT adversary $\mathcal{A}$ can break the soundness of our VIDPF construction. Suppose that there exist a PPT adversary $\mathcal{A}$ that can construct a pair of VIDPF keys (k_0, k_1) such that

$$\Pr\left[\begin{array}{c}(\mathsf{Ver}(\mu_0,\mu_1)=1) \wedge (\exists i \in [n],\\ \left|\{x \in X_i \mid y_0^{(x)} + y_1^{(x)} \neq 0\}\right| > 1)\end{array}\right]$$

is non-negligible. Then the adversary $\mathcal{A}$ can be used to construct a collision for the XOR-collision-resistant hash functions $\{\mathsf{H}_i\}_{i\in[n]}$. W.l.o.g., we assume that this occurs at the layer $j \in [n]$, which implies that there exist two distinct values $u, v \in X_j$ s.t.

$$y_0^{(j,u)} + y_1^{(j,u)} \neq 0 \quad \text{and} \quad y_0^{(j,v)} + y_1^{(j,v)} \neq 0.$$

Since $\mathsf{Ver}(\mu_0,\mu_1) = 1$ we have $\forall x \in X_j$, $\mu_0^{(j,x)} = \mu_1^{(j,x)}$ due to the collision resistant of H'. This implies that there exists a correction seed cs_j such that

$$\mathsf{cs}_j = \tilde{\mu}_0^{(j,u)} \oplus \tilde{\mu}_1^{(j,u)} \quad \text{and} \quad \mathsf{cs}_j = \tilde{\mu}_0^{(j,v)} \oplus \tilde{\mu}_1^{(j,v)},$$

which breaks the XOR-collision-resistant of H_j. □

4. Maliciously Secure $(2+1)$-PC

In this section, we present our maliciously secure MPC protocol that securely realize the functionality $\mathcal{F}_{\mathsf{spline}}$ (see **Figure 5**) for any spline function $\mathsf{spline}_{n,\ell,d,P,E}$ over the ring $\mathbb{Z}_{2^n}$ in the $(2+1)$ party setting.

We first introduce the ideas for evaluating splines in the $(2+1)$ party setting, where each online party P_σ ($\sigma \in \{0,1\}$) takes a spline function $\mathsf{spline}_{n,\ell,d,P,E}$ and an additive share $\langle x\rangle_\sigma$ of the function's input x as input and finally learns an additive share $\langle s\rangle_\sigma$ of the function's output $s = \mathsf{spline}_{n,\ell,d,P,E}(x)$.

Recall that an (ℓ, d)-spline function $\mathsf{spline}_{n,\ell,d,P,E}$ splits the input domain into ℓ intervals and approximates the target function with a polynomial of degree d on each interval. Concretely, for degree-d polynomials $P = (p_1, \ldots, p_\ell) \in (\mathbb{Z}_{2^n}[X])^\ell$, a sorted list of endpoints $E = (e_1, \ldots, e_{\ell+1}) \in \mathbb{Z}_{2^n}^{\ell+1}$ with $e_1 = 0$ and $e_{\ell+1} = 2^n$, and any input $x \in \mathbb{Z}_{2^n}$,

$$\mathsf{spline}_{n,\ell,d,P,E}(x) = \sum_{i\in[\ell]} \mathcal{I}_{x\in[e_i,e_{i+1})} \cdot p_i(x).$$

Our approach follows the idea of GROTTO [13], which makes use of DPF to evaluate splines in the $(2+1)$ party setting. Therefore, we begin by providing an overview of their protocol.

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

The functionality works with three parties: two online parties P_0 and P_1, and a dealer $\mathcal{D}$. The functionality is parameterized by a ring $\mathbb{Z}_{2^n}$ and a spline function $\mathsf{spline}_{n,\ell,d,P,E}$ over $\mathbb{Z}_{2^n}$.

1) $\mathcal{D}$ sends the functionality the message $(\mathsf{init}, \mathcal{D})$.
2) P_0 and P_1 send the functionality the message $(\mathsf{init}, P_\sigma)$ for $\sigma \in \{0,1\}$.
3) Upon receiving all the init messages, the functionality waits to receive a signal from the ideal adversary $\mathcal{S}$.
4) If the functionality receives the message (abort) from $\mathcal{S}$, the functionality sends (abort) to the honest parties.
5) If the functionality receives the message $(\mathsf{continue})$ from $\mathcal{S}$, the functionality sends the two online parties the message (ready) and the dealer its empty output message $(\mathsf{output}, \perp)$.
6) After receiving a message (ready) from the functionality, P_σ ($\sigma \in \{0,1\}$) sends the functionality a message $(\mathsf{input}, [\![x]\!]_\sigma)$, where $[\![x]\!]_\sigma$ is an authenticated share of the input $x \in \mathbb{Z}_{2^n}$.
7) After receiving the two input messages $(\mathsf{input}, [\![x]\!]_0)$ and $(\mathsf{input}, [\![x]\!]_1)$, the functionality reconstructs the input x and computes $s = \mathsf{spline}_{n,\ell,d,P,E}(x)$.
8) The functionality waits to receive a signal from the ideal adversary $\mathcal{S}$.
9) If the functionality receives the message (abort) from $\mathcal{S}$, it sends the two online parties the message (abort).
10) If the functionality receives the message (ok) from $\mathcal{S}$, it shares s into $[\![s]\!]_0$ and $[\![s]\!]_1$, and sends P_σ ($\sigma \in \{0,1\}$) the message $(\mathsf{output}, [\![s]\!]_\sigma)$.

Figure 5: Ideal Functionality $\mathcal{F}_{\mathsf{spline}}$ for spline evaluation.

4.1. The GROTTO Protocol

A key challenge in evaluating splines in the $(2+1)$ party setting is to compute the indicators $\{\mathcal{I}_{x\in[e_i,e_{i+1})}\}_{i\in[\ell]}$ for each interval $[e_i, e_{i+1})$ defined by the endpoints $E = (e_1, \ldots, e_{\ell+1})$. As observed by GROTTO, for each $i \in [\ell]$, the indicator $\mathcal{I}_{x\in[e_i,e_{i+1})}$ can be expressed as:

$$\mathcal{I}_{x\in[e_i,e_{i+1})} = \mathcal{I}_{x\in[0,e_i)} \oplus \mathcal{I}_{x\in[0,e_{i+1})}.$$

Thus, the problem can be reduced to compute the indicators $\{\mathcal{I}_{x\in[0,e_i)}\}_{i\in\{2,\ldots,\ell\}}$ (with the fact that $e_1 = 0$ and $e_{\ell+1} = 2^n$, it always holds that $\mathcal{I}_{x\in[0,e_1)} = 0$ and $\mathcal{I}_{x\in[0,e_{\ell+1})} = 1$).

GROTTO gives an efficient way to evaluate the indicators $\{\mathcal{I}_{x\in[0,e_i)}\}_{i\in\{2,\ldots,\ell\}}$ via DPF. More precisely, GROTTO designs a *prefix-parity* algorithm based on the tree structure of the DPF keys that define the point function $f_{x,1}$. This algorithm is run by each online party P_σ ($\sigma \in \{0,1\}$) and takes the inputs to the Eval algorithm of the DPF scheme, i.e., a DPF key k_σ and a public input $e \in \mathbb{Z}_{2^n}$, as input and outputs an additive share $\langle \mathcal{I}_{x\in[0,e)}\rangle_\sigma^2$ of $\mathcal{I}_{x\in[0,e)}$ instead of that of $f_{x,1}(e)$.

However, when evaluating on different inputs x, this approach requires different DPF keys, as these keys are

generated for the point function $f_{x,1}$, leading to an impractical overhead. To address this issue, GROTTO lets the online party P_σ randomly select $\langle\alpha\rangle_\sigma \in \mathbb{Z}_{2^n}$ for each $\sigma \in \{0,1\}$ and let the dealer $\mathcal{D}$ generate the DPF keys for the point function $f_{\alpha,1}$, where $\alpha = \langle\alpha\rangle_0 + \langle\alpha\rangle_1$. The two online parties then open the value $x - \alpha$ via $\langle\alpha\rangle$ and $\langle x\rangle$ and cyclically shift each endpoint in E to the left by $x - \alpha$ to obtain the shifted endpoints set E'. With the fact that $\mathcal{I}_{x\in[0,e_i)} = \mathcal{I}_{\alpha\in([0,e_i)\lll(x-\alpha))}$, the two parties can efficiently compute $\mathcal{I}_{x\in[0,e_i)}$ via evaluating the prefix parity algorithm on each endpoint in E' with the DPF keys for $f_{\alpha,1}$, which is independent of the input x.

4.1.1. Limitations of the GROTTO. In GROTTO, the indicators $\{\mathcal{I}_{x\in[e_i,e_{i+1})}\}_{i\in[\ell]}$ are shared between the two online parties in the binary domain, i.e., each party P_σ holds $\langle\mathcal{I}_{x\in[e_i,e_{i+1})}\rangle^2_\sigma$ s.t. $\langle\mathcal{I}_{x\in[e_i,e_{i+1})}\rangle^2_0 \oplus \langle\mathcal{I}_{x\in[e_i,e_{i+1})}\rangle^2_1 = \mathcal{I}_{x\in[e_i,e_{i+1})}$. However, $\mathsf{spline}_{n,\ell,d,P,E}$ is defined over the ring $\mathbb{Z}_{2^n}$ and thus an additional conversion is required to convert the indicators from the binary domain to the arithmetic domain before the spline function is evaluated. In addition, a key obstacle to achieving malicious security in GROTTO stems from its reliance on the XOR operation for the prefix-parity algorithm, which fundamentally limits the DPF output group to $\{0,1\}$. This limitation hinders any straightforward extension to the malicious setting.

4.2. Our Approach

To address the limitations of GROTTO, we make use of incremental DPF (IDPF) [18], which allows the online parties to generate an additive share of $\mathcal{I}_{x\in[0,e_i)}$ for each $i \in \{2,\ldots,\ell\}$ over the ring $\mathbb{Z}_{2^n}$ due to its special tree structure.

Recall that in the tree representation of IDPF for the all-prefix point function $f_{\alpha,\vec{\beta}}$ ($\vec{\beta} = (\beta_1,\ldots,\beta_n)$) with input domain $\mathbb{Z}_{2^n}$ and output group $\mathbb{G}$, each leaf node is labeled by a number in $\mathbb{Z}_{2^n}$. This tree representation has $n+1$ layers, where the root node is located at the layer 0 and the leaf nodes are located at the layer n. For each $i \in [n]$, the nodes at the layer i are labeled by numbers in $\mathbb{Z}_{2^i}$. For any node a located at the layer i, its parent node is labeled by $a_{[i-1]}$. For each $i \in [n]$, the node $\alpha_{[i]}$ holds the value β_i while all other nodes hold the value 0.

To evaluate $I_{\alpha\in[0,e)}$, we observe that a number α belongs to the interval $[0,e)$ if and only if there exists $i \in [n]$ such that the nodes $\alpha_{[i]}$ and $e_{[i]}$ have the same parent node (i.e., $\alpha_{[i-1]} = e_{[i-1]}$) and $\alpha_{[i]}$ is the left sibling of $e_{[i]}$, (i.e., $\mathsf{lsb}(\alpha_{[i]}) = 0$ and $\mathsf{lsb}(e_{[i]}) = 1$). Let $\mathsf{lsn}(e)$ denote the set of all left sibling nodes along the path from the root to the leaf node e. If we let $\vec{\beta} = \{1\}^n$ and assign a value 0 to all the other nodes, then the sum of the values of all nodes in $\mathsf{lsn}(e)$ is equal to $\mathcal{I}_{\alpha\in[0,e)}$ as at most one node in $\mathsf{lsn}(e)$ has value 1 according to **Lemma 1**.

Lemma 1. *For any $e \in \mathbb{Z}_{2^n}$ and $\alpha \in \mathbb{Z}_{2^n}$, there exists at most one $s \in \mathsf{lsn}(e)$ such that $s = \alpha_{[|s|]}$.*

Proof. The proof is straightforward. If there exist two distinct nodes $s_1, s_2 \in \mathsf{lsn}(e)$ such that $s_1 = \alpha_{[|s_1|]}$, $s_2 = \alpha_{[|s_2|]}$ and $|s_1| < |s_2|$, then node s_1 is located in the path from the root to node s_2, which contradicts the definition of $\mathsf{lsn}(e)$, as any two nodes in $\mathsf{lsn}(e)$ never locate in the path from the root to any leaf. □

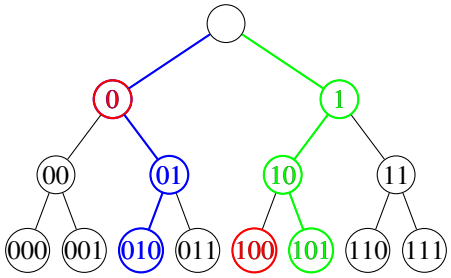

Figure 6: Example tree representation for $\mathbb{Z}_{2^3}$.

For example, see **Figure 6**, which is the binary tree representation of $\mathbb{Z}_{2^3}$. In this example, we have $e = 101$ and $\alpha = 010$. The path from the root to node e is colored in green, and the path from the root to node α is colored in blue. It's easy to see that $\mathsf{lsn}(e) = \{0, 100\}$ (colored in red), as node 0 lies in the path from the root to node α, it holds value 1, and node 100 holds value 0. Therefore, the sum of the values is 1, which is equal to $\mathcal{I}_{\alpha\in[0,e)}$.

Next, we will give a systematic description on how to securely evaluate the spline function $\mathsf{spline}_{n,\ell,d,P,E}$ in the $(2+1)$ party setting via IDPF. Formally, we utilize the IDPF construction from [18] with input domain $\{0,1\}^n$ and output group $\{\mathbb{G}_i = \mathbb{Z}_{2^n}\}_{i\in[n]}$. Similar to GROTTO, our approach requires an offline phase and an online phase, where the offline phase generates IDPF keys that can be used by the online phase to efficiently evaluate the spline function.

In the offline phase, the dealer $\mathcal{D}$ generates a pair of IDPF keys (k_0, k_1) for the all-prefix point function $f_{\alpha,\vec{\beta}=\{1\}^n}$, where $\alpha = \langle\alpha\rangle_0 + \langle\alpha\rangle_1 \bmod 2^n$ and $\langle\alpha\rangle_\sigma$ is selected randomly from the ring $\mathbb{Z}_{2^n}$ by P_σ for each $\sigma \in \{0,1\}$. Then $\mathcal{D}$ sends the IDPF key k_σ to P_σ.

In the online phase, the online parties P_0 and P_1 open the value $x - \alpha$ and cyclically shift each endpoint in E to the left by $x - \alpha$ to obtain the shifted endpoints set E' and corresponding polynomials P' as in GROTTO. W.l.o.g., we assume that $E' = (e'_1,\ldots,e'_{\ell'+1})$ and $P' = (p'_1,\ldots,p'_{\ell'})$ for $\ell' = \ell$ (overflow not occurs) or $\ell' = \ell+1$ (overflow occurs), where $e'_1 = 0$ and $e'_{\ell'+1} = 2^n$. For any $i \in [\ell']$, the polynomial $p'_i \in P'$ corresponding to the indicator $\mathcal{I}_{\alpha\in[e'_i,e'_{i+1})}$ is set to $p_j \in P$ ($j \in [\ell]$) such that $[e'_i, e'_{i+1}) \subseteq [e_j, e_{j+1}) \lll (x-\alpha)$. Then the value $\mathsf{spline}_{n,\ell,d,P,E}(x)$ can be computed from E' and P' as follows:

$$\mathsf{spline}_{n,\ell,d,P,E}(x) = \sum_{i\in[\ell']} \mathcal{I}_{\alpha\in[e'_i,e'_{i+1})} \cdot p'_i(x).$$

To compute the indicators $\{\mathcal{I}_{\alpha\in[e'_i,e'_{i+1})}\}_{i\in[\ell']}$, both parties evaluate the IDPF keys on each $z \in \mathsf{lsn}(e'_i)$ for each

$i \in \{2, \ldots, \ell'\}$ to generate the IDPF output $\langle y^{(z)} \rangle_0$ and $\langle y^{(z)} \rangle_1$ such that $\langle y^{(z)} \rangle_0 + \langle y^{(z)} \rangle_1 = f_{\alpha,\{1\}^n}(z)$. Recall that for any valid input $z \in \cup_{i\in[n]}\{0,1\}^i$, $f_{\alpha,\{1\}^n}(z) = 1$ if and only if $z = \alpha_{[|z|]}$. Thus, we have

$$\sum_{z\in \mathsf{lsn}(e'_i)} (\langle y^{(z)} \rangle_0 + \langle y^{(z)} \rangle_1) = 1$$

if and only if there exists a $z \in \mathsf{lsn}(e'_i)$ such that $z = \alpha_{[|z|]}$. To see why the above is true, we note that for any $e \in \{0,1\}^n$, there exists at most one node in $\mathsf{lsn}(e)$ holding the value 1 according to **Lemma 1**. Therefore, we can construct the additive share of the indicator $\mathcal{I}_{\alpha\in[e'_i,e'_{i+1})}$ for each $i \in [\ell']$ as follows:

$$\langle \mathcal{I}_{\alpha\in[e'_i,e'_{i+1})} \rangle_\sigma := \begin{cases} Y_\sigma^{(2)} & \text{if } i = 1, \\ \sigma - Y_\sigma^{(\ell')} & \text{if } i = \ell', \\ Y_\sigma^{(i+1)} - Y_\sigma^{(i)} & \text{otherwise,} \end{cases}$$

where $Y_\sigma^{(i)} := \sum_{z\in\mathsf{lsn}(e'_i)} \langle y^{(z)} \rangle_\sigma$ for each $i \in [\ell']$ and $\sigma \in \{0,1\}$. We claim that $\langle \mathcal{I}_{\alpha\in[e'_i,e'_{i+1})} \rangle_\sigma$ is an additive share of the indicator $\mathcal{I}_{\alpha\in[e'_i,e'_{i+1})}$ for each $i \in [\ell']$. To see this, we analyze the cases for each $i \in [\ell']$:

- For the case $i = 1$, we have
$$\langle \mathcal{I}_{\alpha\in[e'_1,e'_2)} \rangle_0 + \langle \mathcal{I}_{\alpha\in[e'_1,e'_2)} \rangle_1 = Y_0^{(2)} + Y_1^{(2)},$$
which is equal to 1 if and only if $\alpha \in [0, e'_2)$ since $e'_1 = 0$.
- For the case $i = \ell'$, we have
$$\langle \mathcal{I}_{\alpha\in[e'_{\ell'},e'_{\ell'+1})} \rangle_0 + \langle \mathcal{I}_{\alpha\in[e'_{\ell'},e'_{\ell'+1})} \rangle_1 = 1 - (Y_0^{(\ell')} + Y_1^{(\ell')}).$$
With the fact that $Y_0^{(\ell')} + Y_1^{(\ell')} = \mathcal{I}_{\alpha\in[0,e'_{\ell'})}$, we have $\langle \mathcal{I}_{\alpha\in[e'_{\ell'},e'_{\ell'+1})} \rangle_0 + \langle \mathcal{I}_{\alpha\in[e'_{\ell'},e'_{\ell'+1})} \rangle_1$ equals to 1 if and only if $\alpha \in [e'_{\ell'}, 2^n)$ since $e'_{\ell'+1} = 2^n$.
- For the other cases, we have
$$\langle \mathcal{I}_{\alpha\in[e'_i,e'_{i+1})} \rangle_0 + \langle \mathcal{I}_{\alpha\in[e'_i,e'_{i+1})} \rangle_1 = Y_0^{(i+1)} + Y_1^{(i+1)} - (Y_0^{(i)} + Y_1^{(i)}).$$
With the fact that $Y_0^{(i+1)} + Y_1^{(i+1)} = \mathcal{I}_{\alpha\in[0,e'_{i+1})}$ and $Y_0^{(i)} + Y_1^{(i)} = \mathcal{I}_{\alpha\in[0,e'_i)}$, we have $\langle \mathcal{I}_{\alpha\in[e'_i,e'_{i+1})} \rangle_0 + \langle \mathcal{I}_{\alpha\in[e'_i,e'_{i+1})} \rangle_1$ equals to 1 if and only if $\alpha \in [e'_i, e'_{i+1})$.

Finally, with the indicators $\{\mathcal{I}_{\alpha\in[e'_i,e'_{i+1})}\}_{i\in[\ell']}$, the two parties can locally compute

$$\langle \tilde{p} \rangle = \sum_{i=1}^{\ell'} \langle \mathcal{I}_{\alpha\in[e'_i,e'_{i+1})} \rangle \cdot p'_i \in \mathbb{Z}_{2^n}[X],$$

where $\langle \tilde{p} \rangle$ is an additive share of the polynomial $\tilde{p} \in \mathbb{Z}_{2^n}[X]$. Then, through some well-known techniques that can securely evaluate polynomials via secret shared coefficients and inputs such as Horner's method [33] together with Du-Atallah multiplication [34] and on ABY2.0-style D-ary multiplication [35], the two parties can securely generate the results of spline evaluated on x by computing $\langle \tilde{p}(x) \rangle$.

4.3. Defending Malicious Parties

4.3.1. Defending Against Malicious Dealer. As in the above approach, the dealer $\mathcal{D}$ receives the shares $\langle \alpha \rangle_0, \langle \alpha \rangle_1$ from the two parties P_0 and P_1, and generates the IDPF keys (k_0, k_1) for the all-prefix point function $f_{\alpha,\vec{\beta}=\{1\}^n}$. If the dealer $\mathcal{D}$ is controlled by a malicious adversary, it may deviate from the protocol in two ways:

1) It may generate the not well-formed IDPF keys (k_0, k_1) that have two or more different nonzero outputs at the same layer.
2) It may generate the IDPF keys (k_0, k_1) for a different all-prefix point function $f_{\alpha',\vec{\beta}'}$ with $\alpha' \neq \alpha$ or $\vec{\beta}' \neq \{1\}^n$.

For the first case, we replace the IDPF with the Verifiable IDPF (VIDPF) that proposed in Section 3. The VIDPF provides a verification algorithm Ver, execute by each online party, that can check whether the VIDPF keys (k_0, k_1) generated by the dealer $\mathcal{D}$ are well-formed, i.e., define a valid all-prefix point function.

For the second case, we let the parties P_0 and P_1 additional generate $\{\langle \alpha_i \rangle^2\}_\sigma$ (via $\mathcal{F}_{\mathsf{edaBits}}$), where α_i is the i-th bit of α. Then together with the random selected values $a, b, c, h \in \mathbb{Z}_{2^n}$, the parties can generate tags for α and $\vec{\beta} = \{1\}^n$ as follows:

$$\langle \eta_i \rangle^2 := a_i \wedge (\langle \alpha_i \rangle^2 \oplus b_i) \text{ for each } i \in [n],$$
$$\langle \theta \rangle := c \cdot (\langle 1 \rangle + h \cdot \langle m \rangle),$$

where $\langle m \rangle$ is a randomly generated additive shares. Then, the dealer $\mathcal{D}$ is required to not only generate the VIDPF keys (k_0, k_1) for $f_{\alpha,\vec{\beta}=\{1\}^n}$, but also generate the VIDPF keys $\tilde{k}_0, \tilde{k}_1$ for the $f_{\eta,\vec{\theta}=\{\theta\}^n}$. For any input $z \in \cup_{i\in[n]}\{0,1\}^i$, let $t_\sigma^{(z)}$ be the output layer's control bit for evaluating on z for each $\sigma \in \{0,1\}$, and $t_1^{(|z|)}$ be the $|z|$-th layer's control bit while generating the VIDPF keys (k_0, k_1). According to the correctness of VIDPF, we have $t_1^{(|z|)} = t_1^{(z)}$ and $t_0^{(z)} - t_1^{(z)} = \pm 1$ if and only if $z = \alpha_{[|z|]}$, which implies that $T = (t_0^{(z)} - t_1^{(z)}) \cdot (-1)^{t_1^{(|z|)}}$ equal to 1 if and only if $z = \alpha_{[|z|]}$ and 0 otherwise. Thus, for any input $z \in \cup_{i\in[n]}\{0,1\}^i$ it holds that:

$$c \cdot (f_{\alpha,\{1\}^n}(z) + T \cdot h \cdot m) = f_{\eta,\{\theta\}^n}(a_{[|z|]} \wedge (z \oplus b_{[|z|]})).$$

Therefore, to verify that the VIDPF keys k_0, k_1 from the dealer $\mathcal{D}$ define the desired $f_{\alpha,\{1\}^n}$, the online parties P_0 and P_1 can proceed as follows. They evaluate the keys k_0, k_1 on the set $Z \in \cup_{i\in[n]}\{0,1\}^i$ and the keys $\tilde{k}_0, \tilde{k}_1$ on the set $\{a_{[|z|]} \wedge (z \oplus b_{[|z|]}) \mid z \in Z\}$. By checking if the above equation holds for each $z \in Z$, the parties can confirm the validity of k_0, k_1. Additionally, the equations for different $z \in Z$ can be aggregated into a single equation for efficiency. Specifically, the parties P_0 and P_1 can jointly

Protocol Π_{CAVERN}

Let λ be the computational security parameter. Let VIDPF be a secure VIDPF for input domain $\{0,1\}^n$ and output group $\mathbb{G}_i = \mathbb{Z}_{2^{n+s}}^2$ for each $i \in [n]$. Let Π_{poly} be a subprotocol comes from [13] that evaluates polynomials with secret shared coefficients and inputs. The protocol Π_{CAVERN} consists of an offline phase Π_{off} and an online phase Π_{on}.

Π_{off} **(Offline Phase)**:

1) For each $\sigma \in \{0,1\}$, P_σ samples $\Delta_\sigma \leftarrow \mathbb{Z}_{2^s}$ and sends Δ_σ to $\mathcal{D}$.
2) P_0 and P_1 call $\mathcal{F}_{\mathsf{edaBits}}$ on input n to generate $[\![\alpha]\!], [\![m]\!]$ and its bit decomposition $\{[\![\alpha_i]\!]^2\}_{i\in[n]}, \{[\![m_i]\!]^2\}_{i\in[n]}$ and locally set $[\![\beta]\!]_\sigma := (\sigma, \Delta_\sigma) \in \mathbb{Z}_{2^{n+s}}^2$. Then P_0 and P_1 call $\mathcal{F}_{\mathsf{coin}}$ to sample $a, b, c, h \leftarrow \mathbb{Z}_{2^n}$. The parties P_0 and P_1 parse $a = a_1 \dots a_n$ and $b = b_1 \dots b_n$, and compute $\{[\![\eta_i]\!]^2 = a_i \wedge ([\![\alpha_i]\!]^2 \oplus b_i)\}_{i\in[n]}$, $[\![\theta]\!] = c \cdot ([\![\beta]\!] + h \cdot [\![m]\!])$. Then for each $\sigma \in \{0,1\}$, P_σ sends $(\{[\![\alpha_i]\!]_\sigma^2\}_{i\in[n]}, [\![\beta]\!]_\sigma, \{[\![\eta_i]\!]_\sigma^2\}_{i\in[n]}, [\![\theta]\!]_\sigma, [\![m]\!]_\sigma)$ to $\mathcal{D}$.
3) $\mathcal{D}$ opens the value $\alpha, \beta, \eta, \theta, m$ and checks all the MACs of these values, $\mathcal{D}$ aborts if any of the MACs cannot pass the check. Otherwise, $\mathcal{D}$ computes

$$(k_0, k_1, \{t_0^{(i)}, t_1^{(i)}\}_{i\in[n]}) \leftarrow \mathsf{VIDPF.Gen}(1^\lambda, \alpha, \{(\beta, \mathsf{M}_0[\beta] + \mathsf{M}_1[\beta])\}^n),$$
$$(\tilde{k}_0, \tilde{k}_1, \{\tilde{t}_0^{(i)}, \tilde{t}_1^{(i)}\}_{i\in[n]}) \leftarrow \mathsf{VIDPF.Gen}(1^\lambda, \eta, \{(\theta, \mathsf{M}_0[\theta] + \mathsf{M}_1[\theta])\}^n).$$

   $\mathcal{D}$ sends $(k_\sigma, \tilde{k}_\sigma, \{q^{(i)} := (-1)^{t_1^{(i)}} \cdot m\}_{i\in[n]})$ to P_σ for $\sigma \in \{0,1\}$.
4) Party P_σ outputs $(k_\sigma, \tilde{k}_\sigma, \{q^{(i)}\}_{i\in[n]})$.

Π_{on} **(Online Phase)**:

Input: A spline $\mathsf{spline}_{n,\ell,d,P,E}$. P_0 and P_1 hold an authenticated secret sharing $[\![x]\!] = ([\![x]\!]_0, [\![x]\!]_1)$ where x is the input of $\mathsf{spline}_{n,\ell,d,P,E}$.

1) The parties P_0 and P_1 compute $[\![x - \alpha]\!] := [\![x]\!] - [\![\alpha]\!]$ and open the value $x - \alpha$. Then, P_0 and P_1 cyclically shift each endpoint in E to the left by $x - \alpha$ to obtain the shifted endpoints set E'.
2) For each $\sigma \in \{0,1\}$, P_σ parses $E' = (e'_1, \dots, e_{\ell'+1})$ and $P' = (p'_1, \dots, p'_{\ell'})$. Let $Z := \{z \mid i \in \{2, \dots, \ell'\}, z \in \mathsf{lsn}(e'_i)\}$. Then P_σ computes

$$(\{[\![y^{(z)}]\!]_\sigma, t_\sigma^{(z)}\}_{z\in Z}, \mu_\sigma) \leftarrow \mathsf{VIDPF.Eval}(\sigma, k_\sigma, Z),$$
$$(\{[\![\tilde{y}^{(z)}]\!]_\sigma, \tilde{t}_\sigma^{(z)}\}_{z\in Z}, \tilde{\mu}_\sigma) \leftarrow \mathsf{VIDPF.Eval}(\sigma, \tilde{k}_\sigma, \{a_{[|z|]} \wedge (z \oplus b_{[|z|]}) \mid z \in Z\}).$$

   For each $i \in [\ell']$, P_0 and P_1 compute

$$[\![\mathcal{I}_{\alpha\in[e'_i, e'_{i+1})}]\!] := \begin{cases} \sum_{z\in\mathsf{lsn}(e'_2)} [\![y^{(z)}]\!] & \text{if } i = 1, \\ [\![\beta]\!] - \sum_{z\in\mathsf{lsn}(e_{\ell'})} [\![y^{(z)}]\!] & \text{if } i = \ell', \\ \sum_{z\in\mathsf{lsn}(e'_{i+1})} [\![y^{(z)}]\!] - \sum_{z\in\mathsf{lsn}(e'_i)} [\![y^{(z)}]\!] & \text{otherwise.} \end{cases}$$

   Then P_0 and P_1 locally compute $[\![\tilde{p}]\!] = \sum_{i=1}^{\ell'} [\![\mathcal{I}_{\alpha\in[e'_i, e'_{i+1})}]\!] \cdot p'_i$.
3) P_0 and P_1 call Π_{poly} with the secret shared coefficients $[\![\tilde{p}]\!]$ and input $[\![x]\!]$ to compute $[\![s]\!]$.
4) P_0 and P_1 call $\mathcal{F}_{\mathsf{coin}}$ to sample $\delta \in \mathbb{Z}_{2^n}$. For each $\sigma \in \{0,1\}$, P_σ parses $[\![y^{(z)}]\!]_\sigma = (y_\sigma^{(z)}, \mathsf{M}_\sigma[y^{(z)}]), [\![\tilde{y}^{(z)}]\!]_\sigma = (\tilde{y}_\sigma^{(z)}, \mathsf{M}_\sigma[\tilde{y}^{(z)}])$ for each $z \in Z$, computes

$$\omega_\sigma := c \cdot (\sum_{z\in Z} \delta^z \cdot (y_\sigma^{(z)} + (-1)^\sigma \cdot t_\sigma^{(z)} \cdot q^{(|z|)} \cdot h)) - \sum_{z\in Z} \delta^z \cdot \tilde{y}_\sigma^{(z)},$$

   and sends $(\mu_\sigma, \tilde{\mu}_\sigma, \omega_\sigma)$ to $P_{1-\sigma}$.
5) P_0 and P_1 check whether $\mu_0 = \mu_1$, $\tilde{\mu}_0 = \tilde{\mu}_1$ and $\omega_0 + \omega_1 = 0$ and run the batch check protocol on all the values that have been opened so far. If any of these checks fails, P_0 and P_1 abort. Otherwise, they output $[\![s]\!]$.

Figure 7: Maliciously secure $(2+1)$-PC protocol for evaluating splines.

sample a random value $\delta \leftarrow \mathbb{Z}_{2^n}$ via $\mathcal{F}_{\mathsf{coin}}$ and check the following equation instead:

$$c \cdot \sum_{z \in Z} \delta^z \cdot (f_{\alpha,\{1\}^n}(z) + T \cdot h \cdot m) = \sum_{z \in Z} \delta^z \cdot f_{\eta,\{\theta\}^n}(a_{[|z|]} \wedge (z \oplus b_{[|z|]})).$$

4.3.2. Defending Against Malicious Online Party. To defend against the malicious adversary that controls one of the two parties P_0 and P_1, we replace the additive secret sharing (in the form of $\langle\cdot\rangle$) with the SPD$\mathbb{Z}_{2^n}$ authenticated secret sharing [19] (in the form of $[\![\cdot]\!]$) for all the above shared values. The correctness of the aforementioned computational procedures remains fully preserved, since SPD$\mathbb{Z}_{2^n}$ authenticated secret sharing can be regarded as a generalization of additive secret sharing, inherently supporting the direct conversion of corresponding computational operations. Specifically, each party P_σ locally selects a global key $\Delta_\sigma \in \mathbb{Z}_{2^n}$, and any the value $x \in \mathbb{Z}_{2^n}$ that shared among the parties should include an additional MAC $\mathsf{M}[x]$ that is used to check the integrity of the value. Additionally, to make the output of VIDPF consists with the SPD$\mathbb{Z}_{2^n}$ authenticated secret sharing, we let the output group of VIDPF be $\mathbb{Z}^2_{2^{n+s}}$ instead. And the dealer $\mathcal{D}$ is required to generate the IDPF keys (k_0, k_1) for the all-prefix point function $f_{\alpha,\vec{\beta}}$ such that $\vec{\beta} = \{(1, \mathsf{M}[1])\}^n$.

4.4. The Full Protocol

Now we are ready to describe the full protocol for evaluating splines in the $(2+1)$ party setting. The full protocol is depicted in **Figure 7**. Our full protocol relies on the VIDPF construction $\mathsf{VIDPF} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Ver})$ in Section 3. We let the algorithm VIDPF.Gen additionally outputs $\{t_0^{(i)}, t_1^{(i)}\}_{i\in[n]}$, which is the control bits for each layer. Also, the algorithm VIDPF.Eval is modified to additionally outputs $\{t_\sigma^{(z)}\}_{z\in Z}$, where Z is the input sets and $t_\sigma^{(z)}$ is the $|z|$-th layer's control bit for evaluating z. We take the polynomial evaluation method in [35] as a subprotocol, namely Π_{poly}, which can securely evaluate a polynomial with secret shared coefficients and inputs in the 2PC model within one round communication.

The security of the full protocol can be stated as follows:

Theorem 2. *Given a secure VIDPF* VIDPF *with input domain* $\{0,1\}^n$ *and output group* $\mathbb{G}_i = \mathbb{Z}^2_{2^{n+s}}$ *for each* $i \in [n]$*, protocol* $\Pi_{\textsc{Cavern}}$ *in* **Figure 7** *securely realizes the functionality* $\mathcal{F}_{\mathsf{spline}}$ *with abort in the* $(\mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{edaBits}})$*-hybrid model and with statistical error in the presence of computationally-bounded malicious adversary controlling the dealer* $\mathcal{D}$ *or one of the two online parties* P_0 *and* P_1*.*

Proof. We break down the proof into two cases: (1) the dealer $\mathcal{D}$ is malicious, or (2) one of the online parties P_0 or P_1 is malicious.

$\mathcal{D}$ is Malicious. In this case, the adversarially corrupt party is the dealer $\mathcal{D}$ (denoted as $\mathcal{A}$). We construct a simulator $\mathcal{S}$ that interacts externally with the ideal functionality $\mathcal{F}_{\mathsf{spline}}$ and internally with the adversary $\mathcal{A}$ as follows:

1) $\mathcal{S}$ mimics the role of the two online parties P_0 and P_1 in the protocol Π_{off} via an internal run. If $\mathcal{A}$ outputs (abort) in step 3 of Π_{off}, $\mathcal{S}$ sends (abort) to $\mathcal{F}_{\mathsf{spline}}$ and aborts.
2) $\mathcal{S}$ receives $(k_0, \tilde{k}_0, \{q^{(i)}\}_{i\in[n]})$ and $(k_1, \tilde{k}_1, \{q^{(i)}\}_{i\in[n]})$ from $\mathcal{A}$.
3) $\mathcal{S}$ mimics the role of the two online parties P_0 and P_1 in the protocol Π_{on} via an internal run. If any of the checks in step 5 of Π_{on} fails, $\mathcal{S}$ sends (abort) to $\mathcal{F}_{\mathsf{spline}}$ and aborts. Otherwise, $\mathcal{S}$ sends (ok) to $\mathcal{F}_{\mathsf{spline}}$.

Note that $\mathcal{A}$ does not receive any output from the protocol. Thus, the transcript of the protocol for $\mathcal{A}$ is indistinguishable in the real and ideal worlds. We only need to establish that the honest parties receive their outputs correctly. First we note that the honest parties abort the protocol when step 5 of protocol Π_{on} fails, which indicates that $\mathcal{A}$ breaks the security of VIDPF or finds $\alpha', \vec{\beta}' = \{\beta'_1, \ldots, \beta'_n\}, \eta', \vec{\theta}' = \{\theta'_1, \ldots, \theta'_n\}$ such that

$$a \wedge (\alpha' \oplus b) = \eta'$$

and

$$c \cdot (\beta'_i + h \cdot m) = \theta'_i$$

for each $i \in [n]$, where $f_{\alpha',\vec{\beta}'} \neq f_{\alpha,\{\beta\}^n}$ and/or $f_{\eta',\vec{\theta}'} \neq f_{\eta,\{\theta\}^n}$. With the fact that the values a, b, c, h are uniformly sampled from $\mathbb{Z}_{2^n}$ and keep secret to $\mathcal{A}$, we can conclude that the real world protocol aborts under the same conditions as the ideal world protocol. Finally, if the protocol does not abort, the outputs $[\![s]\!]_0$ and $[\![s]\!]_1$ of the honest parties form randomized shares of $s = \mathsf{spline}_{n,\ell,d,P,E}(x)$.

P_0 or P_1 is Malicious. W.l.o.g., we assume the adversarially corrupt party is P_0 (denoted as $\mathcal{A}$). We construct a simulator $\mathcal{S}$ that interacts externally with the ideal functionality $\mathcal{F}_{\mathsf{spline}}$ and internally with the adversary $\mathcal{A}$ as follows:

1) $\mathcal{S}$ samples $\Delta_1 \leftarrow \mathbb{Z}_{2^s}$ and receives Δ_0 from $\mathcal{A}$.
2) $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{edaBits}}$ and $\mathcal{F}_{\mathsf{coin}}$, and gives $\mathcal{A}$ the values $([\![\alpha]\!]_0, [\![m]\!]_0, \{[\![\alpha_i]\!]^2_0\}_{i\in[n]}, \{[\![m_i]\!]^2_0\}_{i\in[n]}, a, b, c, h)$.
3) $\mathcal{S}$ receives $(\{[\![\alpha_i]\!]^2_0\}_{i\in[n]}, [\![\beta]\!]_0, \{[\![\eta_i]\!]^2_0\}_{i\in[n]}, [\![\theta]\!]_0, [\![m]\!]_0)$ from $\mathcal{A}$ and checks whether the following holds:

$$\{[\![\eta_i]\!]^2_0\}_{i\in[n]} = a \wedge (\{[\![\alpha_i]\!]^2_0\}_{i\in[n]} \oplus b),$$
$$[\![\theta]\!]_0 = c \cdot ([\![\beta]\!]_0 + h \cdot [\![m]\!]).$$

If the check fails, $\mathcal{S}$ sends (abort) to $\mathcal{F}_{\mathsf{spline}}$ and aborts.
4) $\mathcal{S}$ calls the PPT simulator $\mathsf{Sim}_{\mathsf{VIDPF}}$ for VIDPF as defined in Section 3 to generate two indistinguishable keys:

$$(k_0, \perp) \leftarrow \mathsf{Sim}_{\mathsf{VIDPF}}(1^\lambda, 0, X),$$
$$(\tilde{k}_0, \perp) \leftarrow \mathsf{Sim}_{\mathsf{VIDPF}}(1^\lambda, 0, X).$$

Then $\mathcal{S}$ randomly selects $q^{(i)} \leftarrow \{-r, r\}$ for each $i \in [n]$ with $r \leftarrow \mathbb{Z}_{2^n}$ and sends $(k_0, \tilde{k}_0, \{q^{(i)}\}_{i\in[n]})$ to $\mathcal{A}$.
5) $\mathcal{S}$ mimics the role of P_1 to open the value $x - \alpha$ by interacting with $\mathcal{A}$.

6) $\mathcal{S}$ generates $\mu_0', \tilde{\mu}_0', \omega_0'$ as in step 1 - 4 of Π_{on} via the knowledge of $k_0, \tilde{k}_0$ and sends $(\mu_1 = \mu_0', \tilde{\mu}_1 = \tilde{\mu}_0', \omega_1 = -\omega_0')$ to $\mathcal{A}$.
7) $\mathcal{S}$ receives $(\mu_0, \tilde{\mu}_0, \omega_0)$ from $\mathcal{A}$ and checks whether $\mu_0 = \mu_1$, $\tilde{\mu}_0 = \tilde{\mu}_1$ and $\omega_0 + \omega_1 = 0$. If any of these checks fails, $\mathcal{S}$ sends (abort) to $\mathcal{F}_{\mathsf{spline}}$ and aborts.
8) $\mathcal{S}$ calls the batch check simulator $\mathsf{Sim}_{\mathsf{BatchCheck}}$ from [19] to simulate the view of $\mathcal{A}$ in step 5 of Π_{on}.
9) If $\mathcal{S}$ receives (abort) from $\mathcal{A}$ in any of the above step, it sends (abort) to $\mathcal{F}_{\mathsf{spline}}$ and aborts.
10) If the protocol does not abort until this step, $\mathcal{S}$ sends (ok) to the functionality $\mathcal{F}_{\mathsf{spline}}$ and the output of $\mathcal{A}$ is set to be $[\![s]\!]_0$.

We show that all the real interaction transcripts can be simulated. The VIDPF keys generates by $\mathsf{Sim}_{\mathsf{VIDPF}}$ are indistinguishable from the real ones due to the security of VIDPF. Also, the randomly selected values $\{q^{(i)}\}_{i\in[n]}$ are indistinguishable from the real ones since the value r is uniformly random in $\mathbb{Z}_{2^n}$ and $t_1^{(i)}$ is hidden to $\mathcal{A}$. For the batch check, since the global mac key Δ is only known to $\mathcal{S}$ (hidden from $\mathcal{A}$), the view of $\mathcal{A}$ in the batch check can be simulated by $\mathsf{Sim}_{\mathsf{BatchCheck}}$. The output distribution is indistinguishable in both the real and ideal world since the output $[\![s]\!]_0$ forms randomized share of $\mathsf{spline}_{n,\ell,d,P,E}(x)$. □

4.5. Cost Analysis

In this section, we analyze the cost of our protocol Π_{CAVERN} for evaluating splines. The cost of our protocol is mainly determined by the cost of the VIDPF and polynomial evaluation. In terms of computational cost, we measure it by the number of PRG operations while evaluating the VIDPF. And the communication cost is measured by the number of bits transmitted between parties. Furthermore, when evaluating multiple splines, the verification in step 4-5 can be executed in batches to amortize its cost. Accordingly, the online cost excludes these steps.

Offline Phase. In the offline phase, the dealer $\mathcal{D}$ receives $(\Delta_\sigma, \{[\![\alpha_i]\!]^2_\sigma\}_{i\in[n]}, [\![\beta]\!]_\sigma, \{[\![\eta_i]\!]^2_\sigma\}_{i\in[n]}, [\![\theta]\!]_\sigma, [\![m]\!]_\sigma)$ from P_σ for $\sigma \in \{0,1\}$ as in step 1 -2. This incurs a communication cost of $(7s + 14n + 4ns)$-bits for the dealer. The dealer $\mathcal{D}$ then generates two pairs of VIDPF keys (k_0, k_1) and $(\tilde{k}_0, \tilde{k}_1)$ as in step 3 and sends them to P_0 and P_1 with additionally outputs $\{q^{(i)}\}_{i\in[n]}$, where each key is of size $(\lambda + 2n + 2ns + 5n\lambda + 2n^2)$ bits. In total, the dealer's computational cost is asymptotically $O(n(n+s)/\lambda)$ PRG operations for generating the VIDPF keys, and the total communication cost is $(7s + 18n + 2\lambda + 10n\lambda + 8ns + 6n^2)$ bits for the dealer in the offline phase.

Online Phase. In the offline phase, the parties open the value $x - \alpha$ and cyclically shift the endpoints in E to the left by $x - \alpha$ to obtain E' as in step 1. The communication cost of the opening is $O(n + s)$ bits. Then, the parties evaluate the VIDPF on the set Z as in step 2. The primary cost of this step can be measured by how many distinct nodes the VIDPF.Eval algorithm must traverse, which is upper bounded by ℓn as analyzed in [13]. Given that each traversal incurs $O((n + s)/\lambda)$ PRG operations, the online computational cost of our protocol is $O(\ell n(n+s)/\lambda)$ PRG operations. In step 3, the parties call the polynomial evaluation subprotocol Π_{poly}, which occurs a communication cost of $(d + 2)(n + s)$ bits as analyzed in [13]. In total, for each party P_σ ($\sigma \in \{0,1\}$), the online computational cost is $O(\ell n(n + s)/\lambda)$ PRG operations, and the online communication cost is $(d+3)(n+s)$ bits within two rounds.

5. Implementation and Benchmarks

To empirically evaluate the performance of CAVERN, we implemented our protocol in the setting of $(2+1)$-PC over the ring $\mathbb{Z}_{2^n}$, and compared its performance to prior works.

5.1. Setting and Implementation Details

Splines of Choice. We thoroughly evaluate our implementation by securely computing a set of nonlinear functions include *rectified linear unit* (ReLU), *Gaussian error function* (gerf), *hyperbolic tangent* (tanh), *logistic sigmoid function* (sigmoid), *logarithm base 2* (log2), and *reciprocal square root* (isqrt). These splines are also provided by the reference implementation of GROTTO, and we employ the same set of splines for our benchmarks. Note that isqrt is the costliest function in our benchmarks, requiring the highest number of parts to achieve the desired approximation accuracy.

Development Framework. Our implementation of CAVERN is written in C++. We follow the approach of Wang et al. [36] by using a fixed-key AES cipher to construct a Matyas-Meyer-Oseas [37] one-way compression function. We use AES-based PRFs to construct our PRGs and our hash functions. We use AES implementation from cryptoTools [38] that internally uses AES x86 instructions. The experiments were conducted using a single core of an Intel Xeon E-2286 CPU with 64GB RAM, running Ubuntu 20.04LTS.

Concrete Parameters. All our benchmarks set the computational security parameter to $\lambda = 126$[1], the statistical security parameter to $\kappa = 40$, the data size to $n = 64$ bits and $s = 64$ for the SPD$\mathbb{Z}_{2^n}$ authenticated secret sharing scheme.

5.2. Benchmark Results

We compare the performance of CAVERN with GROTTO [13] and SHARK [9], which are the most relevant prior works for securely evaluating splines. **TABLE 2** presents a detailed breakdown of the communication cost and execution times for several common splines when evaluating with CAVERN, SHARK, and GROTTO. The comparison includes metrics for both the offline phase (covering computation and communication overheads) and the online phase (similarly detailing computation, communication, and the number of communication rounds).

1. For $\lambda = 126$, it is possible to implement the PRG G in VIDPF using two calls to fixed-key AES [9], [36]

TABLE 2: Cost comparison for selected splines with CAVERN, SHARK, and GROTTO.

Function	Formula	Scheme	Parts	Degree	Offline		Online		
					comp.	comm.	comp.	comm.	rounds
ReLU	$\max(0, x)$	GROTTO	2	1	0.054ms	8.576KB	0.102ms	0.032KB	2
		SHARK			0.088ms	2.269KB	0.169ms	0.048KB	2
		CAVERN			0.105ms	8.440KB	0.229ms	0.064KB	2
gerf	$(2/\sqrt{\pi}) \int_0^x e^{-t^2} \mathrm{d}t$	GROTTO	70	3	0.064ms	1.192KB	4.782ms	0.056KB	2
		SHARK			0.092ms	2.464KB	6.712ms	0.080KB	2
		CAVERN			0.120ms	8.920KB	8.496ms	0.112KB	2
tanh	$(e^x - e^{-x})/(e^x + e^{-x})$	GROTTO	84	3	0.067ms	1.192KB	5.238ms	0.056KB	2
		SHARK			0.089ms	2.464KB	7.353ms	0.080KB	2
		CAVERN			0.124ms	8.920KB	10.446ms	0.112KB	2
sigmoid	$1/(1 + e^{-x})$	GROTTO	98	3	0.066ms	1.192KB	6.541ms	0.056KB	2
		SHARK			0.089ms	2.464KB	8.827ms	0.080KB	2
		CAVERN			0.119ms	8.760KB	11.439ms	0.112KB	2
log2	$\log_2(x)$	GROTTO	193	3	0.059ms	1.192KB	11.359ms	0.056KB	2
		SHARK			0.083ms	2.464KB	16.265ms	0.080KB	2
		CAVERN			0.102ms	8.760KB	19.784ms	0.112KB	2
isqrt	$1/\sqrt{x}$	GROTTO	330	3	0.056ms	1.192KB	18.397ms	0.056KB	2
		SHARK			0.087ms	2.464KB	28.179ms	0.080KB	2
		CAVERN			0.107ms	8.760KB	35.128ms	0.112KB	2

Offline Phase. During the offline phase of CAVERN, SHARK, and GROTTO, the dealer must (i) receive shares of α from both parties; (ii) generate the DPF keys for α; and (iii) send the shares to the two online parties. As shown in **TABLE 2**, across all splines, the offline computation cost of CAVERN is approximately 1.7-1.9 times that of GROTTO and about 1.2-1.5 times that of SHARK. This is because CAVERN *additionally requires the dealer to generate shares for the labels of* α to ensure the dealer honestly generates the DPF keys for α, incurring extra computation. In terms of offline communication, CAVERN is markedly higher than GROTTO (by roughly 7.5-8.0 times) and SHARK (by roughly 3.6-3.8 times) across all splines, since it *employs VIDPF to defend against malicious dealer*, which yields longer keys than the DPF and DCF used in GROTTO and SHARK. Also, it's worth noting that the offline cost of CAVERN, SHARK, and GROTTO is independent of the number of parts of the spline due to *the offline phase is designed to be independent of the function to be evaluated*.

Online Phase. For the online phase of CAVERN, SHARK, and GROTTO, the two online parties securely evaluate the spline on their shares of the input x. As shown in **TABLE 2**, the computational cost of CAVERN is approximately 1.7-2.2 times that of GROTTO and about 1.2-1.5 times that of SHARK for all the splines, which is mainly due to the online parties of CAVERN *required to evaluate two VIDPFs: one for evaluating the spline and another for defending against malicious dealer.* Additionally, it's worth noting that the online computational cost of CAVERN, SHARK, and GROTTO increases linearly with the number of parts of the spline, which is consistent with the analysis in Section 4.5. In terms of the communication cost, both CAVERN and GROTTO need to open the value $x-\alpha$ in the first round and open $d+3$ values (where d is the degree of polynomials in spline) for the spline evaluation in the second round. However, the communication cost of CAVERN is twice that of GROTTO for all the splines, which mainly due to the fact that CAVERN *employs SPD$\mathbb{Z}_{2^n}$ authenticated secret sharing scheme* to ensure the integrity of the results, which needs to transmit $n+s$ bits to open each value for security, while GROTTO only needs to transmit n bits to open each value. For SHARK, the online parities need to open $d+1$ values in the first round and 1 value in the second round, incurring $(d+2)(n+s)$ bits of communication. The experimental results show that CAVERN has a higher communication cost than SHARK for all the splines, which is consistent with the analysis above.

In a nutshell, CAVERN only imposes a constant factor overhead on the top of GROTTO and SHARK, while providing stronger security guarantees. Specifically, the employ of VIDPF and SPD$\mathbb{Z}_{2^n}$ authenticated secret sharing contribute to these increased costs. Regardless, one can conclude that CAVERN is efficient and practical for securely evaluating splines in the $(2+1)$ party setting with malicious security.

6. Conclusion

We introduced CAVERN, a maliciously secure $(2+1)$-PC protocol for evaluating splines on additively secret shared inputs over the ring $\mathbb{Z}_{2^n}$ in the preprocessing model, while ensuring that the protocol is secure against a malicious adversary controlling the dealer or one of the two online parties. Our protocol only imposes a constant factor overhead on the top of the semi-honest protocol GROTTO [13], and the trusted-dealer-based maliciously secure 2PC protocol SHARK [9], while providing stronger security guarantees. We demonstrated the practicality and effectiveness of CAVERN through implementing and evaluating on different splines, such as rectified linear unit, Gaussian error function, hyperbolic tangent, logistic sigmoid function, logarithm base

2, and reciprocal square root. We believe CAVERN facilitates real-world adoption of maliciously secure $(2+1)$-PC. There are several potential future directions for our research: (1) Extending CAVERN to support Privacy Preserving Machine Learning (PPML) applications, such as secure training and inference of neural networks, (2) Reducing the communication cost of CAVERN by employing more efficient cryptographic primitives, such as zero-knowledge fully linear interactive oracle proof (zk-FLIOP) [39], and (3) improving the efficiency of our construction for CAVERN using orthogonal optimization techniques, such as hardware acceleration.

Acknowledgments

The authors would like to thank the shepherd and the anonymous reviewers for their insightful comments and suggestions. The corresponding author Liang Feng Zhang's research is partially supported by the National Natural Science Foundation of China under Grant No. 62372299.

References

[1] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game, or a completeness theorem for protocols with honest majority," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 307–328.

[2] A. C.-C. Yao, "How to generate and exchange secrets," in *27th annual symposium on foundations of computer science (Sfcs 1986)*. IEEE, 1986, pp. 162–167.

[3] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 19–38.

[4] P. Bogetoft, I. Damgård, T. Jakobsen, K. Nielsen, J. Pagter, and T. Toft, "A practical implementation of secure auctions based on multiparty integer computation," in *International Conference on Financial Cryptography and Data Security*. Springer, 2006, pp. 142–147.

[5] R. Poddar, S. Kalra, A. Yanai, R. Deng, R. A. Popa, and J. M. Hellerstein, "Senate: a {Maliciously-Secure}{MPC} platform for collaborative analytics," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2129–2146.

[6] "Mpc deployments dashboard," https://mpc.cs.berkeley.edu/, accessed: 2025-5-29.

[7] A. Agarwal, S. Peceny, M. Raykova, P. Schoppmann, and K. Seth, "Communication-efficient secure logistic regression," in *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2024, pp. 440–467.

[8] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee, "Function secret sharing for mixed-mode and fixed-point secure computation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2021, pp. 871–900.

[9] K. Gupta, N. Chandran, D. Gupta, J. Katz, and R. Sharma, "Shark: Actively secure inference using function secret sharing," Cryptology ePrint Archive, Paper 2025/716, 2025. [Online]. Available: https://eprint.iacr.org/2025/716

[10] K. Gupta, D. Kumaraswamy, N. Chandran, and D. Gupta, "Llama: A low latency math library for secure inference," *Cryptology ePrint Archive*, 2022.

[11] N. Jawalkar, K. Gupta, A. Basu, N. Chandran, D. Gupta, and R. Sharma, "Orca: Fss-based secure training and inference with gpus," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 597–616.

[12] T. Ryffel, P. Tholoniat, D. Pointcheval, and F. Bach, "Ariann: Low-interaction privacy-preserving deep learning via function secret sharing," *arXiv preprint arXiv:2006.04593*, 2020.

[13] K. Storrier, A. Vadapalli, A. Lyons, and R. Henry, "Grotto: Screaming fast (2+ 1)-pc or $\mathbb{Z}_{2^n}$ via (2, 2)-dpfs," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2143–2157.

[14] A. Vadapalli, F. Bayatbabolghani, and R. Henry, "You may also like... privacy: Recommendation systems meet pir," *Proceedings on Privacy Enhancing Technologies*, 2021.

[15] S. Wagh, "Pika: Secure computation using function secret sharing over rings," *Proceedings on Privacy Enhancing Technologies*, 2022.

[16] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015, pp. 337–367.

[17] L. de Castro and A. Polychroniadou, "Lightweight, maliciously secure verifiable function secret sharing," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2022, pp. 150–179.

[18] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Lightweight techniques for private heavy hitters," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 762–776.

[19] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing, "Spd$\mathbb{Z}_{2^k}$: Efficient mpc mod 2^k for dishonest majority," in *Annual International Cryptology Conference. Springer*, 2018, pp. 769–798.

[20] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 619–631.

[21] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: Improvements and extensions," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 1292–1303.

[22] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *Journal of the ACM (JACM)*, vol. 27, no. 4, pp. 701–717, 1980.

[23] R. Zippel, "Probabilistic algorithms for sparse polynomials," in *International symposium on symbolic and algebraic manipulation*. Springer, 1979, pp. 216–226.

[24] W. Zhang, X. Guo, K. Yang, R. Zhu, Y. Yu, and X. Wang, "Efficient actively secure dpf and ram-based 2pc with one-bit leakage," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 561–577.

[25] S. Garg, A. Goel, A. Jain, G.-V. Policharla, and S. Sekar, "{zkSaaS}:{Zero-Knowledge}{SNARKs} as a service," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4427–4444.

[26] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zksnarks with universal and updatable srs," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2020, pp. 738–768.

[27] M. Hamilis, E. Boyle, N. Gilboa, Y. Ishai, and A. Nof, "Preprocessing for life: Dishonest-majority mpc with a trusted or untrusted dealer," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 41–41.

[28] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of CRYPTOLOGY*, vol. 13, pp. 143–202, 2000.

[29] O. Goldreich, *Foundations of Cryptography, Volume 2*. Cambridge university press Cambridge, 2004.

[30] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure mpc for dishonest majority–or: breaking the spdz limits," in *Computer Security–ESORICS 2013: 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings 18*. Springer, 2013, pp. 1–18.

[31] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl, "Improved primitives for mpc over mixed arithmetic-binary circuits," in *Annual international cryptology conference*. Springer, 2020, pp. 823–852.

[32] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM (JACM)*, vol. 33, no. 4, pp. 792–807, 1986.

[33] W. Horner, "A new method of solving numerical equations of all orders, by continuous approximation." in *Abstracts of the Papers Printed in the Philosophical Transactions of the Royal Society of London*, vol. 2. JSTOR, 1815, pp. 117–117.

[34] W. Du and M. J. Atallah, "Protocols for secure remote database access with approximate matching," in *E-Commerce Security and Privacy*. Springer, 2001, pp. 87–111.

[35] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "{ABY2. 0}: Improved {Mixed-Protocol} secure {Two-Party} computation," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2165–2182.

[36] F. Wang, C. Yun, S. Goldwasser, V. Vaikuntanathan, and M. Zaharia, "Splinter: Practical private queries on public data," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 299–313.

[37] "Generating strong one-way functions with cryptographic algorithm," *IBM Technical Disclosure Bulletin*, vol. 27, pp. 5658–5659, 1985.

[38] "Cryptotools," https://github.com/ladnir/cryptoTools, accessed: 2025-5-30.

[39] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Zero-knowledge proofs on secret-shared data via fully linear pcps," in *Annual International Cryptology Conference*. Springer, 2019, pp. 67–97.

Appendix A. Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

A.1. Summary

This paper presents a construction for secure piecewise polynomial computation for two parties with an additional offline helper and improves upon prior work by providing security in the malicious corruption setting (one out of three parties can be corrupt).

The implementation shows that the solution compared with the semi-honest case has comparable efficiency.

A.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field.
- Creates a New Tool to Enable Future Science.
- Addresses a Long-Known Issue.

A.3. Reasons for Acceptance

1) The primitive of spline computation is well-motivated.
2) The proposed solution does not suffer significantly from the upgrade in security from semi-honest to malicious.