Verifiable Secret Sharing Based on Fully Batchable Polynomial Commitment for Privacy-Preserving Distributed Computation

Xiangyu Kong [*] Min Zhang[*] Yu Chen[*]

Abstract

Privacy-preserving distributed computation enables a resource-limited client to securely delegate computations on sensitive data to multiple servers by distributing shares of the data. In such systems, verifiable secret sharing (VSS) is a fundamental component, ensuring secure data distribution and directly impacting the overall performance. The most practical approach to construct VSS is through polynomial commitment (PC), with two main research directions to improve the VSS efficiency. The first focuses on improving the dealer time by designing PC that supports batch evaluation, i.e., generating multiple evaluation&proof pairs in one shot. The second aims to reduce the broadcast cost by designing PC that supports batch opening, i.e., producing a compact proof for multiple evaluations.

Recently, Zhang et al. (Usenix Security 2022) proposed a transparent PC that supports batch evaluation and obtained a transparent VSS with optimal dealer time. However, their scheme does not support batch opening, leading to high broadcast costs in VSS. To the best of our knowledge, no transparent PC currently supports both batch evaluation and batch opening, thus limiting the performance of existing VSS schemes.

In this paper, we propose a transparent fully batchable polynomial commitment (TFB-PC), that simultaneously supports batch evaluation and batch opening. Leveraging TFB-PC, we present a VSS scheme with optimal complexity: $O(n \log n)$ dealer time, $O(n)$ participant time and $O(n)$ communication cost. Furthermore, we implement our VSS scheme and compare its performance with Zhang et al.'s VSS (the naive approach). Results show that our scheme achieves $954\text{-}27,595\times$ reduction in communication cost and a $1,028\text{-}1,155,106\times$ speed up in participant time for $2^{11}\text{-}2^{21}$ parties.

Keywords: Verifiable secret sharing, polynomial commitment, privacy-preserving distributed computation.

[*]Shandong University. Email: {xykong, zm_min}@mail.sdu.edu.cn, yuchen.prc@gmail.com

Contents

1 Introduction

Privacy-preserving distributed computation allows a resource-limited client to delegate sensitive computations to multiple servers while ensuring data privacy. The core component for securely distributing the data in such systems is verifiable secret sharing (VSS), which ensures the data is shared among a set of servers with security, even if the client or servers act maliciously.

An (n, t)-VSS scheme consists of two phases: *sharing* and *reconstruction*. The *sharing* phase, which is crucial for distributing data in privacy-preserving distributed computation, is further divided into two rounds: *dealing* and *complaint*. In the *dealing* round, the dealer (client) $\mathcal{D}$ first splits a secret s into n shares and distributes one share to each participant (server) $\mathcal{V}_i$ over a private channel. In the *complaint* round, each participant checks the validity of their received share and, if it is invalid, broadcasts a complaint against the dealer. If there are complaints, the dealer is either disqualified or broadcasts valid shares to resolve them. In the *reconstruction* phase, participants can recover the secret s by pooling at least $t + 1$ valid shares.

One of the most practical ways to construct VSS is through a polynomial commitment (PC) [KZG10, TCZ+20, ZXH+22], which we refer to as PC-based VSS hereafter. The polynomial commitment scheme allows a prover to commit to a polynomial f and later reveal the evaluation y at a queried point x to the verifier, along with a proof π that demonstrates $y = f(x)$. Given a polynomial commitment scheme, it is straightforward to develop an (n, t)-VSS scheme as below. In the *dealing* round of the *sharing* phase, to share a secret s, the dealer $\mathcal{D}$ first picks a random t-degree polynomial f such that $f(0) = s$, then commits to f using the PC and broadcasts the commitment c. Next, for each $i \in \{1, \ldots, n\}$, $\mathcal{D}$ computes the evaluation y_i and the corresponding proof π_i for $y_i = f(i)$, and sends (y_i, π_i) to $\mathcal{V}_i$ as the i-th share and the corresponding verification information over a private channel. In the *complaint* round of the *sharing* phase, each $\mathcal{V}_i$ checks the validity of y_i using π_i and c, and broadcasts a complaint if it is invalid. If there are more than t complaining participants, then $\mathcal{D}$ is disqualified. Otherwise, $\mathcal{D}$ broadcasts $\{y_i, \pi_i\}_{i \in I}$ where I is the set of all complaining participants. We refer to the case where there is no complaint as *the best case*, and to the case where there are t complaints as *the worst case*. In this work, we focus on the latter one. In the *reconstruction* phase, given at least $t + 1$ valid pairs of (y_i, π_i), anyone can recover s by reconstructing the polynomial f via Lagrange interpolation and computing $s = f(0)$.

In the realm of PC-based VSS, two batching features for polynomial commitments are utilized to improve the efficiency of the sharing phase. One is batch opening, which enables the generation of a compact batch proof for multiple evaluations, thereby reducing broadcast overhead in the *complaint* round, as illustrated in Fig.1. The other is batch evaluation, which allows for the the rapid computation of several (y_i, π_i) pairs in one shot, thus decreasing the dealer time in the *dealing* round, as shown in Fig.2.

A line of works [KZG10, TCZ+20, YLF+22, ZXH+22] demonstrated the impact of these features on VSS efficiency. In particular, Kate et al.[KZG10] first introduced batch opening to their polynomial commitment scheme, known as the KZG scheme, and obtained KZG-VSS scheme with $O(n^2)$ dealer time, $O(n \log^2 n)$ participant time and $O(1)$ broadcast in the complaint round. Later, Tomescu et al. [TCZ+20] proposed batch evaluation for KZG scheme, reducing the dealer time of KZG-VSS to $O(n \log n)$, but sacrificed the participant time. Subsequently, Zhang et al. [ZXH+22] achieved batch evaluation for KZG scheme by leveraging the Fast Fourier Transformation (FFT), obtaining the most efficient KZG-based VSS scheme without any trade-off. However, all of these VSS schemes require a trusted setup. To avoid this, Yurek et al. [YLF+22] present a VSS scheme based on transparent polynomial commitment. Concurrently, Zhang et al. [ZXH+22] proposed a transparent polynomial commitment scheme with batch evaluation, achieving $O(n \log n)$ prover time, which we refer to as $\text{ZXH}^{+}_{\text{trans}}$ scheme. Despite its computational efficiency, the $\text{ZXH}^{+}_{\text{trans}}$ scheme lacks support for batch opening, resulting in significant broadcast overhead during the complaint round of the $\text{ZXH}^{+}_{\text{trans}}$-VSS scheme.

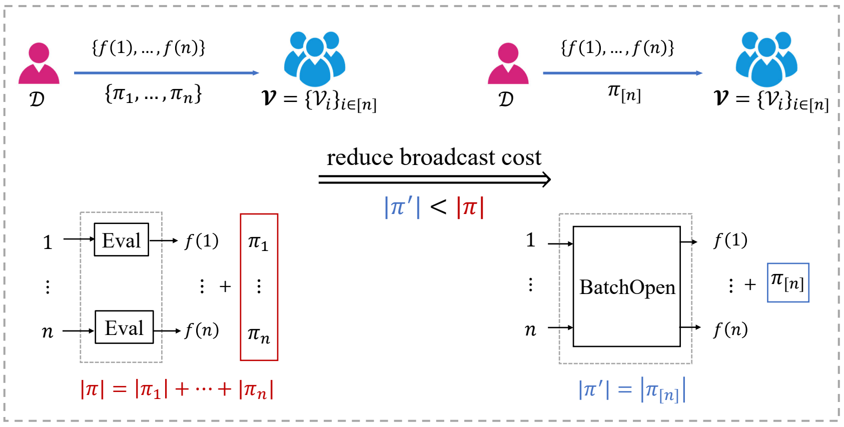

Figure 1: Batch opening to reduce broadcast cost

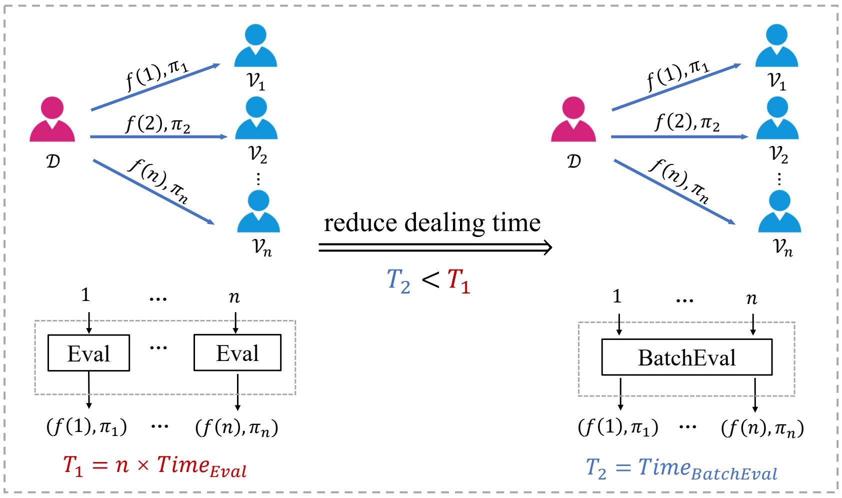

Figure 2: Batch evaluation to reduce dealing time

To our knowledge, no transparent polynomial commitment exists that supports both batch evaluation and batch opening, and the efficiency of existing VSS schemes remains unsatisfactory, particularly in large-scale privacy-preserving distributed computations. In light of this, we ask the question:

Can we construct a transparent polynomial commitment scheme that simultaneously supports batch evaluation and batch opening, while achieving a transparent VSS with optimal efficiency?

1.1 Our Contribution

In this paper, we propose a transparent, fully batchable polynomial commitment scheme and construct an optimal VSS scheme to answer the above question. Our contributions can be summarized as follows:

A fully batchable polynomial commitment scheme. We formalize the definitions of two standard features for polynomial commitments: batch evaluation and batch opening, and refer to polynomial commitments that support both as fully batchable polynomial commitments (FB-PC). Then we give the first construction of transparent FB-PC with $O(n \log n)$ prover time, $O(n)$ verifier time and $O(\log^2 n)$ proof size. Our scheme improves on the $\text{ZXH}^+_{\text{trans}}$ scheme [ZXH+22] by reducing the proof size and the verifier time by $O(n)$ and $O(\log^2 n)$ respectively, while maintaining the optimized efficiency of its prover time. A detailed comparison is provided in Table 1.

An optimal VSS scheme. We present a generic construction of VSS utilizing FB-PC, and yield an optimal VSS scheme by integrating our transparent FB-PC. Our VSS scheme has $O(n \log n)$ prover time, $O(n)$ participant time and $O(n)$ communication cost. Compared to existing VSS schemes, our instantiation stands out with comprehensive advantages. Details are provided in Table 2.

Implementation and evaluation. We implement our VSS scheme and evaluate its performance against the $\mathrm{ZXH}^{+}_{\mathrm{trans}}$-VSS [ZXH+22] in the same setting. Results show that, our VSS scheme has a comparable dealer time with $\mathrm{ZXH}^{+}_{\mathrm{trans}}$-VSS, while reducing its communication cost by 954-$27,595\times$, and improving the participant time by $1,028$-$1,155,106\times$ for 2^{11} to 2^{21} parties.

Table 1: Polynomial commitment with batching features: comparison with prior works.

Scheme	**Trans.**	**Batch evaluation**			**Batch opening**		
		Prover time	**Verifier time**	**proof size**	**Prover time**	**Verifier time**	**proof size**
KZG [KZG10]	×	$O(n^2)$	$O(1)$	$O(1)$	$O(n\log^2 n)$	$O(n\log^2 n)$	$O(1)$
AMT [TCZ+20]	×	$O(n\log n)$	$O(\log n)$	$O(\log n)$	$O(n\log^2 n)$	$O(n\log^2 n)$	$O(1)$
$\mathrm{ZXH}^{+}_{\mathrm{KZG}}$ [ZXH+22]	×	$O(n\log n)$	$O(1)$	$O(1)$	$O(n\log^2 n)$	$O(n\log^2 n)$	$O(1)$
hbACSS [YLF+22]	✓	$O(n^2)$	$O(n)$	$O(\log n)$	$O(n^2)$	$O(n^2)$	$O(n\log n)$
$\mathrm{ZXH}^{+}_{\mathrm{trans}}$ [ZXH+22]	✓	$O(n\log n)$	$O(\log^2 n)$	$O(\log^2 n)$	$O(n\log n)$	$O(n\log^2 n)$	$O(n\log^2 n)$
Our PC	✓	$O(n\log n)$	$O(\log^2 n)$	$O(\log^2 n)$	$O(n\log n)$	$O(n)$	$O(\log^2 n)$

We assume $deg(f) = \Theta(n)$. **Trans.** means without a trusted setup.

Table 2: Worst-case asymptotic complexity of VSS schemes.

Scheme	**Trans.**	**Dealer time**			**Participant time**			**Communication**			
		Dealing	Complaint	**Total**	Dealing	Complaint	**Total**	Priv. proof	Brd. proof	Brd. share	**Total**
Feldman-VSS [Fel87]	✓	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$
KZG-VSS [KZG10]	×	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	$O(n\log^2 n)$	$O(n\log^2 n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$
AMT-VSS [TCZ+20]	×	$O(n\log n)$	$O(n\log^2 n)$	$O(n\log^2 n)$	$O(\log n)$	$O(n\log^2 n)$	$O(n\log^2 n)$	$O(\log n)$	$O(\log n)$	$O(n)$	$O(n)$
$\mathrm{ZXH}^{+}_{\mathrm{KZG}}$-VSS [ZXH+22]	×	$O(n\log n)$	$O(n\log^2 n)$	$O(n\log^2 n)$	$O(1)$	$O(n\log^2 n)$	$O(n\log^2 n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$
hbACSS-VSS [YLF+22]	✓	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n^2)$	$O(n^2)$	$O(\log n)$	$O(n\log n)$	$O(n)$	$O(n\log n)$
$\mathrm{ZXH}^{+}_{\mathrm{trans}}$-VSS [ZXH+22]	✓	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(\log^2 n)$	$O(n\log^2 n)$	$O(n\log^2 n)$	$O(\log^2 n)$	$O(n\log^2 n)$	$O(n)$	$O(n\log^2 n)$
Our VSS	✓	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(\log^2 n)$	$O(n)$	$O(n)$	$O(\log^2 n)$	$O(\log^2 n)$	$O(n)$	$O(n)$

Assume $t = \Theta(n)$. Trans. means without a trusted setup. Dealing denotes the dealing round and Complaint denotes the complaint round. Priv. proof represents the private proof each participant receives via a private channel. Brd. share and Brd. proof denotes the broadcast shares and proofs in the complaint round, respectively.

1.2 Technique Overview

Revisiting the $\mathrm{ZXH}^{+}_{\mathrm{trans}}$-VSS. The $\mathrm{ZXH}^{+}_{\mathrm{trans}}$-VSS represents the current state-of-the-art PC-based VSS scheme, which stands out for eliminating the need for a trusted setup and achieving optimal dealer time. Given these advantages, we adopt it as the foundation for our work.

Since $\mathrm{ZXH}^{+}_{\mathrm{trans}}$-VSS scheme is built on the $\mathrm{ZXH}^{+}_{\mathrm{trans}}$ scheme, we revisit it at first. different form other works, Zhang et al. [ZXH+22] utilize the Fast Fourier transform (FFT) circuit [Wei69] to compute evaluations, moving away from traditional algebraic methods. Specifically, the prover runs the circuit C taking the coefficients of polynomial f as input and the evaluations $f(\omega_0), ..., f(\omega_{n-1})$ as output, where ω is the n-th root of unity. The C is structured as a layered arithmetic circuit with depth d over a finite field $\mathbb{F}$. Each gate in the i-th layer takes the outputs of two gates in the $(i+1)$-th layer, with layer 0 as the output layer and layer d as the input layer. They define a function $V_i : \{0,1\}^{s_i} \to \mathbb{F}$ that takes a binary string $\vec{b} \in \{0,1\}^{s_i}$ as input and returns the output of gate b in layer i, where b is the gate label and $s_i = \lceil \log S_i \rceil$, here S_i is the number of gates in the i-th layer. Thus, they denote the entire output (the evaluations) of circuit C as $V_0(\vec{x})$ and the input (the polynomial coefficients) as $V_d(\vec{x})$ for $\vec{x} \in \{0,1\}^{\log n}$.

When committing to the polynomial f, the prover utilizes the multivariate polynomial commitment to create a commitment for $\tilde{V}_d(\vec{x})$, the multilinear extension of $V_d(\vec{x})$. After the verifier queries the evaluation $f(\omega_i)$, the prover sends the i-th output out_i of circuit to the verifier and claims that it equals $f(\omega_i)$. For proving the correctness of out_i, the prover must convince the verifier of the statement: $\mathsf{out}_i = \sum_{\vec{x}\in\{0,1\}^{\log n}} \tilde{\beta}(\vec{i},\vec{x})\tilde{V}_0(\vec{x})$, where $\beta(\vec{i},\vec{x})$ is the identity function to select the i-th output of the circuit.

Efficient batch opening for $\mathrm{ZXH}^{+}_{\mathrm{trans}}$. When proving $|I|$ evaluations to a verifier, the prover needs to convince the verifier about the validity of $|I|$ statements:

$$\{\mathsf{out}_i = \sum_{\vec{x}\in\{0,1\}^{\log n}} \tilde{\beta}(\vec{i},\vec{x})\tilde{V}_0(\vec{x})\}_{i\in I}.$$

This approach would result in a proof size that scales linearly with $|I|$. An intriguing question is whether the proof size can be reduced to match that of a single statement.

Batch proving protocol for multi-output circuit. Before answering the above question, a more general problem arises. Suppose there is a circuit C with n distinct outputs, the verifier is concerned with receiving and verifying a subset of $|I|$ outputs. Can the prover generate a compact batching proof for these $|I|$ outputs? A straightforward method of independently applying the protocol for each of the $|I|$ outputs would result in a proof size of at least $O(|I| \cdot d \log |C|)$.

The batch selective function. Inspired by the selective identity function, we explore the possibility of designing a batch selection function that aggregates multiple outputs into a single statement.

The first attempt is summing up all the identity functions $\tilde{\beta}(\vec{i}, \vec{x})$ for $i \in I$ to construct the batch selection function $\sum_{i\in I} \tilde{\beta}(\vec{i}, \vec{x})$. Consequently, the batch statement can be expressed as:

$$\sum_{i\in I} \mathsf{out}_i = \sum_{\vec{x}\in\{0,1\}^{\log n}} \sum_{i\in I} \tilde{\beta}(\vec{i}, \vec{x}) \tilde{V}_0(\vec{x}).$$

However, this method is insecure, because the prover is only bound to $\sum_{i\in I} \mathsf{out}_i$ rather than to each individual output $\{\mathsf{out}_i\}_{i\in I}$.

To get around this problem, we adjust the batch selective function by using the random linear combination, leading to the following batch statement:

$$\sum_{i\in I} r^k \mathsf{out}_i = \sum_{\vec{x}\in\{0,1\}^{\log n}} \sum_{i\in I} r^k \tilde{\beta}(\vec{i}, \vec{x}) \tilde{V}_0(\vec{x}),$$

where k is the position of i in set I.

The Schwartz-Zippel Lemma guarantees that if the batching statement is valid, then the $|I|$ statements for every $i \in I$ are valid. Consequently, the prover just needs to convince the validity of the batching statement to verifier with the proof size of $O(d \log |C| + \log^2 n)$, independent of $|I|$.

1.3 Related Work

Polynomial commitment. Kate, Zaverucha, and Goldberg [KZG10] introduced the concept of polynomial commitment and provided two constructions that the commitment and proof have constant size. They also proposed batch opening for proving n evaluations to a verifier such that the prover only needs to compute a batch proof that is smaller than the size of n proofs. Later, Boneh et al. [BDFG20] generalized batch opening from the same polynomial to different polynomials, where a single group element can be a batching proof for multiple polynomials each evaluated at a different arbitrary subset of points.

Recent works [TCZ+20, YLF+22, ZXH+22] have focused on improving the prover's efficiency for computing multiple proofs to multiple verifiers. Tomescu et al. [TCZ+20] showed how to compute n evaluation proofs in $O(n \log n)$ time compared to $O(n^2)$ in KZG scheme [KZG10], but introduced a logarithmic overhead on the verification time and proof size. To address this problem, Zhang et al. [ZXH+22] implement batch evaluation for KZG scheme [ZXH+22] using FFT on group elements, achieving the same prover time without increasing the proof size and verification time. They also constructed a transparent polynomial commitment scheme where the prover can produce n proofs in $O(n \log n)$ time. It removed the trusted setup using GKR protocol and Virgo [ZXZS20] and only incurred cheap symmetric-key operations such as hashing and field arithmetic instead of the costly modular exponentiation.

Verifiable secret sharing. Chor et al [CGMA85] first introduced VSS. Feldman [Fel87] constructed the first efficient VSS scheme, known as Feldman-VSS, with computational hiding and information-theoretic binding. Later, Pedersen introduced its counterpart with information-theoretic hiding and computational binding. However, both schemes require $O(n)$ broadcast during the dealing round. To address the problem, Kate et al. [KZG10] constructed an eVSS scheme that reduces the broadcast cost and the verification time to $O(1)$ using the constant-sized polynomial commitment. Unfortunately, they increased the dealing time to $O(n^2)$ due to computing $O(n)$ shares and proofs in the dealing round. They also used the polynomial commitment with batch opening to reduce the broadcast of the complaint round.

Tomescu et al. [TCZ+20] proposed AMT-VSS from the polynomial commitment with batch evaluation to reduce the dealing time of eVSS from $O(n^2)$ to $O(n \log n)$. However, the communication and verification time for each participant also increases to $O(\log n)$. To deal with this, Zhang et al. [ZXH+22] constructed $\mathrm{ZXH}^{+}_{\mathrm{KZG}}$-VSS based on their $\mathrm{ZXH}^{+}_{\mathrm{KZG}}$ polynomial commitment scheme, with the broadcast proof size and verification time for each participant being $O(1)$. The above two VSS schemes also support batch opening in the complaint round. Their dealer time is $O(n \log^2 n)$, participant time is $O(n \log^2 n)$ and the communication is $O(n)$ in the sharing phase. However, all the bove VSS scheme require a trusted setup, to remove it, Yurek et al. [YLF+22] gives a VSS scheme from transparent polynomial commitment. To reduce the prover time, Zhang et al. [ZXH+22] constructed a transparent $\mathrm{ZXH}^{+}_{\mathrm{trans}}$-VSS scheme from their $\mathrm{ZXH}^{+}_{\mathrm{trans}}$ polynomial commitment scheme that the overall dealer time is $O(n \log n)$. But their broadcast overhead and the verification time for each participant are $O(n \log^2 n)$.

2 System Architecture

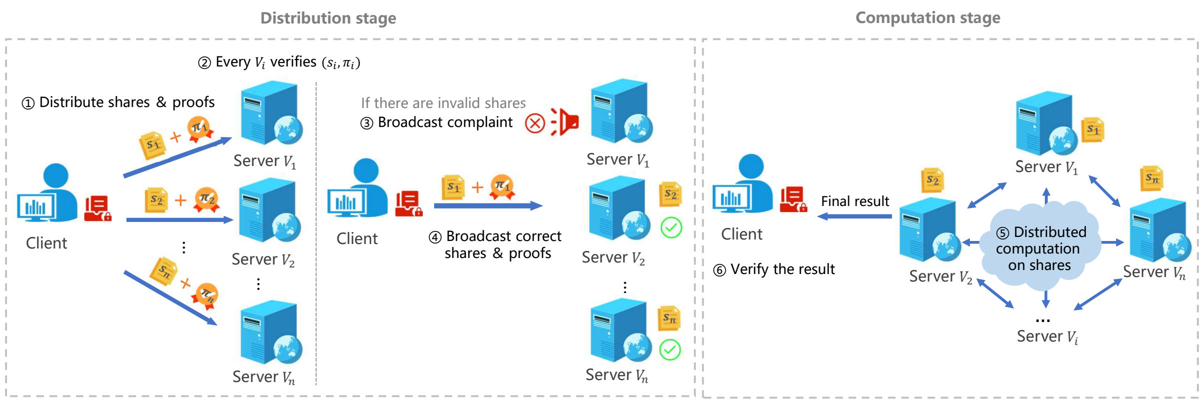
Distribution stage
Computation stage
② Every V_i verifies (s_i, π_i)
① Distribute shares & proofs
If there are invalid shares
③ Broadcast complaint
Server V_1
Server V_2
Server V_n
Client
④ Broadcast correct shares & proofs
Final result
⑥ Verify the result
⑤ Distributed computation on shares
Server V_i

Figure 3: Architecture of privacy-preserving distributed computation

In this section, we introduce the system and security model, focusing primarily on the distribution of data.

2.1 System Model

Fig. 3 presents the system model, which comprises a client and multiple servers.

Client: The client owns the secret data and wants to perform computations on it, but has limited computational resources. It distributes the data shares to each server and attaches a proof to ensure the share is correct.

Servers: There are n servers $\mathcal{V}_1, \cdots, \mathcal{V}_n$ in system model, every server $\mathcal{V}_i$ needs to receive the correct share and perform the distributed computation.

As depicted in Fig. 3, the privacy-preserving distributed computation consists of two stages: the distribution and computation. In the distribution stage, the client sends secret shares and proofs to the servers. Each server verifies the validity of its share; if correct, it accepts the share; otherwise, it rejects it and broadcasts a complaint to accuse the client. Then, the client broadcast the correct shares and proofs for all complaining servers. In the computation stage, servers perform the computation on the shares collaboratively, and send the results to the client.

2.2 Security Model

The client may act maliciously to distribute incorrect shares, and the servers may also act maliciously to return incorrect computation results and attempt to obtain as much sensitive information as possible about the entire secret data.

It's required that if the malicious client generate incorrect shares, then with a high probability the servers would reject and cannot output the correct computation result. Otherwise, the servers should return correct computation results. Additionally, the servers can not learn extra sensitive information about the secret data except the share and public information.

3 Preliminaries

Notations. The security parameter is denoted by λ, and PPT stands for Probabilistic Polynomial Time. The $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ denotes the negligible function, where for each positive polynomial $p(\cdot)$, there exists $\mathsf{negl}(k) < \frac{1}{p(k)}$ for sufficiently large integer k. For set S, let $|S|$ denote the size of set S. Let $[n]$ denote the set $\{0, 1, \cdots, n-1\}$. $y \overset{r}{\leftarrow} S$ denotes picking y uniformly at random from the set S and $|\pi|$ denotes the length of π.

3.1 Verifiable Secret Sharing

An (n, t)-VSS scheme among n participants $\mathcal{V}_i, \cdots, \mathcal{V}_n$ with a dealer $\mathcal{D}$ consists of two phases: a sharing phase and a reconstruction phase.

- Sharing phase: The sharing phase consists of two rounds. *Dealing round:* the dealer splits a secret s into n shares and distributes share s_i and the verification information π_i to participant $\mathcal{V}_i$ for all $i \in [n]$ over a private channel. *Complaint round:* each participant $\mathcal{V}_i$ checks the validity of its share s_i using the verification information π_i and broadcasts a complaint if it is invalid. The dealer takes some mechanisms to resolve complaints.
- Reconstruction phase: If any set of $t+1$ or more participants $\mathcal{V}_i$ publish their accepted shares along with the corresponding verification information (i, s_i, π_i), all $t+1$ or more participants $\mathcal{V}_i$ verify each of the broadcast shares (i, s_i, π_i). If more than t shares are valid, the secret s can be recovered using any $t+1$ valid pairs (i, s_i).

An (n, t)-VSS scheme has two properties:

- Correctness. If the dealer is honest, then the honest participants output the secret s at the end of the reconstruction phase.
- Secrecy. The adversary who can corrupt fewer than $t+1$ participants has no additional information about the secret s.

3.2 Polynomial Commitment

Definition 3.1 (**Polynomial Commitment**)**.** A polynomial commitment for univariate polynomial $f \in \mathbb{F}^d[X]$ and $z \in \mathbb{F}$ consists of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, d)$: on input the security parameter λ and a bound d on the degree of the polynomial, outputs the public parameter pp (which implicitly defines the message space $\mathcal{F}$, randomness space $\mathcal{R}$, and commitment space $\mathcal{C}$). We assume that all other algorithms input pp which we omit.
- $c \leftarrow \mathsf{Commit}(f)$: on input a polynomial $f(x) = \sum_{i=0}^{d} c_i x^i$, outputs the commitment c that the prover commits to f.
- $\{0,1\} \leftarrow \mathsf{VerifyPoly}(c, f, d)$: on input the commitment c and the opened polynomial f verifies that if c is the commitment to f. If so, outputs 1, otherwise outputs 0.
- $(y, \pi_x) \leftarrow \mathsf{Eval}(f, x)$: on input the polynomial f and an evaluation point x, outputs $y = f(x)$ and the corresponding proof π_x.
- $\{0,1\} \leftarrow \mathsf{Verify}(c, x, y, \pi_x)$: on input the commitment c, the evaluation point x, the answer y and the proof π_x, the verifier checks the correctness of the evaluation. [1]

A polynomial commitment scheme satisfies the following properties:

- **Correctness.** For any polynomial $f \in \mathbb{F}^d[X]$ and $x \in \mathbb{F}$, the following probability is 1:

$$\Pr\left[\begin{array}{c} \mathsf{VerifyPoly}(c, f) = 1 \wedge \\ \mathsf{Verify}\,(c, x, y, \pi_x) = 1 \end{array} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, d) \\ c \leftarrow \mathsf{Commit}\,(f) \\ (y, \pi_x) \leftarrow \mathsf{Eval}\,(f, x) \end{array}\right].$$

[1]For some schemes [GWC19, ZXZS20, BDFG21], algorithms Eval and Verify are derived from an interactive public-coin protocol through Fiat-Shamir transformation.

- **Polynomial binding.** A polynomial commitment scheme is polynomial binding if for all PPT adversaries $\mathcal{A}$, the following probability is $\mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{c}\mathsf{VerifyPoly}\,(c, f) = 1 \wedge \\ \mathsf{VerifyPoly}(c, f') = 1 \wedge \\ f \neq f'\end{array} : \begin{array}{c}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, d) \\ (c, f, f') \leftarrow \mathcal{A}(\mathsf{pp})\end{array}\right].$$

- **Evaluation binding.** For any polynomial f, evaluation point $x \in \mathbb{F}$ and $y \neq y'$, any PPT adversaries $\mathcal{A}$, the following probability is $\mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{c}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, d), \\ (c, \langle x, y, \pi_x\rangle, \langle x, y', \pi'_x\rangle) \leftarrow \mathcal{A}(\mathsf{pp}) : \\ \mathsf{Verify}\,(c, x, y, \pi_x) = 1 \wedge \\ \mathsf{Verify}\,(c, x, y', \pi'_x) = 1 \wedge y \neq y'\end{array}\right].$$

Definition 3.2 (**Multivariate polynomial commitment**)**.** The polynomial commitment could be extended for multivariate polynomials $f \in \mathcal{F} : \mathbb{F}^\ell \to \mathbb{F}$. The algorithms and definitions are similar to those of the univariate polynomial, and we use MVPC to denote the multivariate polynomial commitment schemes.

3.3 Interactive Proofs for Layered Circuits

Sumcheck Protocol. The sumcheck protocol, introduced by Lund et al. [LFKN90], is an interactive proof that plays a key role in probabilistic proofs and cryptography protocols. The sumcheck protocol is designed to prove that a polynomial sums to the specified value over a defined domain. More specifically, the prover wishes to convince the verifier of the statement $H = \sum_{b_1, b_2, ..., b_\ell \in \{0,1\}^\ell} f(b_1, b_2, ..., b_\ell)$, where f is a ℓ-variate polynomial of degree d. The protocol involves ℓ rounds of interaction between the prover and the verifier, employing a recursive structure. In this structure, statements about polynomials with arity l are recursively reduced to statements about polynomials with smaller arity, until arity 0 is reached. At the final stage of the protocol, the verifier queries an oracle for the polynomial f at a random point $(r_1, \cdots, r_\ell)$, obtaining the evaluation $f(r_1, \cdots, r_\ell)$ to complete the protocol. The sumcheck protocol is complete and has a soundness error $\epsilon = \frac{d\ell}{|\mathbb{F}|}$. The proof size and verification time of the protocol are both $O(d\ell)$, where d is the degree of f.

GKR Protocol. Goldwasser, Kalai, and Rothblum [GKR08] proposed an interactive proof for circuit evaluation, which is known as the GKR protocol. In this protocol, the prover and verifier agree on a log-space uniform arithmetic circuit C of fan-in 2 over a finite field $\mathbb{F}$. The goal is for the prover to convince the verifier of the value of the output gate(s) of C. Following the convention in prior works [CMT12, Tha13, CFS17, XZZ+19, ZXZS20, ZXH+22], let C be a layered arithmetic circuit with depth d, where each gate in the i-th layer takes the outputs of two gates in the $(i+1)$-th layer, with layer 0 representing the output layer and layer d representing the input layer. The values in layer i can be written as the sumcheck equation of the values in layer $i+1$. Let S_i denote the number of gates in the i-th layer and $s_i = \lceil \log S_i \rceil$. We define a function $V_i : \{0,1\}^{s_i} \to \mathbb{F}$ that takes a binary string $\vec{b} \in \{0,1\}^{s_i}$ as input and returns the output of gate b in layer i, where b is called the gate label and $\vec{b}$ is its binary representation. With this definition, V_0 corresponds to the output of the circuit, and V_d corresponds to the input layer. Additionally, we define two functions $\mathrm{add}_i, \mathrm{mult}_i : \{0,1\}^{s_{i-1}+2s_i} \to \{0,1\}$, referred to as wiring predicates in the literature. add_i (mult_i) takes one gate label $\vec{z} \in \{0,1\}^{s_{i-1}}$ in layer $i-1$ and two gate labels $\vec{x}, \vec{y} \in \{0,1\}^{s_i}$ in layer i, and outputs 1 if and only if gate $\vec{z}$ is an addition (multiplication) gate that takes the output of gates $\vec{x}, \vec{y}$ as input. By taking their multilinear extensions, for any $\vec{g}^{(i)} \in \mathbb{F}^{s_i}$, $\tilde{V}_i$ can be written as:

$$\begin{aligned}\tilde{V}_i(\vec{g}^{(i)}) &= \sum_{\vec{x}, \vec{y} \in \{0,1\}^{s_{i+1}}} \widetilde{\mathrm{add}}_{i+1}(\vec{g}^{(i)}, \vec{x}, \vec{y}) \left(\tilde{V}_{i+1}(\vec{x}) + \tilde{V}_{i+1}(\vec{y})\right) \\ &\quad + \widetilde{\mathrm{mult}}_{i+1}(\vec{g}^{(i)}, \vec{x}, \vec{y}) \left(\tilde{V}_{i+1}(\vec{x})\tilde{V}_{i+1}(\vec{y})\right) \\ &= \sum_{\vec{x}, \vec{y} \in \{0,1\}^{s_{i+1}}} f_i(\vec{g}^{(i)}, \vec{x}, \vec{y}). \end{aligned} \tag{1}$$

The protocol begins when the prover sends the claimed output of the circuit to the verifier. From the claimed output, the verifier computes the multilinear function $\tilde{V}_0$ based on the claimed output, then picks a random $\vec{g}^{(0)} \in \mathbb{F}^{s_0}$ and computes $\tilde{V}_0(\vec{g}^{(0)})$. The prover needs to convince the verifier that: $\tilde{V}_0(\vec{g}^{(0)}) = \sum_{\vec{x},\vec{y}\in\{0,1\}^{s_1}} f_1(\vec{g}^{(0)},\vec{x},\vec{y})$. To do so, the prover and verifier invoke a sumcheck protocol on it. As described in 3.3, at the end of the sumcheck, the verifier needs an oracle access to $f_1(\vec{g}^{(0)},\vec{u},\vec{v})$, where $\vec{u},\vec{v}$ are randomly selected in $\mathbb{F}^{s_1}$. To compute $f_1(\vec{g}^{(0)},\vec{u},\vec{v})$, the verifier computes $\widetilde{\mathrm{add}}_1(\vec{g}^{(0)},\vec{u},\vec{v})$ and $\widetilde{\mathrm{mult}}_1(\vec{g}^{(0)},\vec{u},\vec{v})$ locally (since these functions are publicly known), asks the prover to send $\tilde{V}_1(\vec{u})$ and $\tilde{V}_1(\vec{v})$ computes $f_1(\vec{g}^{(0)},\vec{u},\vec{v})$ to complete the sumcheck protocol. In this way, the prover and verifier reduce a claim about the output at layer 0 to two claims about values at layer 1. They could invoke two sumcheck protocols on $\tilde{V}_1(\vec{u})$ and $\tilde{V}_1(\vec{v})$ recursively to layers above, but the number of claims and the sumcheck protocols would grow exponentially in d.

To reduce two claims $\tilde{V}_1(\vec{u})$ and $\tilde{V}_1(\vec{v})$ to one, the verifier defines a line $\ell : \mathbb{F} \to \mathbb{F}^{s_1}$ as the unique line such that $\ell(0) = u, \ell(1) = v$. The verifier sends $\ell(x)$ to the prover, who responds with a degree s_i univariate polynomial $h(x) = \tilde{V}_1(\ell(x))$. The verifier checks that $h(0) = \tilde{V}_1(\vec{u})$ and $h(1) = \tilde{V}_1(\vec{u})$. Then, the verifier randomly selects $r \in \mathbb{F}$ and computes a new claim $h(r) = \tilde{V}_1(\ell(r)) = \tilde{V}_1(\vec{g^{(1)}})$ on $\vec{g}^{(1)} = \ell(r)$. The verifier then sends r and $\vec{g}^{(1)}$ to the prover. In this way, the two claims are reduced to one claim $\tilde{V}_1(\vec{g}^{(1)})$. Combining this protocol with the sumcheck protocol on Equation 1, the prover and verifier can recursively reduce a claim on layer i to a claim on layer $i+1$, eventually arriving at a claim for the input layer $\tilde{V}_d(\vec{g}^d)$. Notably, since the verifier knows the input of C, it can validate the last statement $\tilde{V}_d(\vec{g}^d)$ directly.

Theorem 3.1. *Let $C : \mathbb{F}^m \to \mathbb{F}^n$ be a depth-d layered arithmetic circuit. There exists an interactive proof protocol for the function computed by C with soundness $O(d\log|C|/|\mathbb{F}|)$. The total communication is $O(d\log|C|)$ and the running time of the prover is $O(|C|)$. When C has regular wiring pattern* [2]*, the running time of the verifier is $O(m+n+d\log|C|)$.*

Multilinear extension. Let $V : \{0,1\}^\ell \to \mathbb{F}$ be a function. The multilinear extension of V is the unique polynomial $\tilde{V} : \mathbb{F}^\ell \to \mathbb{F}$ s.t. $\tilde{V}(x_1, x_2, \ldots, x_\ell) = V(x_1, x_2, \ldots, x_\ell)$ for all $x_1, x_2, \ldots, x_\ell \in \{0,1\}$. $\tilde{V}$ can be expressed as:

$$\tilde{V}(x_1, x_2, \ldots, x_\ell) = \sum_{\vec{b}\in\{0,1\}^\ell} \prod_{i=1}^{\ell} ((1-x_i)(1-b_i) + x_i b_i) \cdot V(\vec{b}),$$

where b_i is the i-th bit of $\vec{b}$.

Identity function. Let $\beta : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ be the identity function such that $\beta(\vec{x},\vec{y}) = 1$ if $\vec{x} = \vec{y}$, and $\beta(\vec{x},\vec{y}) = 0$ otherwise. Suppose $\tilde{\beta}$ is the multilinear extension of β. Then $\tilde{\beta}$ can be expressed as:

$$\tilde{\beta}(\vec{x},\vec{y}) = \prod_{i=1}^{\ell} ((1-x_i)(1-y_i) + x_i y_i) .$$

Schwartz-Zippel Lemma. Let $\mathbb{F}$ be a finite field, and let $g : \mathbb{F}^m \to \mathbb{F}$ be a nonzero m-variate polynomial of total degree at most d. Then on any finite set $S \subset \mathbb{F}$,

$$\Pr_{x\to S^m}[g(x) = 0] \le d/|S|.$$

Here, $x \leftarrow S^m$ denotes an x drawn uniformly at random from the product set S^m, and $|S|$ denotes the size of S. In other words, if x is chosen uniformly at random from S^m, then the probability that $g(x) = 0$ is at most $d/|S|$. In particular, any two distinct polynomials of total degree at most d can agree on at most a $d/|S|$ fraction of points in S^m.

4 Fully Batchable Transparent Polynomial Commitment Scheme

In this section, we first provide the formal definition of the fully batchable polynomial commitment (FB-PC), then present a batch proving protocol for multi-output circuits, followed by a transparent construction of FB-PC based on this protocol, and prove its security.

[2] "Regular" circuits is defined in [CMT12].

4.1 Definition of FB-PC

We formalize the definitions of two standard features for polynomial commitments as follows:

Definition 4.1 (Batch Opening). A polynomial commitment scheme supporting batch opening additionally provides efficient algorithms $(\mathsf{BatchOpen}, \mathsf{BatchVerify})$ which are defined as follows:

- $(\{y_i\}_{i \in I}, \pi_I) \leftarrow \mathsf{BatchOpen}(f, I)$: on input the polynomial f and the point set I, outputs the evaluation y_i for every point i in set I and the batch proof π_I. For efficiency, it is required that $|\pi_I| < |I| \cdot |\pi_i|$.
- $\{0, 1\} \leftarrow \mathsf{BatchVerify}(c, I, \{y_i\}_{i \in I}, \pi_I)$: on input the commitment c, point set I, evaluation set $\{y_i\}_{i \in I}$ and the batch proof π_I, verifies the correctness of $\{y_i\}_{i \in I}$.

A polynomial commitment scheme with batch opening satisfies the following property:

- Batch opening binding. For all PPT adversaries $\mathcal{A}$, the following probability is $\mathsf{negl}(\lambda)$[3]:

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}\left(1^\lambda, n\right), \\ (c, \langle I, y_I, \pi_I \rangle, \langle J, y'_J, \pi'_J \rangle) \leftarrow \mathcal{A}(\mathsf{pp}) : \\ \mathsf{BatchVerify}\left(c, I, y_I, \pi_I\right) = 1 \wedge \\ \mathsf{BatchVerify}\left(c, J, y'_J, \pi'_J\right) = 1 \wedge \\ \exists z \in I \cap J \text{ s.t. } y_z \neq y'_z \end{array}\right].$$

Definition 4.2 (Batch Evaluation). A polynomial commitment scheme with batch evaluation additionally provides efficient algorithm $\mathsf{BatchEval}$, which is defined as follows:

- $\{y_i, \pi_i\}_{i \in I} \leftarrow \mathsf{BatchEval}(f, I)$: on input the polynomial f and the point set I, outputs the evaluation and the proof pairs (y_i, π_i) for every $i \in I$ [4]. For efficiency, the condition $p(\lambda, d, I) < |I| q(\lambda, d)$ must be satisfied, where $p(\lambda, d, I)$ represents the running time of algorithm $\mathsf{BatchEval}(f, I)$, and $q(\lambda, d)$ is the running time of $\mathsf{Eval}(f, i)$.

A polynomial commitment scheme is referred as *fully batchable polynomial commitment scheme* if it supports batch opening and batch evaluation simultaneously.

4.2 Construction of Transparent FB-PC

Batch Proving Protocol for Multi-Output Circuits. We propose a batch proving protocol for multi-output circuits, where the verifier receives multiple outputs out of the entire output of a circuit C. Suppose the input of circuit C has size m, and the output has size n. The prover holds the input $[\mathsf{in}]$ and computes the output $[C(\mathsf{in})]$. The verifier queries the i-th output of C and receives $[\mathsf{out}]_i$, which is claimed to be equal to $[C(\mathsf{in})]_i$. The i-th output $[C(\mathsf{in})]_i$ is equivalently written as $V_0(\vec{i})$, where $V_0(\vec{x})$ denotes the entire output of C for $\vec{x} \in \{0,1\}^{\log n}$ and $\vec{i}$ is the binary representation of gate i, so $V_0(\vec{i})$ corresponds to the i-th value in the output layer. Therefore, proving the correctness of $[\mathsf{out}]_i$ is equivalent to proving the following statement:

$$[\mathsf{out}]_i = \sum_{\vec{x} \in \{0,1\}^{\log n}} \tilde{\beta}(\vec{i}, \vec{x}) \tilde{V}_0(\vec{x}),$$

where $\tilde{\beta}(\vec{i}, \vec{x})$ is the identity function to select the i-th output of the circuit.

For multiple outputs $\{[\mathsf{out}]_i\}_{i \in I}$, where $I \subset [n]$ is the set of the output indeces, the prover wishes to convince the verifier of the validity of multiple statements:

$$\{[\mathsf{out}]_i = \sum_{\vec{x} \in \{0,1\}^{\log n}} \tilde{\beta}(\vec{i}, \vec{x}) \tilde{V}_0(\vec{x})\}_{i \in I}.$$

[3]In our definition, the $\mathsf{BatchOpen}$ algorithm outputs the evaluations directly rather than the interpolation polynomial of the opening points and evaluations as in Kate's definition [KZG10]. This adjustment makes our definition more general, suiting not only KZG polynomial scheme but also others that are not homomorphic.

[4]The Eval algorithm can be used to verify each pair.

A naive approach is that the prover and verifier independently prove each statement $|I|$ times, resulting in $O(|I||C|)$ prover time, and $O(|I|d\log|C|)$ for both proof size and verification time, assuming the size of the circuit is $|C|$ and the depth is d.

To reduce the proof size, we propose a batch selective function $\sum_{i\in I} r^k\tilde{\beta}(\vec{i},\vec{x})$ that aggregates multiple outputs into a batch statement:

$$\sum_{i\in I} r^k[\mathsf{out}]_i = \sum_{\vec{x}\in\{0,1\}^{\log n}}\sum_{i\in I} r^k\tilde{\beta}(\vec{i},\vec{x})\tilde{V}_0(\vec{x}),$$

where k is the position of i in set I. Therefore, proving $\{[\mathsf{out}_i]\}_{i\in I}$ are correct is equivalent to proving the above batch statement. Initially, $\mathcal{P}$ sends $\{[\mathsf{out}]_i\}_{i\in I}$ to $\mathcal{V}$, who responds by sending a random $r \xleftarrow{r} \mathbb{F}$ to $\mathcal{P}$ and computes $\{r^k\}_{k\in[|I|]}$ to get the batch statement. They then run the sumcheck protocol, which reduces the claim about $\sum_{i\in I} r^k[\mathsf{out}]_i$ to the claim about $\tilde{V}_0(\vec{g}^{(0)})$, where $\vec{g}^{(0)}$ is the random challenges produced in the sumcheck protocol. Next, to prove the correctness of $\tilde{V}_0(\vec{g}^{(0)})$, they run the GKR protocol [GKR08] to further reduce the claim about $\tilde{V}_0(\vec{g}^{(0)})$ to the claim about $\tilde{V}_d(\vec{g}^{(d)})$ in the input layer, where $\vec{g}^{(d)}$ are the random challenges produced during the GKR protocol. Unlike the final round of the GKR protocol, the input to the circuit is not known to the verifier here, and thus the verifier cannot check $\tilde{V}_d(\vec{g}^{(d)})$ locally. Thus, the verifier and the prover invoke the multivariate polynomial commitment [ZXZS20] to validate that $\tilde{V}_d(\vec{g}^{(d)})$ is an evaluation of $V_d(\vec{x})$ at $\vec{g}^{(0)} \in \mathbb{F}^{\log n}$.

Theorem 4.1. *Protocol 1 is an argument system for the statements* $\{[\mathsf{out}]_i\}_{i\in I} = \{[C(\mathsf{in})]_i\}_{i\in I}$ *such that* $[\mathsf{out}] = [C(\mathsf{in})]$ *with soundness error* $O(\max\{|I|/|\mathbb{F}|, d\log|C|/|\mathbb{F}|\})$. *The prover time is* $O(|C| + m\log m)$, *the verifier time and the proof size are both* $O(d\log|C| + \log^2 m)$.

Proof. **Completeness**. The completeness is straightforward. By the definition of $\tilde{V}_0(\vec{x})$ and $\tilde{\beta}(\vec{i},\vec{x})$, if $[\mathsf{out}]_i = \tilde{V}_0(\vec{i})$ for every $i\in I$, then it follows that $\tilde{V}_0(\vec{i}) = \sum_{\vec{x}\in\{0,1\}^{\log n}}\tilde{\beta}(\vec{i},\vec{x})\tilde{V}_0(\vec{x})$. Therefore, the random linear combination of $\{[\mathsf{out}]_i\}_{i\in I}$ is equal to the random linear combination of $\{\sum_{\vec{x}\in\{0,1\}^{\log n}}\tilde{\beta}(\vec{i},\vec{x})\tilde{V}_0(\vec{x})\}_{i\in I}$.

Soundness. If $\{[\mathsf{out}]_i\}_{i\in I} \neq \{[C(\mathsf{in})]_i\}_{i\in I}$, let $\tilde{V}_0'(\vec{g}^{(0)})$ denote the correct value corresponding to C in step 5. If $\tilde{V}_0(\vec{g}^{(0)}) \neq \tilde{V}_0'(\vec{g}^{(0)})$, then $\mathcal{V}$ outputs 1 in Step 7 with the probability at most $O(d\log|C|/|\mathbb{F}|)$ by the soundness error of the GKR protocol. If $\tilde{V}_0(\vec{g}^{(0)}) = \tilde{V}_0'(\vec{g}^{(0)})$, the verifier passes Step 5 with the probability of $\mathsf{max}\{O(\log n/|\mathbb{F}|), O(|I|/|\mathbb{F}|)\}$ by the union bound for the soundness error of the sumcheck protocol and the Schwartz-Zippel Lemma. Thus, the total probability is bounded by $\mathsf{max}\{O(|I|/|\mathbb{F}|), O(d\log|C|/|\mathbb{F}|)\}$ by the union bound.

Efficiency. The proof size and the verification time in Step 5 are $O(\log n)$ while the proof size and the verification time in Step 6 are $O(d\log|C|)$. If Protocol 1 employs Virgo [ZXZS20] as the transparent MVPC scheme, the proof size and the verification time are $O(\log^2 m)$ in Step 7. Thus, the overall verification time and proof size are $O(d\log|C| + \log^2 m)$. The prover runs in $O(n\log n)$ in step 3 to compute n evaluations by using the FFT algorithm and sends $|I|$ evaluations to verifier. The prover runs in $O(n)$ time in Step 5 and $O(|C|)$ time to invoke the GKR protocol on C in Step 6 by Theorem 1. The prover time is $O(m\log m)$ for the MVPC scheme Virgo [ZXZS20] in step 7. Therefore, the total prover time is $O(|C| + m\log m)$ asymptotically. □

Protocol 1 can be converted into a non-interactive protocol that is secure in the random oracle model by using the Fiat-Shamir transformation [FS86, BR93]. It is known that if a public-coin interactive proof for a language satisfies a property called round-by-round soundness, then the non-interactive proof is sound in the random oracle model [CCH+19, BCS16]. Canetti et al. [CCH+19] showed that the GKR protocol and any other interactive proofs based on the sumcheck protocol satisfy round-by-round soundness, and hence applying the Fiat-Shamir transformation to Protocol 1 yields a non-interactive protocol that is secure in the random oracle model.

A Transparent FB-PC. We present a transparent fully batchable polynomial commitment scheme, referred to as TFB-PC, which achieves $O(n\log n)$ prover time, $O(n)$ verifier time and $O(\log^2 n)$ proof size for n evaluations. Compared to the $\mathrm{ZXH}^+_{\mathrm{trans}}$ scheme [ZXH+22], both the prover time and verifier time are optimal, and the proof size is reduced by a factor of $O(n)$.

We achieve this by introducing the batch opening technique for $\mathrm{ZXH}^+_{\mathrm{trans}}$ scheme. The core components are two algorithms $\mathsf{BatchOpen}$ and $\mathsf{BatchVerify}$, as detailed in Scheme 1. These algorithms are

Protocol 1. Batch proving protocol for multi-output circuits.
Let λ be the security parameter. Let $C : \mathbb{F}^m \rightarrow \mathbb{F}^n$ be a d-depth layered arithmetic circuit. The input of circuit C is $[\mathsf{in}]$ with size m, and the output is $[\mathsf{out}]$ with size n. For any $I \subset [n]$, the prover $\mathcal{P}$ needs to convince the verifier $\mathcal{V}$ that $\{[\mathsf{out}]_i\}_{i\in I} = \{[C(\mathsf{in})]_i\}_{i\in I}$, where $[\cdot]_i$ denotes the i-th value in set $[\cdot]$. Without loss of generality, assume m and n are both powers of 2, padding them if not.

1. Set $\mathsf{pp} \leftarrow \mathsf{MVPC.Setup}(1^\lambda)$.
2. $\mathcal{P}$ computes the multilinear extension for the input of C as $\tilde{V}_d(\vec{x})$, where $x \in \mathbb{F}^{\log m}$. Then it invokes $\mathsf{MVPC.Commit}(\tilde{V}_d(\vec{x}), \mathsf{pp})$ to generate $\mathsf{com}_{\tilde{V}_d}$ and sends $\mathsf{com}_{\tilde{V}_d}$ to $\mathcal{V}$.
3. $\mathcal{P}$ sends $\{[\mathsf{out}]_i\}_{i\in I}$ to $\mathcal{V}$.
4. $\mathcal{V}$ randomly picks $r \xleftarrow{r} \mathbb{F}$ and sends it to $\mathcal{P}$.
5. For $I \subset [n]$ and every $i \in I$, $\mathcal{P}$ and $\mathcal{V}$ run a sumcheck protocol on
$$\sum_{i\in I} r^k [\mathsf{out}]_i = \sum_{\vec{x}\in\{0,1\}^{\log n}} \sum_{i\in I} r^k \tilde{\beta}(\vec{i}, \vec{x}) \tilde{V}_0(\vec{x}),$$
where $\vec{i}$ is the binary string of i, and k is the position of i in set I. At the end of the protocol, $\mathcal{V}$ computes $\sum_{i\in I} r^k \tilde{\beta}(\vec{i}, \vec{g}^{(0)})$ locally and receives $\tilde{V}_0(\vec{g}^{(0)})$ sent by $\mathcal{P}$ to check the last statement of the sumcheck protocol, where $\vec{g}^{(0)}$ is the random challenges generated in the sumcheck protocol.
6. To prove the correctness of $\tilde{V}_0(\vec{g}^{(0)})$, $\mathcal{P}$ invokes the GKR protocol on the circuit C to generate the proof untill to the last round of the GKR protocol.
7. In the last round of the GKR protocol, $\mathcal{P}$ and $\mathcal{V}$ have the claim about $\tilde{V}_d(\vec{g}^{(d)})$. $\mathcal{P}$ invokes $\mathsf{MVPC.Open}$ to compute the evaluation of $\tilde{V}_d(\vec{x})$ at point $\vec{g}^{(d)}$ and the corresponding proof. $\mathcal{V}$ invokes $\mathsf{MVPC.Verify}$ to verify according $\mathrm{com}_{\tilde{V}_d}$. If it is equal to $\tilde{V}_d(\vec{g}^{(d)})$, $\mathcal{V}$ outputs 1, otherwise, outputs 0.

Scheme 1. Batch opening for $\mathbf{ZXH^{+}_{trans}}$ scheme.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, d)$: invokes $\mathsf{pp}_{\mathrm{MVPC}} \leftarrow \mathsf{MVPC.Setup}(1^\lambda)$ and outputs $\mathsf{pp} = \mathsf{pp}_{\mathrm{MVPC}}$.
- $c \leftarrow \mathsf{Commit}(f)$: on input a polynomial $f(x) = \sum_{i=0}^{d} a_i x^i \in \mathbb{F}^d[X]$, the algorithm computes the multilinear extension of $\{a_i\}_{i=0}^{d}$ as $\tilde{V}_0$, runs $com_{\tilde{V}_d} \leftarrow \mathsf{MVPC.Commit}(\tilde{V}_0, \mathsf{MVPC.pp})$ and outputs $c = com_{\tilde{V}_d}$.
- $\mathsf{BatchOpen}(f, I)$: on input polynomial $f(x)$ and the point set I,
  1. computes $\tilde{V}_0(\vec{i})$ for every $i \in I$.
  2. computes $r = H(c, I, \{\tilde{V}_0(\vec{i})\}_{i \in I})$.
  3. computes the non-interactive proof for sumcheck protocol on
  $$\sum_{i \in I} r^k \tilde{V}_0(\vec{i}) = \sum_{\vec{x} \in \{0,1\}^{\log n}} \sum_{i \in I} r^k \tilde{\beta}(\vec{i}, \vec{x}) \tilde{V}_0(\vec{x})$$
  and attaches the non-interactive sumcheck proof in the batch proof π_I. The last part of the sumcheck proof is $\tilde{V}_0(\vec{g}^{(0)})$ for the common random vector of $\vec{g}^{(0)}$.
  4. computes the non-interactive GKR proof on the circuit C for $\tilde{V}_0(\vec{g}^{(0)})$ and attaches it in the batch proof π_I. The last part of the GKR proof is the claim about $\tilde{V}_d(\vec{g}^{(d)})$.
  5. invokes $(\tilde{V}_d(\vec{g}^{(d)}), \pi_{\mathrm{MVPC}}) \leftarrow \mathsf{MVPC.Open}(\tilde{V}_d, \vec{g}^{(d)}, \mathsf{MVPC.pp})$ to generate the proof for the last part of the GKR proof and attaches π_{MVPC} in the batch proof π_I.
  6. outputs the evaluations $\{V_0(\vec{i})\}_{i \in I}$ and final proof π_I.
- $\mathsf{BatchVerify}(c, I, \{V_0(\vec{i})\}_{i \in I}, \pi_I)$: on input the points I, evaluations $\{V_0(\vec{i})\}_{i \in I}$ and the batch proof π_I, parse the proof π as three parts (the sumcheck proof, the GKR proof and the MVPC proof) and checks:
  1. the sumcheck proof with random challenges $\vec{g}^{(0)}$ provided by $\mathcal{P}$.
  2. the GKR proof with random challenges $\vec{g}^{(d)}$ provided by $\mathcal{P}$.
  3. the MVPC proof using $\mathsf{MVPC.Verify}(\mathsf{com}_{\tilde{V}_d}, \vec{g}^{(d)}, \tilde{V}_d(\vec{g}^{(d)}), \pi_{\mathrm{MVPC}})$.
  4. the generation process of all random challenges using the hash function H.

  If all the above checks pass, outputs 1; else, outputs 0.

derived by instantiating the circuit C in non-interactive Protocol 1 with the FFT circuit [Wei69]. The circuit takes the coefficients of the polynomial f and the n-th root of unity ω as input, and outputs evaluations $f(\omega_0), ..., f(\omega_{n-1})$.

Table 3: The ratio of batch opening for polynomial commitments.

Scheme	**Batch Ratio**		
	$\mathcal{P}$ time	**$\mathcal{V}$ time**	**proof size**
KZG [KZG10]	$O(\log^2 n/n)$	$O(\log^2 n)$	$O(1/n)$
AMT [TCZ+20]	$O(\log^2 n)$	$O(\log n)$	$O(1/n \log n)$
$\mathrm{ZXH}^{+}_{\mathrm{KZG}}$ [ZXH+22]	$O(\log n)$	$O(\log^2 n)$	$O(1/n)$
hbACSS [YLF+22]	$O(1)$	$O(1)$	$O(1)$
$\mathrm{ZXH}^{+}_{\mathrm{trans}}$ [ZXH+22]	$O(1)$	$O(1)$	$O(1)$
Our scheme	$O(1)$	$O(1/\log^2 n)$	$O(1/n)$

Batch ratio includes the ratio of the $\mathcal{P}$ time, $\mathcal{V}$ time, and proof size for batch opening versus naive opening. The ratio of less than 1 indicates an improvement in efficiency compared to the counterpart naive opening.

Table 3 compares our batch opening with prior schemes. The batch ratio includes the ratio of prover time, verifier time, and proof size for batch opening versus standard opening, i.e., the time taken by $\mathsf{BatchOpen}$ and $\mathsf{BatchVerify}$ compared to running n separate Eval and Verify operations. A smaller ratio indicates a more efficient batch opening, with values less than 1 indicating improvements over the naive approach. Compared to other works, our batch opening achieves a more significant efficiency improvement. The ratio of our $\mathcal{P}$ time, $\mathcal{V}$ time and proof size is $O(1)$, $O(1/\log^2 n)$ and $O(1/n)$ respectively, demonstrating that we reduce the verifier time and proof size of $\text{ZXH}^+_{\text{trans}}$ by $O(\log^2 n)$ and $O(n)$ respectively without sacrificing the prover time. Following $\text{ZXH}^+_{\text{trans}}$, we assume the degree of polynomial is $t = \Theta(n)$.

Since $\text{ZXH}^+_{\text{trans}}$ scheme already offers a favorable prover time of $O(n \log n)$. Building upon this, batch opening in Scheme 1 reduces the verifier time to $O(n)$ and the proof size to $O(\log^2 n)$ while maintaining the prover time at $O(n \log n)$. Therefore, our transparent FB-PC, supporting batch evaluation and batch opening, achieves $O(n \log n)$ prover time, $O(n)$ verifier time and $O(n \log^2 n)$ proof size.

Theorem 4.2. *If the non-interactive Protocol 1 for $\mathcal{P}$ and $\mathcal{V}$ is sound, then Scheme 1 is a polynomial commitment scheme supporting batch opening.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that breaks the batch opening binding with non-negligible probability ϵ, we show how to use $\mathcal{A}$ to build an adversary $\mathcal{B}$ that breaks the security of non-interactive Protocol 1.

Suppose $\mathcal{A}$ outputs $(c, \langle I, y_I, \pi_I \rangle, \langle I', y'_{I'}, \pi'_{I'} \rangle)$ such that $\exists i \in I \cap I' \wedge y_i \neq y'_i$ and $\mathsf{BatchVerify}(c, I, y_I, \pi_I) \wedge \mathsf{BatchVerify}(c, I', y'_I, \pi'_I) = 1$. Since $\exists i \in I \cap I' \wedge y_i \neq y'_i$, there must be at least one incorrect evaluation $y \in \{y_i, y'_i\}$ that claims to equal $f(i)$.

$\mathcal{B}$ chooses a tuple from (c, I, y_I, π_I) and (c, I', y'_I, π'_I), then designates it as (c, J, y_J, π). The probability that the incorrect evaluation y is in y_J is greater than $1/2$. $\mathcal{B}$ compute the random value $r = H(c, J, y_J)$ and set the statement as $\sum_{j \in J} r^k y_j = \sum_{x \in \{0,1\}^{\log n}} \sum_{j \in J} r^k \tilde{\beta}(\vec{j}, \vec{x}) \tilde{V}_0(\vec{x})$, where k is the position of j in set J. Then $\mathcal{B}$ outputs π as the proof of the statement. The probability that $\mathcal{B}$ breaks the security of non-interactive Protocol 1 is greater than $\epsilon/2$, which contradicts the security of Protocol 1. □

5 Fully Optimal VSS

We first introduce a generic VSS construction based on FB-PC. Then, by integrating our transparent FB-PC scheme, we achieve a fully optimal VSS scheme with $O(n \log n)$ dealer time, $O(n)$ participant time, and $O(n)$ communication cost.

Generic VSS Construction from FB-PC. In contrast to the generic VSS construction described by Zhang et al. [ZXH+22], we replace the single evaluation algorithm Eval in the dealing round with the batch evaluation algorithm $\mathsf{BatchEval}$. Similarly, in the complaint round, we replace Eval and $\mathsf{VerifyEval}$ with the batch opening algorithms $\mathsf{BatchOpen}$ and $\mathsf{BatchVerify}$, respectively. Details are provided in Scheme 1. As for the security, previous work has shown that if the underlying polynomial commitment scheme is secure, the VSS scheme is secure [KZG10, TCZ+20, ZXH+22].

A Transparent Fully Optimal VSS Scheme. We instantiate FB-PC in our generic VSS construction with TFB-PC, obtaining a transparent VSS scheme, referred to as TFB-PC-VSS, which achieves optimal time complexities. The dealer time is $O(n \log n)$ and participant time is $O(n)$, both of which are optimal, since simply computing n shares would incur $O(n \log n)$ time using the FFT algorithm and reading n shares would incur $O(n)$ time. Our broadcast proof in the complaint round is $O(\log^2 n)$ and the overall communication is $O(n)$, which is also optimal since there are $O(n)$ shares to be broadcast in the complaint round. Our scheme reduces the participant time and communication cost of $\text{ZXH}^+_{\text{trans}}$-VSS by $O(\log^2 n)$ (see table 2).

6 Implementation and Evaluation

We present the details of our implementation and report experimental results. Additionally, we compare our approach with the $\text{ZXH}^+_{\text{trans}}$ schemes [ZXH+22] and discuss their performance.

Scheme 2. An (n, t)-VSS scheme from our FB-PC.

Sharing phase

Dealing round:

1. $\mathcal{P}$ picks $f \in_R \mathbb{F}[X]$ of degree t such that $s = f(0)$, computes $s_i = f(u_i)$ for all $i \in [n]$.
2. $\mathcal{P}$ runs $c \leftarrow \mathsf{Commit}(f, \mathsf{pp})$ and broadcasts c to all verifiers.
3. $\mathcal{P}$ runs $\{y_i, \pi_i\}_{i\in[n]} \leftarrow \mathsf{BatchEval}\,(f, [n])$ and sends (y_i, π_i) to $\mathcal{V}_i$ for all $i \in [n]$ over an authenticated, private channel.

Complaint round:

1. For all $i \in [n]$, $\mathcal{V}_i$ invokes $b \leftarrow \mathsf{VerifyEval}(c, u_i, y_i, \pi_i)$. If $b = 0$, $\mathcal{V}_i$ broadcasts a complaint to accuse the dealer.
2. If the size of the set S of complaining players is larger than t, the dealer is disqualified. Otherwise, the dealer invokes $(\{y_i\}_{i\in S}, \pi_S) \leftarrow \mathsf{BatchOpen}(f, S)$ and reveals the correct shares with proofs by broadcasting $(\{y_i\}_{i\in S}, \pi_S)$.
3. Every participant invokes $\{0, 1\} \leftarrow \mathsf{BatchVerify}\,(c, S, \{y_i\}_{i\in S}, \pi_S)$. If the batching proof does not pass (or dealer did not broadcast), the dealer is disqualified. Otherwise, each $\mathcal{V}_i$ now has the correct share $f(u_i)$.

Reconstruction phase

Given c and shares $\{i, y_i, \pi_i\}_{i\in T}$ such that $|T| > t$, the reconstructor:

1. runs $b_i \leftarrow \mathsf{VerifyEval}(c, u_i, y_i, \pi_i)$ if y_i has not been broadcast in the complaint round, else set $b_i = 1$.
2. recovers f with $\{i, y_i\}_{i\in T}$ by Lagrange interpolation and obtains $s = f(0)$ if $b_i = 1$ for all $i \in T$,

6.1 Implementation Details

Our schemes are implemented in C++ based on the open-source codebase of the $\text{ZXH}^{+}_{\text{trans}}$ scheme, available at `https://github.com/real-world-cryprography`. The implementation includes the following key enhancements:

- **Secure Fiat-Shamir Transformation.** The Fiat-Shamir transformation is widely used in cryptographic protocols to convert interactive proofs into non-interactive ones. Recent research highlights that improper application of the Fiat-Shamir transformation can lead to adaptive attacks [HLPT20, DMWG23]. The Fiat-Shamir transformation implemented by Zhang et al. [ZXH+22] is a weaker version, as it neglects certain public information, such as the circuit description and claimed values, which exposes the protocol to potential adaptive attacks. To address this vulnerability, we implement a more robust version of the Fiat-Shamir transformation in our scheme. However, this enhancement comes at the cost of increased memory usage and slower computation.
- **Hash Chaining.** To optimize memory usage in the Fiat-Shamir transformation, we employ hash chaining [Tha22]. Specifically, for the i-th round challenge r_i, instead hashing all preceding public messages, we compute it via hashing only the $(i-1)$-th round message and challenge r_{i-1}. This technique reduces memory requirements by approximately 3%, as it allows us to store only the latest round messages rather than the entire transcript.

6.2 Experimental Setup

We run the experiments on an AMD ecs.c6a.16xlarge instance with an AMD EPYC$^{\text{TM}}$ ROME 7H12, 64 vCPU, and 128 GiB of RAM. In all experiments, we set the degree of the polynomial as $t = n/2$, and the number of parties n ranges from 2^{11} to 2^{21}.

While the stronger Fiat-Shamir transformation improves security, it increases computational cost and memory usage, which limits the size of the instances that can be processed efficiently. On an AMD ecs.ebmr6a.64xlarge instance with an AMD ROME 7H12 CPU (256vCPU and 2048 GiB RAM), we can run instances only up to 2^{15} with the strong Fiat-Shamir transformation. Therefore, to ensure broader evaluation, we compare the performance of our work with Zhang et al. [ZXH+22] using their weaker Fiat-Shamir implementation. This allows us to handle instances up to 2^{21} on a more resource-constrained setup, the AMD ecs.c6a.16xlarge instance.

6.3 Performance of TFB-PC

We compared TFB-PC with the $\text{ZXH}^{+}_{\text{trans}}$ scheme. Our experimental results, as depicted in Fig. 4, clearly demonstrate the efficiency of our batch opening.

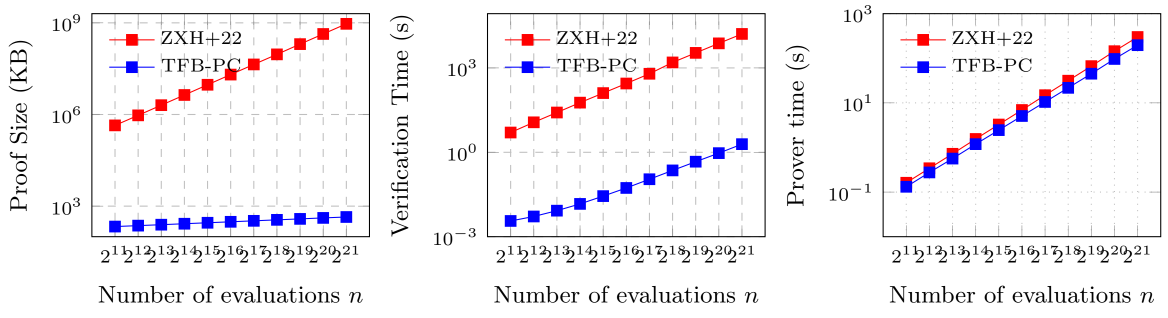

Figure 4: Comparison of transparent polynomial commitments

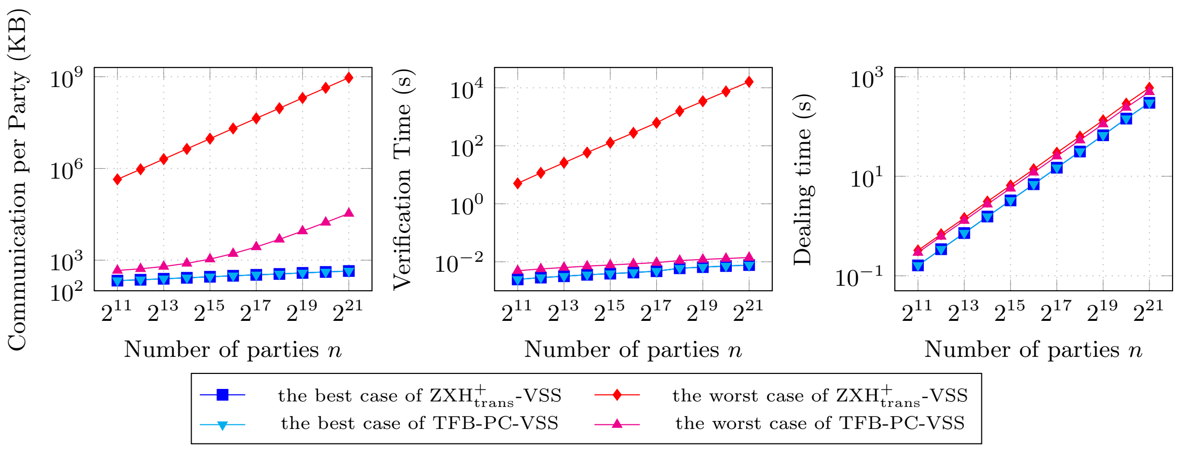

Figure 5: VSS comparison

TFB-PC significantly reduces the proof size for n evaluations compared to $\text{ZXH}^{+}_{\text{trans}}$ scheme. The proof size of TFB-PC is 214.44KB for 2^{11} evaluations and 440.77KB for 2^{21} evaluations, which is $2,168$-$2,305,164\times$ smaller than of $\text{ZXH}^{+}_{\text{trans}}$ scheme, where the proof sizes are 430MB and 885.57GB, respectively. The data clearly shows that the proof size of TFB-PC is smaller by an entire order of magnitude.

The verification of TFB-PC is $2,061$-$2,571,456\times$ faster than $\text{ZXH}^{+}_{\text{trans}}$ scheme. For 2^{11} evaluations, our scheme takes just 2.42ms, while $\text{ZXH}^{+}_{\text{trans}}$ takes 5.13s. The disparity grows with larger batch sizes, and for 2^{21} evaluations, our scheme requires only 6.23ms, whereas $\text{ZXH}^{+}_{\text{trans}}$ takes a prohibitive 4.09

hours. The prover time of $\text{ZXH}^+_{\text{trans}}$ is comparable to ours, but we benefit from eliminating the need to compute the common random challenge, which further enhances its practical applicability.

6.4 Performance of TFB-PC-VSS

We evaluated TFB-PC-VSS against $\text{ZXH}^+_{\text{trans}}$-VSS [ZXH+22], particularly focusing on scenarios involving a complaint round where the dealer broadcasts $O(n)$ shares in the worst case. In the best case, TFB-PC-VSS performs comparably to $\text{ZXH}^+_{\text{trans}}$-VSS since it has no complaints and does not require batch opening.

TFB-PC-VSS demonstrates a reduction in communication cost by up to 954-$27,595\times$ and participant time by up to $1,028$-$1,155,106\times$ compared to $\text{ZXH}^+_{\text{trans}}$-VSS, for 2^{11}-2^{21} parties. As shown in Fig. 4, our communication per participant is only 461.27KB for 2^{11} evaluations and 32.86MB for 2^{21} evaluations. In contrast, $\text{ZXH}^+_{\text{trans}}$-VSS requires a significantly larger communication cost of 429.87MB and 885.6GB for the same evaluations, respectively. Furthermore, the verification time in our scheme is highly efficient, taking only 4.91ms for 2^{11} evaluations and 13.99ms for 2^{21} evaluations. By comparison, $\text{ZXH}^+_{\text{trans}}$-VSS takes 5.05s and a prohibitive 4.49h for the same tasks. The dealer time of our scheme is similar to that of $\text{ZXH}^+_{\text{trans}}$-VSS in Fig. 5 because we have the same prover time complexity.

TFB-PC-VSS has a stable performance in the worst and best case compared to $\text{ZXH}^+_{\text{trans}}$-VSS. In the best case, the performance of TFB-PC-VSS and $\text{ZXH}^+_{\text{trans}}$-VSS are the same, with communication per participant is 214.83KB for 2^{11} evaluations and 442.8KB for 2^{21} evaluations, with participant times of 2.5ms and 7.7ms, respectively. In the worst case, where complaints occur, the broadcast proof of TFB-PC-VSS grows modestly from 214.44KB for 2^{11} parties to 440.77KB for 2^{21} evaluations, while that of $\text{ZXH}^+_{\text{trans}}$-VSS rises sharply from 430MB to 885.57GB. Moreover, the verification time of TFB-PC-VSS remains remarkably consistent, ranging from 2.45ms for 2^{11} parties to 6.28ms for 2^{21} parties, while that of $\text{ZXH}^+_{\text{trans}}$-VSS ranges from 5.05s for 2^{11} parties to 4.49h for 2^{21} parties. This stark contrast highlights the efficiency and scalability of our approach, especially in large-scale applications.

7 Conclusion

In this paper, we propose a transparent VSS scheme with optimal dealer time, participant time, and communication cost, suitable for large-scale privacy-preserving distributed computation. The core is the first transparent fully batchable polynomial commitment scheme, TFB-PC, which supports both batch opening and batch evaluation. Compared to the $\text{ZXH}^+_{\text{trans}}$ scheme [ZXH+22], TFB-PC reduces the proof size by $O(n)$ and verifier time by $O(\log^2 n)$, without compromising the prover's efficiency. Leveraging FB-PC, we present a generic VSS construction and achieve an optimal VSS scheme, TFB-PC-VSS. We implement TFB-PC-VSS and evaluate its performance against the $\text{ZXH}^+_{\text{trans}}$-VSS [ZXH+22] in the same setting. The experimental results show that, TFB-PC-VSS has a comparable dealer time with $\text{ZXH}^+_{\text{trans}}$-VSS, while reducing its communication cost by 954-$27,595\times$, and improving the participant time by $1,028$-$1,155,106\times$ for 2^{11} to 2^{21} parties.

References

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography - 14th International Conference, TCC 2016-B*, pages 31–60. Springer, 2016.

[BDFG20] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. *IACR Cryptol. ePrint Arch.*, page 81, 2020.

[BDFG21] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In *Advances in Cryptology - CRYPTO 2021*, pages 649–680. Springer, 2021.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93*, pages 62–73. ACM, 1993.

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-shamir: From practice to theory. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 1082–1090. ACM, 2019.

[CFS17] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. A zero knowledge sumcheck and its applications. *CoRR*, 2017.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th Annual Symposium on Foundations of Computer Science*, pages 383–395. IEEE Computer Society, 1985.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 90–112. ACM, 2012.

[DMWG23] Quang Dao, Jim Miller, Opal Wright, and Paul Grubbs. Weak fiat-shamir attacks on modern proof systems. In *44th IEEE Symposium on Security and Privacy, SP 2023*, pages 199–216. IEEE, 2023.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science*, pages 427–437. IEEE Computer Society, 1987.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86*, pages 186–194. Springer, 1986.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, 2008*, pages 113–122. ACM, 2008.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, page 953, 2019.

[HLPT20] Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. How not to prove your election outcome. In *2020 IEEE Symposium on Security and Privacy, SP 2020*, pages 644–660. IEEE, 2020.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology - ASIACRYPT 2010*, pages 177–194. Springer, 2010.

[LFKN90] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *31st Annual Symposium on Foundations of Computer Science, 1990*, pages 2–10. IEEE Computer Society, 1990.

[TCZ$^+$20] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan-Gueta, and Srinivas Devadas. Towards scalable threshold cryptosystems. In *2020 IEEE Symposium on Security and Privacy, SP 2020*, pages 877–893. IEEE, 2020.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Advances in Cryptology - CRYPTO 2013*, pages 71–89. Springer, 2013.

[Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. *Found. Trends Priv. Secur.*, pages 117–660, 2022.

[Wei69] Clifford J. Weinstein. Quantization effects in digital filters. 1969.

[XZZ$^+$19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Advances in Cryptology - CRYPTO 2019*, pages 733–764. Springer, 2019.

[YLF$^+$22] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew Miller. hbacss: How to robustly share many secrets. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022*. The Internet Society, 2022.

[ZXH$^+$22] Jiaheng Zhang, Tiancheng Xie, Thang Hoang, Elaine Shi, and Yupeng Zhang. Polynomial commitment with a one-to-many prover and applications. In *31st USENIX Security Symposium, USENIX Security 2022*, pages 2965–2982, 2022.

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy, SP 2020*, pages 859–876. IEEE, 2020.