Polynomial Commitment with a One-to-Many Prover and Applications

Jiaheng Zhang
UC Berkeley

Tiancheng Xie
UC Berkeley

Thang Hoang
Virginia Tech

Elaine Shi
CMU

Yupeng Zhang
Texas A&M University

Abstract

Verifiable Secret Sharing (VSS) is a foundational cryptographic primitive that serves as an essential building block in multi-party computation and decentralized blockchain applications. One of the most practical ways to construct VSS is through a polynomial commitment, where the dealer commits to a random polynomial whose 0-th coefficient encodes the secret to be shared, and proves the evaluation of the committed polynomial at a different point to each of N verifiers, i.e., the polynomial commitment is used in a "one-to-many" fashion.

The recent work of Tomescu et al. (IEEE S&P 2020) was the first to consider polynomial commitment with "one-to-many prover batching", such that the prover can prove evaluations at N different points at the cost of $\widetilde{O}(1)$ proofs. However, their scheme is not optimal and requires a trusted setup.

In this paper, we asymptotically improve polynomial commitment with one-to-many prover batching. We propose two novel schemes. First, we propose a scheme with optimal asymptotics in all dimensions in the trusted setup setting. Second, we are the first to consider one-to-many prover batching for *transparent* polynomial commitments, and we propose a transparent scheme whose performance approximately matches the best-known scheme in the trusted setup setting.

We implement our schemes and evaluate their performance. Our scheme in the trusted setup setting improves the proof size by 20× and the verifier time by 7.8× for 2^{21} parties, with a small overhead on the prover time. Our transparent polynomial commitment removes the trusted setup and further improves the prover time by 2.3×.

1 Introduction

In an $(N, t+1)$ Verifiable Secret Sharing (VSS) protocol [3, 9, 11, 25, 32], roughly speaking, there is a *dealer* and N receivers. The dealer has a secret s, and it wants to split s into N shares, and gives out one share to each receiver. The secret s can be reconstructed if at least $t+1$ receivers combine their shares. However, any coalition of t or fewer receivers cannot learn any information about s (assuming that the dealer is honest). The scheme is "verifiable" if honest receivers can reliably detect a cheating dealer who deals internally inconsistent shares to different receivers. VSS is a foundational building block and widely used in multi-party computation [3, 9, 32], threshold cryptosystems [32, 35], and distributed key generation (DKG) [23, 24, 35]. Recently, VSS has received increasing attention since decentralized blockchains provide a large-scale playground for threshold cryptosystems [20, 35].

Several recent works [25, 35] showed that *round-efficient* VSS can be constructed from *polynomial commitment* schemes — this is one of the most practical approaches for constructing VSS. In a polynomial commitment scheme, a dealer (also called a prover) can produce a commitment c of a polynomial f whose coefficients are assumed to be in some finite field. Later, during an opening phase, the dealer can claim that the committed polynomial evaluates to y at a given point x, and it can prove to a verifier that this is indeed the correct evaluation result by producing an ideally succinct proof π. Given a polynomial commitment scheme, it is relatively straightforward to construct a VSS scheme [25, 35]. Specifically, the dealer chooses a random degree-t polynomial f whose 0-th coefficient encodes the secret s. The dealer now commits to the polynomial and broadcasts the commitment c to all receivers. Next, it chooses N distinct points $x_1, x_2, \ldots, x_N$, and gives $y_i = f(x_i)$ to receiver $i \in [N]$ respectively, and proves to the receiver that the purported outcome y_i is correct with respect to the commitment c. If the dealer is honest, then optimistically the protocol can end here. If the dealer is dishonest and deals incorrect shares to many receivers, the receivers can resort to some complaint mechanism to disqualify the dealer (assuming a synchronous network).

To construct VSS from polynomial commitments, the underlying polynomial commitment scheme is used in a *one-to-many* fashion, i.e., for the same committed polynomial, the dealer needs to prove N evaluations on different points to N different receivers. To produce these N proofs, a naïve approach is for the prover to repeat N times the proving algorithm of the underlying polynomial commitment scheme, thus incurring N times the computational overhead. The recent work of Tomescu et al. [35] showed an elegant *one-to-many prover batching* technique for the well-known KZG polynomial commitment [25], such that the dealer can produce N proofs with only $\widetilde{O}(1)$ slowdown (relative to computing a single proof). Tomescu et al.'s work, however, does not achieve prove batching directly for the KZG scheme, but rather, a more involved variant of KZG. It breaks down the proof generation of the KZG scheme into $\log N$ steps and constructs an authenticated multipoint evaluation tree (AMT) to store redundant computations and improve the efficiency of multiple proofs. Consequently, the verification time and proof size become a logarithmic factor more costly than the original KZG protocol. Another limitation of Tomescu et al.'s work is that it relies on a trusted setup. If the trusted setup is compromised, then the soundness of the scheme can be broken. In decentralized blockchain applications, such a trusted setup is undesirable.

In this paper, we revisit the interesting direction suggested by Tomescu et al. [35]. We ask the following two questions:

1. Can we achieve one-to-many prover batching directly for the KZG polynomial commitment? If so, can we preserve the optimal verification time and proof size of KZG, while computing N proofs for the cost of one (or for the cost of $\widetilde{O}(1)$ proofs)?
2. Can we approximately match the asymptotic overhead of Tomescu et al. [35] in all dimensions, but remove the trusted setup?

1.1 Our Results and Contributions

We answer these questions with two novel constructions of "polynomial commitment with one-to-many prover batching".

- **Prover batching for the KZG polynomial commitment.** The first contribution is a new algorithm for computing N KZG proofs for the same committed polynomial, paying the cost of only $\widetilde{O}(1)$ proofs. Since our algorithm does not modify the underlying KZG polynomial commitment, we inherit the constant verification time and constant proof size of KZG. Our scheme achieves asymptotic optimality in all dimensions: the proof size and verification time are optimal; the prover time for generating N proofs is $O(N \log N)$ which is also optimal, since simply evaluating the polynomial at N different points would incur $N \log N$ time using the Fast Fourier Transformation (FFT), assuming that $t = \Theta(N)$. Therefore, our scheme also subsumes the results of the original Kate et al.'s paper [25] and Tomescu et al.'s paper [35][1].
- **Transparent polynomial commitment with prover batching.** Our second contribution is a transparent polynomial commitment scheme where a dealer can produce N proofs in $O(N \log N)$ time, and the verification time and proof size are both $O(\log^2 N)$. Here, the prover time is optimal for the same reason as mentioned earlier. Both the proof size and the verification time are succinct and only a logarithmic factor worse than Tomescu et al. [35].
- **Implementation and evaluation.** We fully implemented both our schemes and evaluated their performance. We then used our new "polynomial commitment schemes with prover batching" to implement VSS and DKG protocols. We compared the efficiency of the resulting schemes with prior work in the same setting. With $N = 2^{21}$ parties, our KZG-based polynomial commitment and the corresponding VSS scheme reduced the proof size of the AMT scheme [35] by 20×, reduced the verifier time by 7.8×, while introducing a small overhead of 3× on the prover time. These led to 3.3× better computation time and 20× smaller communication in the DKG scheme. Our transparent scheme not only removes the trusted setup but also improves the prover time by an order of magnitude. However, it does introduce a large proof size. Our code is open source (the code is available at `https://github.com/sunblaze-ucb/eVSS`).
- **Techniques: "one-to-many zero-knowledge proof".** To construct our transparent polynomial commitment with prover batching, we come up with a more general technique which can be of independent interest and lead to other interesting applications. Basically, consider a circuit C with N outputs, wherein the prover wants to prove one output to each verifier respectively. We give a "one-to-many zero-knowledge proof" construction where the prover's computation is only $\widetilde{O}(|C|)$ where $|C|$ denotes the size of C, whereas a straightforward application of existing techniques where the prover produces a separate proof for each verifier would have incurred at least $N \cdot |C|$ prover time.

Table 1 shows how our "polynomial commitment with one-to-many prover batching" compares with prior schemes. Given such a polynomial commitment scheme, one can directly construct (synchronous) VSS and DKG using existing techniques described by Tomescu et al. [35]. Table 2 shows the asymptotic overhead of the resulting VSS and DKG schemes and how our work improves over prior work.

1.2 Technical Highlights

1.2.1 Transparent Polynomial Commitment with Prover Batching

A naïve idea is to commit to the coefficients of the polynomial f using a vector commitment, resulting in a concise

[1] After submitting our paper, we found that the same algorithm was also proposed independently by Dankrad Feist and Dmitry Khovratovich at `https://github.com/khovratovich/Kate`. We thank Alin Tomescu for pointing it out.

Table 1: Polynomial commitment with a one-to-many prover: comparison with prior works. We assume $t = \Theta(N)$.

Scheme	Trans.	$\mathcal{P}$ time	$\mathcal{V}$ time	Proof size
KZG [25]	✗	$O(N^2)$	$O(1)$	$O(1)$
AMT [35]	✗	$O(N \log N)$	$O(\log N)$	$O(\log N)$
hbACSS [40]	✓	$O(N^2)$	$O(N)$	$O(\log N)$
Our KZG-based	✗	$O(N \log N)$	$O(1)$	$O(1)$
Our Transparent	✓	$O(N \log N)$	$O(\log^2 N)$	$O(\log^2 N)$

• **Trans.** means without a trusted setup. $\mathcal{P}$ **time** represents the dealer time for producing *all N proofs*. $\mathcal{V}$ **time** represents the verification time per verifier.

commitment c. Now, the dealer can use a non-interactive zero-knowledge proof system to produce a proof that vouches for the evaluation at a specific point. Since the dealer needs to produce a proof for each of the N verifiers, the naïve approach is to repeat the zero-knowledge proof N times — however, this approach would result in at least N^2 prover time for producing all N proofs (since evaluating the polynomial at each point requires at least N amount of computation).

Note that if the dealer only had to evaluate the polynomial f at N different points without having to produce proofs, this could be accomplished through the Fast Fourier Transform (FFT) in $O(N \log N)$ time. The intriguing question is whether we can evaluate the polynomial at all N points and produce all N proofs in $O(N \log N)$ time as well.

A more general problem: one-to-many zero-knowledge proof. To answer this question, we in fact turn our attention to a more general problem. Suppose that there is some circuit C with N different outputs. There are N verifiers, and each of them cares about receiving and verifying one of the outputs of C. Can the prover produce all N proofs in time $\widetilde{O}(|C|)$ where $|C|$ denotes the size of the circuit[2]? Note that in comparison, the naïve approach of repeating the ZKP independently N times would result in at least $N \cdot |C|$ prover time.

Brief background on GKR. To achieve this, we will base our scheme on Virgo [41], which is in turn based on the famous GKR protocol [19]. For simplicity, we will explain our intuition without worrying about zero-knowledge, and therefore we can think of the original GKR protocol. It helps to first consider the interactive version, and then we will describe a new Fiat-Shamir-style transformation to make our interactive protocol non-interactive in the random oracle model.

In the GKR protocol, the prover starts with the output layer (henceforth called the last layer). Proving the output layer boils down to proving a sumcheck statement for a special polynomial that encodes the wiring structure of the output layer of the circuit. This sumcheck would then be reduced to proving two sumchecks for the last but one layer, which can be coalesced into a single sumcheck proof (for the last but one layer) by taking random linear combinations. This goes on recursively layer by layer. If the prover simply ran GKR with each verifier separately, the prover would have to prove a different statement to each verifier at every layer.

Idea 1: using an extra sumcheck protocol to unify statements for all layers. Our idea is to introduce a clever sumcheck protocol after the output layer, such that after the sumcheck protocol with each verifier, the prover would be proving the same statement to all N verifiers for all other layers, *as long as all verifiers use the same random challenges in every round of the protocol*. This way, except for proof component corresponding to the extra sumcheck, computing the proof components for all other layers is a shared effort among all verifiers. Moreover, we propose a new algorithm for the prover to run all sumchecks with N verifiers in $O(N \log N)$ time. We defer the details of the construction to Section 3.

Idea 2: a new Fiat-Shamir-style transformation to make it non-interactive. With the first idea, we could achieve prover batching as long as all verifiers use the same random challenges in every round of the interactive protocol. Our final construction is non-interactive, and to achieve this we describe a new Fiat-Shamir-style transformation such that the prover can emulate the verifiers' challenges non-interactively by making queries to a random oracle.

Note that applying the standard Fiat-Shamir transformation does not work for us since we additionally require that the random challenges are shared among all verifiers in each round. Recall that the standard Fiat-Shamir transformation queries the random oracle on the transcript with the verifier so far. In our case, the transcript with each verifier differs in the output layer. If we simply hashed the entire transcript, it would result in different challenges for different verifiers.

A conceptually simple but somewhat inefficient approach to overcome this discrepancy among verifiers is to use a Merkle tree to hash all verifiers' transcripts and use the root as a unified random challenge among all verifiers. The prover also needs to send the corresponding Merkle branch to each individual verifier, such that a verifier can ascertain that its view of the transcript so far has been incorporated in generating the random challenge. A similar idea was suggested by Yurek et al. [40] but for a somewhat different purpose. The drawback with this approach is that it incurs a logarithmic blowup in proof size and verifier time for every round of the protocol.

Our final approach is a hybrid one. We apply the Merkle tree only to the first logarithmically many rounds, i.e., rounds for the extra sumcheck protocol after the output layer. Recall that for every other layer in the circuit, the prover would be proving the same statement to all verifiers, and therefore the transcripts among the verifiers would converge after the extra sumcheck protocol. Thus for all other layers, we can rely on the standard Fiat-Shamir heuristic. This hybrid approach would save us a logarithmic factor in comparison with applying Merkle hash tree to every round.

Proving soundness of our new Fiat-Shamir-style transform. In the most general setting, the standard Fiat-Shamir transformation is known to work only for constant-round interactive proofs if we want the soundness loss in the reduction to be polynomially bounded. In our case, the original inter-

[2] We use the notation $\widetilde{O}$ to hide polylogarithmic factors.

Table 2: Comparison of our schemes and prior works in VSS and DKG settings. We assume $t = \Theta(N)$.

Scheme	Trans.	Broad-cast	Optimistic case			Worst-case‡		
			$\mathcal{P}$ time	$\mathcal{V}$ time	Communication	$\mathcal{P}$ time	$\mathcal{V}$ time	Communication
Feldman-VSS [15]	✓	$O(N)$	$O(N\log N)$	$O(N)$	$O(N)$	$O(N\log N)$	$O(N^2)$	$O(N)$
eVSS [25]	✗	$O(1)$	$O(N^2)$	$O(1)$	$O(1)$	$O(N^2)$	$O(N)$	$O(N)$
AMT-VSS [35]	✗	$O(1)$	$O(N\log N)$	$O(\log N)$	$O(\log N)$	$O(N\log N)$	$O(N\log N)$	$O(N\log N)$
hbACSS-VSS [40]	✓	$O(1)$	$O(N^2)$	$O(N)$	$O(\log N)$	$O(N^2)$	$O(N\log N)$	$O(N\log N)$
Our KZG-based-VSS	✗	$O(1)$	$O(N\log N)$	$O(1)$	$O(1)$	$O(N\log N)$	$O(N)$	$O(N)$
Our Transparent-VSS	✓	$O(1)$	$O(N\log N)$	$O(\log^2 N)$	$O(\log^2 N)$	$O(N\log N)$	$O(N\log^2 N)$	$O(N\log^2 N)$

Table 3: VSS schemes

Scheme	Trans.	Broadcast	Optimistic case		Worst-case‡	
			Computation	Communication	Computation	Communication
JF-DKG [18]	✓	$O(N)$	$O(N^2)$	$O(N)$	$O(N^3)$	$O(N^2)$
eJF-DKG [23]	✗	$O(1)$	$O(N^2)$	$O(N)$	$O(N^2)$	$O(N^2)$
AMT-DKG [35]	✗	$O(1)$	$O(N\log N)$	$O(N\log N)$	$O(N^2\log N)$	$O(N^2\log N)$
hbACSS-DKG [40]	✓	$O(1)$	$O(N^2)$	$O(N\log N)$	$O(N^2\log N)$	$O(N^2\log N)$
Our KZG-based-DKG	✗	$O(1)$	$O(N\log N)$	$O(N)$	$O(N^2)$	$O(N^2)$
Our Transparent-DKG	✓	$O(1)$	$O(N\log^2 N)$	$O(N\log^2 N)$	$O(N^2\log^2 N)$	$O(N^2\log^2 N)$

Table 4: DKG schemes (per party overhead)

In this table we only compare VSS/DKG schemes that are in the same synchronous model and incur a constant number of rounds. We discuss other schemes in different settings (e.g., asynchronous, gossip model with $O(\log N)$ rounds) in Section 1.3. **Trans.** means without a trusted setup. $\mathcal{P}$ **time** represents the dealer's computation for producing N *proofs* for N receivers. $\mathcal{V}$ **time** represents the verification time per receiver. **Communication** represents the proof size for each receiver in VSS setting and the total communication for each party in DKG setting. ‡ **Worst-case** represents $\Theta(N)$ bad shares, which results in the complaint round.

active protocol is $O(d \cdot \log N)$ rounds where d denotes the depth of the circuit C. Nonetheless, we can still prove the soundness of our new Fiat-Shamir-style transformation with only polynomial loss in soundness in our reduction. To prove this, we suggest a different way to view our transformation. We focus on the perspective of a single verifier and consider a variant (denoted T_{du}) of our original interactive protocol. In T_{du}, we introduce some dummy rounds and dummy messages which correspond to hash computations in the Merkle tree. We then view our final non-interactive proof as applying an alternative heuristic transformation to the modified protocol T_{du}. Using this alternative view, we are able to use techniques from Ben-Sasson et al. [4] to prove soundness. First, we show that the protocol T_{du} satisfies a stronger notion of soundness called *state restoration soundness*. Given the stronger soundness property, we can show that applying the aforementioned heuristic transformation to T_{du} gives a sound non-interactive protocol in the random oracle model.

Putting everything together. So far, we have described our ideas neglecting the zero knowledge requirement. It is relatively easy to augment the protocol with zero knowledge using techniques proposed in [10, 39, 41]. The modifications to the protocol do not fundamentally alter the soundness proof of our Fiat-Shamir-style transformation. We refer to the details of how to achieve zero knowledge in the full version.

Summarizing the above, we now have a non-interactive, one-to-many zero-knowledge proof system with a batched prover. One-to-many polynomial commitment is a special case of this more general problem where the circuit C is an FFT circuit that evaluates the polynomial at N different points. We emphasize that in solving the one-to-many polynomial commitment problem, we actually come up with a "one-to-many zero-knowledge proof" technique that is much more general, and can be of independent interest and will likely lead to broader applications.

1.2.2 New Prover Batching

We propose a new prover batching technique for the KZG polynomial commitment. We review the KZG polynomial commitment scheme in Section 4.1. Our novel technique is the following. We show that if the dealer needs to open the polynomial f at the points $\omega, \omega^2, \ldots, \omega^N$ where ω is the N-th root of unity, then the N proof terms can be computed efficiently using a constant number of FFT and inverse FFT invocations. Moreover, the FFT computation can be directly applied to the public parameters of the KZG commitment scheme in the base group of a bilinear map, without knowing the trapdoor, as FFT only involves additions and scalar multiplications. Observing this requires some more involved algebraic manipulations which we defer to Section 4.2.

1.3 Related work

VSS. Chor et al. [11] were the first to introduce the notion of VSS. Feldman [15] constructed the first efficient Feldman-VSS scheme with homomorphic encryption schemes. Feldman-VSS is computational hiding and information-theoretic binding. The following work of Pedersen [31] presented a counterpart protocol with information-theoretic hiding and computational binding. However, both schemes broadcast $O(N)$ messages during the dealing phase and cost $O(N)$ time for each verifier to check the correctness of the share. Kate et al. [25] reduced the broadcast message and the verification time to $O(1)$ in eVSS by the constant-sized KZG polynomial commitment. Their polynomial commitment needed a trusted setup and increased the dealer's computation to $O(N^2)$. Tomescu et al. [35] achieved a quasi-linear dealing time at the cost of the $O(\log N)$ verification time. The communication for each verifier also increased to $O(\log N)$.

DKG. VSS plays an essential role in constructing DKG protocols. Ingemarsson and Simmons [22] first proposed DKG. Pederson [31] improved their scheme for discrete log-based cryptosystems. Gennaro et al. [18] showed that the secret generated by Pederson's scheme was biased and fixed the problem in their JF-DKG schemes. Neji et al. [30] debiased the secret by a more efficient method. Moreover, JF-DKG scheme was converted into an adaptively secure DKG by Canetti et al. [8]. All DKG protocols mentioned above need $O(N)$ broadcast messages. Later on, Kate's eJF-DKG [23] tamed the broadcasting cost to $O(1)$ on top of eVSS. Tomescu et al. [35] built AMT-DKG based on their AMT-VSS scheme to achieve a space-time trade-off for eJF-DKG. In this work, our KZG-based-DKG applies our KZG-based-VSS directly to DKG to remove the overhead on time and space in eJF-DKG and AMT-DKG respectively. Recently, Gurkan et al. [21] presented an aggregatable VUF-DKG without a trusted setup. It achieves $O(N\log^2 N)$ computation and communication complexity, which is asymptotically the same as our Transparent-DKG. However, the scheme is in the "gossip" model where each party sends messages to her neighbors and has $\log N$ rounds.

Disambiguation. In our experiments, we focus on VSS and DKG in the *synchronous* setting. Earlier works have also shown that polynomial commitment schemes give rise to *asynchronous* VSS and DKG schemes [14, 17, 24, 26, 40]. In the asynchronous setting, the resulting VSS and DKG schemes would also benefit from one-to-many prover batching. An interesting future direction is to apply our prover batching technique and improve VSS and DKG in the asynchronous setting.

2 Preliminary

We use $\mathsf{negl}(\cdot) : \mathbb{N} \to \mathbb{R}$ to denote the negligible function, where for each positive polynomial $f(\cdot)$, $\mathsf{negl}(k) < \frac{1}{f(k)}$ for sufficiently large integer k. We use $\mathsf{nonegl}(\cdot) : \mathbb{N} \to \mathbb{R}$ to denote the complement of $\mathsf{negl}(\cdot)$. Let λ denote the security parameter. "PPT" stands for Probabilistic Polynomial Time. We use $f(), h()$ for polynomials, x, y, z for single variable, $\vec{x}, \vec{y}, \vec{z}$ for vectors of variables and $\vec{g}, \vec{u}, \vec{v}$ for vectors of values. x_i denotes the i-th element in $\vec{x}$. We use capital letters such as A to represent arrays in algorithms, and $A[i]$ denotes the i-th element in the array. For a multivariate polynomial f, its "variable-degree" is the maximum degree of f in any of its variables. Let $[k]$ denote the set of $\{0, 1, \ldots, k-1\}$.

Let $\mathbb{F}$ be a prime field. We use $\omega^0, \ldots, \omega^{N-1}$ to denote N roots of unity on $\mathbb{F}$ such that $\omega^N = 1$ in $\mathbb{F}$. Let $\mathbb{H} = \{\omega^0, \ldots, \omega^{N-1}\}$ be a subset of $\mathbb{F}$. We often rely on polynomial arithmetics, which can be efficiently performed via fast Fourier transforms (FFT) and their inverses (IFFT). In particular, polynomial evaluations and interpolations over $\mathbb{H}$ can be performed in $O(N \log N)$ field operations via the standard FFT and IFFT algorithms [12].

Convolution of two vectors: Let A, B be two arrays of length n. Their convolution, denoted as $C = A * B$, is defined as: $C[j] = \sum_{i=0}^{j} A[i]B[j-i]$, for $j \in [2n]$, assuming the values of vectors A and B are zeros when the index is out of range (i.e., $\geq n$). It is known that the convolution is equivalent to the multiplication of two polynomials, which can be computed efficiently using FFT and inverse FFT. In particular, $C = \mathsf{IFFT}(\mathsf{FFT}(A) \odot \mathsf{FFT}(B))$, where $\odot$ denotes the Hadamard (element-wise) product, and the two FFTs evaluate A and B on $2n$ points.

Merkle Tree. Merkle tree [29] has been widely used for the vector commitment because of its simplicity and efficiency. The prover time is linear in the size of the vector while the verifier time and proof size are logarithmic in the size of the vector. Given a vector of $\vec{r} = (r_0, \ldots, r_{N-1})$, it consists of three algorithms:$\mathsf{rt} \leftarrow \mathsf{MT.Commit}(\vec{r})$, $(r_i, \mathsf{path}_i) \leftarrow \mathsf{MT.Open}(i, \vec{r})$ and $\{1, 0\} \leftarrow \mathsf{MT.Verify}(\mathsf{rt}, i, r_i, \mathsf{path}_i)$.

We require not only the root to be hiding, but also opening $\vec{r}$ at the index of i does not leak any information about $\vec{r}$ other than r_i, which is treated as the privacy property of Merkle tree. Formally speaking, for any vector $\vec{r}$ of size N, any PPT algorithm $\mathcal{A}$, there exists a simulator $\mathcal{S}_{\mathsf{MT}}$ such that $\mathsf{rt} \leftarrow \mathsf{MT.Commit}(\vec{r})$, $(r_i, \mathsf{path}_i) \leftarrow \mathsf{MT.Open}(i, \vec{r})$, $\mathsf{rt}', \mathsf{path}'_i \leftarrow \mathcal{S}_{\mathsf{MT}}(r_i, N)$,

$$|\Pr[\mathcal{A}(\mathsf{rt}, r_i, \mathsf{path}_i) = 1] - \Pr[\mathcal{A}(\mathsf{rt}', r_i, \mathsf{path}'_i) = 1]| \leq \mathsf{negl}(\lambda).$$

The privacy property can be achieved by concatenating a random number on each leaf of Merkle tree when committing.

We define interactive proofs and VSS in Appendix A. We also give the standard construction of VSS from polynomial commitment in Appendix A.

2.1 Polynomial commitment

Univariate polynomial commitment. Let $\mathbb{F}$ be a finite field and f be a polynomial on $\mathbb{F}$ with degree D. A univariate polynomial commitment (PC) for $f \in \mathbb{F}^D[X]$ and $a \in \mathbb{F}$ consists of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{KeyGen}(1^\lambda, D)$: Given the security parameter and a bound on the degree of the polynomial, the algorithm generates public parameter pp.

- $\mathsf{com}_f \leftarrow \mathsf{Commit}(f, r_f, \mathsf{pp})$: Given a polynomial $f(x) = \sum_{i=0}^{D} c_i x^i$, the prover commits f with the private randomness r_f and the public parameter pp. r_f can be none.
- $(y, \pi) \leftarrow \mathsf{Open}(f, r_f, a, \mathsf{pp})$: For an evaluation point a, the prover computes $y = f(a)$ and the proof π.
- $\{1, 0\} \leftarrow \mathsf{Verify}(\mathsf{com}_f, a, y, \pi, \mathsf{pp})$. Given the commitment com_f, the evaluation point a, the answer y and the proof π, the verifier checks the correctness of the evaluation.

Definition 1. *A* PC *scheme satisfies the following properties:*

- ***Completeness.*** *For any polynomial $f \in \mathbb{F}^D[X]$ and $a \in \mathbb{F}$, the following probability is 1.*

$$\Pr\left[\begin{array}{rl} \mathsf{pp} \leftarrow \mathsf{KeyGen}(1^\lambda, D) & \\ \mathsf{com}_f \leftarrow \mathsf{Commit}(f, r_f, \mathsf{pp}): & \mathsf{Verify}(\mathsf{com}_f, a, y, \pi, \mathsf{pp}) = 1 \\ (y, \pi) \leftarrow \mathsf{Open}(f, r_f, a, \mathsf{pp}) & \end{array}\right]$$

- ***Proof of Knowledge.*** *For any polynomial-sized circuit $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}$ and a negligible function* $\mathsf{negl}(\cdot)$*, such that for any auxiliary string z and any $\lambda \in \mathbb{N}$, the following probability is* $\mathsf{negl}(\lambda)$.

$$\Pr\left[\begin{array}{rl} \mathsf{pp} \leftarrow \mathsf{KeyGen}(1^\lambda, D) & \\ (\pi^*, \mathsf{com}^*, y^*, a^*) \leftarrow \mathcal{A}(1^\lambda, z, \mathsf{pp}): & \mathsf{Verify}(\mathsf{com}^*, a^*, y^*, \pi^*, \mathsf{pp}) = 1 \\ f^* \leftarrow \mathcal{E}(1^\lambda, z, \mathsf{pp}) & \wedge f^*(a^*) \neq y^* \end{array}\right]$$

If a PC *scheme satisfies an additional property of zero knowledge, then it is a zero-knowledge univariate polynomial commitment scheme (*zkPC*).*

- ***Zero Knowledge.*** *For security parameter λ, polynomial f, adversary $\mathcal{A}$, and simulator $\mathcal{S}$, consider the following two experiments:*

  $\mathsf{Real}_{\mathcal{A},f}(1^\lambda)$*:*

  - $\mathsf{pp} \leftarrow \mathsf{KeyGen}(1^\lambda, D)$
  - $\mathsf{com}_f \leftarrow \mathsf{Commit}(f, r_f, \mathsf{pp})$
  - $a \leftarrow \mathcal{A}(1^\lambda, \mathsf{com}_f, \mathsf{pp})$
  - $(y, \pi) \leftarrow \mathsf{Open}(f, r_f, a, \mathsf{pp})$
  - $b \leftarrow \mathcal{A}(1^\lambda, \mathsf{com}_f, a, y, \pi, \mathsf{pp})$
  - *Output b*

  $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda)$*:*

  - $(\mathsf{com}_f, \mathsf{pp}, \mathsf{trap}) \leftarrow \mathcal{S}(1^\lambda, D)$
  - $a \leftarrow \mathcal{A}(1^\lambda, \mathsf{com}_f, \mathsf{pp})$
  - $\pi \leftarrow \mathcal{S}(\mathsf{com}_f, a, f(a), \mathsf{pp}, \mathsf{trap})$.
  - $b \leftarrow \mathcal{A}(1^\lambda, \mathsf{com}_f, a, f(a), \pi, \mathsf{pp})$
  - *Output b*

  For any non-uniform polynomial-time adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for all polynomial $f \in \mathbb{F}^D[X]$,

$$|\Pr[\mathsf{Real}_{\mathcal{A},f}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda).$$

A polynomial commitment scheme is said to be transparent if the public parameter pp *is simply a uniform random string, i.e., there is no secret state to generate* pp.

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 and zkMVPC to denote the multivariate schemes.

Protocol 1 (Sumcheck). *It proceeds in ℓ rounds.*

- *In the first round, $\mathcal{P}$ sends a univariate polynomial*

$$h_1(x_1) \overset{def}{=} \sum_{b_2,\ldots,b_\ell \in \{0,1\}} h(x_1, b_2, \ldots, b_\ell),$$

  $\mathcal{V}$ checks $\mu = h_1(0) + h_1(1)$. Then $\mathcal{V}$ sends a random challenge $r_1 \in \mathbb{F}$ to $\mathcal{P}$.
- *In the i-th round, where $2 \leq i \leq \ell - 1$, $\mathcal{P}$ sends a univariate polynomial*

$$h_i(x_i) \overset{def}{=} \sum_{b_{i+1},\ldots,b_\ell \in \{0,1\}} h(r_1, \ldots, r_{i-1}, x_i, b_{i+1}, \ldots, b_\ell),$$

  $\mathcal{V}$ checks $h_{i-1}(r_{i-1}) = h_i(0) + h_i(1)$, and sends a random challenge $r_i \in \mathbb{F}$ to $\mathcal{P}$.
- *In the ℓ-th round, $\mathcal{P}$ sends a univariate polynomial*

$$h_\ell(x_\ell) \overset{def}{=} h(r_1, r_2, \ldots, r_{\ell-1}, x_\ell),$$

  $\mathcal{V}$ checks $h_{\ell-1}(r_{\ell-1}) = h_\ell(0) + h_\ell(1)$. The verifier generates a random challenge $r_\ell \in \mathbb{F}$. Given oracle access to an evaluation $h(r_1, r_2, \ldots, r_\ell)$ of h, $\mathcal{V}$ will accept if and only if $h_\ell(r_\ell) = h(r_1, r_2, \ldots, r_\ell)$. The instantiation of the oracle access depends on the application of the sumcheck protocol.

2.2 Interactive Proofs for Layered Circuits

We present the GKR protocol, an efficient interactive proof for layered arithmetic circuits by Goldwasser et al. [19].

2.2.1 Sumcheck Protocol

The GKR protocol uses the sumcheck protocol as a major building block. The problem is to sum a multivariate polynomial $h : \mathbb{F}^\ell \to \mathbb{F}$ on the Boolean hypercube: $\sum_{b_1, b_2, \ldots, b_\ell \in \{0,1\}} h(b_1, b_2, \ldots, b_\ell)$. Directly computing the sum requires an exponential time in ℓ, as there are 2^ℓ combinations of $b_1, \ldots, b_\ell$. Lund et al. [28] proposed a *sumcheck* protocol that allows a verifier $\mathcal{V}$ to delegate the computation to a computationally unbounded prover $\mathcal{P}$.We describe the sumcheck protocol in Protocol 1. The proof size of the protocol is $O(D\ell)$, where D is the variable-degree of h, as in each round, $\mathcal{P}$ sends a univariate polynomial of one variable in h, which can be uniquely defined by $D+1$ points. The verifier time is $O(D\ell)$. The prover time depends on the degree and the sparsity of h, and we will give the complexity later in our scheme. The sumcheck protocol is complete and sound with $\varepsilon = \frac{D\ell}{|\mathbb{F}|}$.

Definition 2 (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 $\vec{b}$.

Definition 3 (Identity function). *Let $\beta : \{0,1\}^\ell \times \{0,1\}^\ell \rightarrow \{0,1\}$ 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: $\tilde{\beta}(\vec{x},\vec{y}) = \prod_{i=1}^{\ell}((1-x_i)(1-y_i)+x_iy_i)$.*

2.2.2 GKR Protocol

With the sumcheck protocol as a building block, Goldwasser et al. [19] proposed an interactive proof for the evaluation of layered arithmetic circuits. Let C be a layered arithmetic circuit with depth d over a finite field $\mathbb{F}$. Each gate in the i-th layer takes inputs from two gates in the $(i+1)$-th layer; layer 0 is the output layer and layer d is the input layer. The values in layer i of the circuit can be written as a sumcheck equation of the values in layer $i+1$. Following the convention in prior works of GKR protocols [13, 34, 39, 41, 42], we denote the number of gates in the i-th layer as S_i and let $s_i = \lceil \log S_i \rceil$. We then define a function $V_i : \{0,1\}^{s_i} \rightarrow \mathbb{F}$ that takes a binary string $\vec{b} \in \{0,1\}^{s_i}$ and returns the output of gate $\vec{b}$ in layer i, where $\vec{b}$ is called the gate label. With this definition, V_0 corresponds to the output of the circuit, and V_d corresponds to the input layer. We also define two additional functions $add_i, mult_i : \{0,1\}^{s_{i-1}+2s_i} \rightarrow \{0,1\}$, 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:

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

With Equation 1, as $\tilde{add}_{i+1}$ and $\tilde{mult}_{i+1}$ are publicly known, upon receiving the output, the verifier can reduce a claim of $\tilde{V}_0(\vec{g}^{(0)})$ to a claim $\tilde{V}_1(\vec{g}^{(1)})$ about layer 1, and recursively to $\tilde{V}_d(\vec{g}^d)$ through sumcheck protocols layer by layer. With the optimal algorithms for the prover in the GKR protocol proposed in [39], we have the following theorem:

Theorem 1. *[39]. Let $C : \mathbb{F}^n \rightarrow \mathbb{F}^k$ 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 $\mathcal{P}$ is $O(|C|)$. When C has regular wiring pattern[3], the running time of $\mathcal{V}$ is $O(n+k+d\log|C|)$.*

Lifting GKR protocols to argument systems. The GKR protocol is not an argument system supporting witness from $\mathcal{P}$, as in the last round, $\mathcal{V}$ needs to evaluate $\tilde{V}_d$ defined by the input of the circuit at a random point locally. To address this problem, in [42], Zhang et al. first construct an argument system by combining the polynomial commitments with the GKR protocol. In their scheme, $\mathcal{P}$ first commits to the multilinear extension of $\mathcal{P}$'s witness by MVPC.Commit before the GKR protocol. In the last random of the GKR protocol, instead of evaluating locally, $\mathcal{V}$ queries $\mathcal{P}$ the evaluation on $\mathcal{P}$'s witness. $\mathcal{P}$ invokes MVPC.Open to prove the correctness of the evaluation and $\mathcal{V}$ validates it using MVPC.Verify. Combined with $\mathcal{V}$'s public input, $\mathcal{V}$ is able to verify the last claim about $\tilde{V}_d$ in the GKR protocol. Subsequent works [37,39,41,43] improve the efficiency and achieve zero-knowledge based on the framework. We follow the same framework in our scheme with a transparent setup, and we present the protocol explicitly in Section 3.

3 Transparent Polynomial Commitment with Prover Batching

We first present our transparent polynomial commitment scheme with prover batching for multiple evaluations. There are several candidates of transparent polynomial commitment schemes recently [7, 27, 36, 37, 41] with the prover time of $\tilde{O}(t)$ for a single evaluation. However, in the application of the VSS scheme in Protocol 4, if the dealer naively runs the transparent polynomial commitment scheme on N evaluations separately, the running time will be $\tilde{O}(Nt)$.

In our scheme, we reduce the prover time of generating all N proofs to $O(N \log N)$ field operations, which is asymptotically the same as evaluating the polynomial at N points. We propose the notion of *one-to-many* zero knowledge arguments, where each verifier receives one output out of the entire output of a common computation represented by circuit C. Instead of running a zero knowledge proof protocol with each verifier separately, which may take $\tilde{O}(N|C|)$ time for the prover in the worst case, our scheme reduces the prover time to $\tilde{O}(|C|+N\log N)$.

When applied to the polynomial commitment for VSS, we set the evaluations at powers of the N-th root of unity ω (i.e., $\omega^N = 1 \mod p$). In this way, we realize the polynomial commitment with prover batching by instantiating the circuit C in our one-to-many zero knowledge argument with the classical butterfly circuit [38] for the FFT algorithm. The circuit takes the coefficients of the polynomial f as input, and outputs $f(\omega^0), \ldots, f(\omega^{N-1})$, where each verifier $\mathcal{V}_j$ receives $f(\omega^j)$. With our one-to-many zero knowledge argument, the prover is able to generate all proofs in time $O(N \log N)$.

Below, we will first describe our one-to-many zero knowledge argument scheme assuming that somehow, all verifiers send the same challenge in every round (Protocol 2) — to

[3] "Regular" circuits is defined in [13, Theorem A.1]. Roughly speaking, it means the mutilinear extension of its wiring predicates can be evaluated at a random point in time $O(\log |C|)$.

aid understanding, the reader may assume for the time being that all verifiers query a trusted random oracle in each round to generate a common random challenge. Later, we will describe a new Fiat-Shamir-style transformation (Protocol 3) to make Protocol 2 non-interactive, such that all verifiers would effectively share the same challenges in this non-interactive version. Finally, we will prove the soundness of our new Fiat-Shamir transformation using the techniques inspired by Ben-Sasson, Chiesa, and Spooner [4]. For simplicity, we first describe a simplified version of our protocol *without zero-knowledge*. In full version, we will describe how to use standard techniques to additionally achieve zero-knowledge.

3.1 One-to-Many Argument System Given Shared Random Challenges

Following the notation of the GKR protocol in Section 2.2, we denote the entire output of circuit C as $\tilde{V}_0(\vec{x})$ for $\vec{x} \in \{0,1\}^{\log N}$. Suppose the size of the output is N and each verifier $\mathcal{V}_j$ receives one output $\tilde{V}_0(\vec{j})$, where $\vec{j}$ is the binary representation of j. Using its multilinear extension, we can write each $\tilde{V}_0(\vec{j})$ as a sumcheck of $\tilde{V}_0(\vec{x})$ using the identity function $\tilde{\beta}$:

$$\tilde{V}_0(\vec{j}) = \sum\nolimits_{\vec{x}\in\{0,1\}^{\log N}} \tilde{\beta}(\vec{j},\vec{x})\tilde{V}_0(\vec{x}).$$

$\mathcal{P}$ and $\mathcal{V}_j$ can run one sumcheck protocol to reduce the claim about $\tilde{V}_0(\vec{j})$ to the claim about $\tilde{V}_0(\vec{g}^{(0)})$ for $\vec{g}^{(0)} \in \mathbb{F}^{\log N}$. This is equivalent to adding an additional layer of "selector" that selects the j-th output for $\mathcal{V}_j$. Assuming all verifiers share the same random challenge in every round, they will share the random vector of $\vec{g}^{(0)}$ during the sumcheck protocol and share the last claim of $\tilde{V}_0(\vec{g}^{(0)})$. Then $\mathcal{P}$ invokes the GKR-based argument on the circuit C to prove to all verifiers the correctness of $\tilde{V}_0(\vec{g}^{(0)})$. Given the common randomness during the invocation, the prover computes the same message for all verifiers in every round. Thus in this step, the total computational cost of $\mathcal{P}$ the same as that of proving to a single verifier, which is $\tilde{O}(|C|)$. We present the formal protocol in Protocol 2.

It remains to show that in Step 4 of Protocol 2, the prover can generate all messages in the sumcheck protocols with all the verifiers in $O(N \log N)$ time. As there are N different sumcheck protocols and the size of the polynomial in each sumcheck is $O(N)$, naively running Step 4 takes $O(N^2)$ time using existing techniques. However, we observe that all the sumcheck protocols with different $\mathcal{V}_j$ share the same polynomial $\tilde{V}_0(\vec{x})$. The only difference is that the identity function $\tilde{\beta}$ takes different $\vec{j}$. By utilizing the special structure of the identity function $\tilde{\beta}$, we are able to come up with a new algorithm to run all sumcheck protocols efficiently. The algorithm initializes and updates a lookup table based on $\tilde{V}_0$ once for all verifiers in every round. Then using the lookup table, the prover is able to generate the message for each sumcheck protocol in every round in a constant time. Thus the prover time is $O(N)$ per round, and thus is $O(N \log N)$ in total. We present the formal algorithm in Algorithm 1. As shown in Step 6 of Algorithm 1, in the i-th round of the sumcheck (see Protocol 1 for the message in each round of sumcheck), the messages defined by $\tilde{V}_0$ are shared among all verifiers and can be computed by the prover in $O(N/2^i)$ time. In Step 7, the lookup table is updated based on the randomness received in this round in $O(N/2^i)$. Then in Step 9, the prover generates the messages for each verifier $\mathcal{V}_j$ utilizing the closed form of the identity function as described in Section 2. Since the sumcheck protocol has $\log N$ rounds in total, the total prover time is $O(N + \frac{N}{2} + \ldots + 1 + N \log N) = O(N \log N)$. Using Algorithm 1 for Step 4 in Protocol 2, and the transparent zkMVPC scheme in [41] with $O(n \log n)$ prover time, $O(\log^2 n)$ verifier time and proof size, where n is the size of the input plus the witness, we have the following theorem.

Protocol 2. Batching prover computation for N verifiers that share random challenges.
Let λ be the security parameter. Let C: $\mathbb{F}^n \to \mathbb{F}^N$ be a d-depth layered arithmetic circuit. For any $j \in [N]$, $\mathcal{P}$ needs to convince $\mathcal{V}_j$ that $\boldsymbol{out}_j = [C(\boldsymbol{in})]_j$ where $[C(\boldsymbol{in})]_j$ is the j-th output of the circuit given input $\boldsymbol{in}$, and $\boldsymbol{out}_j$ is the the claimed result for $\mathcal{V}_j$. Without loss of generality, assume n and N are both powers of 2 and we can pad them if not.
Here, we assume that all verifiers obtain their random challenge from a common random oracle in each round.

1. *Set* $\mathsf{pp} \leftarrow \mathsf{MVPC.KeyGen}(1^\lambda)$.
2. *$\mathcal{P}$ invokes* $\mathsf{MVPC.Commit}(\tilde{V}_d, \mathsf{pp})$ *to generate* $\mathsf{com}_{\tilde{V}_d}$ *and broadcasts* $\mathsf{com}_{\tilde{V}_d}$ *to all verifiers. $\tilde{V}_d$ is the multilinear extension of input values, as defined in the GKR protocol.*
3. *For each $j \in [N]$, $\mathcal{P}$ sends $\tilde{V}_0(\vec{j})$ as $\boldsymbol{out}_j$ to $\mathcal{V}_j$ separately.*
4. *For each $j \in [N]$, $\mathcal{P}$ and $\mathcal{V}_j$ run a sumcheck protocol on*

$$\tilde{V}_0(\vec{j}) = \sum_{\vec{x}\in\{0,1\}^{\log N}} \tilde{\beta}(\vec{j},\vec{x})\tilde{V}_0(\vec{x}),\ \vec{j} \text{ is the binary string of } j$$

At the end of the protocol, $\mathcal{V}_j$ receives $\tilde{V}_0(\vec{g}^{(0)})$ for the common random vector of $\vec{g}^{(0)}$. $\mathcal{V}_j$ computes $\tilde{\beta}(\vec{j},\vec{g}^{(0)})$ and checks the last statement of the sumcheck protocol.

5. *For all verifiers $\mathcal{P}$ invokes the GKR protocol on the circuit C to generate the proof given $\tilde{V}_0(\vec{g}^{(0)})$.*
6. *For all verifiers, in the last round of the GKR protocol, they have the claim about $\tilde{V}_d(\vec{g}^{(d)})$. $\mathcal{P}$ and $\mathcal{V}_j$ invoke* $\mathsf{MVPC.Open}$ *and* $\mathsf{MVPC.Verify}$ *on $\tilde{V}_d(\vec{g}^{(d)})$ with* $\mathsf{com}_{\tilde{V}_d}$ *and* pp. *If it is equal to $\tilde{V}_d(\vec{g}^{(d)})$ sent by $\mathcal{P}$, $\mathcal{V}_j$ outputs* 1, *otherwise $\mathcal{V}_j$ outputs* 0.

Theorem 2. *For each $\mathcal{V}_j$ and $\mathcal{P}$, Protocol 2 is an argument system for the function $[\boldsymbol{out}]_j = [C(\boldsymbol{in})]_j$ such that $\boldsymbol{out} = C(\boldsymbol{in})$ with soundness $O(d \log |C|/|\mathbb{F}|)$. The proof size is $O(d \log |C| + \log^2 n)$ and the verifier time is $O(d \log |C| + \log^2 n)$. If N verifiers have the common random challenge in every round, the total prover time is $O(|C| + N \log N + n \log n)$.*

Algorithm 1 $\{a_{1,0},\dots,a_{1,N-1},\dots,a_{\log N,0},\dots,a_{\log N,N-1}\} \leftarrow \mathsf{SumCheck}\ (\tilde{V}(\vec{x}),\, g_1^{(0)},\,\dots,\, g_{\log N}^{(0)})$

Input: $\tilde{V}(\vec{x})$ for $\vec{x} \in \{0,1\}^{\log N}$, random $g_1^{(0)},\dots,g_{\log N}^{(0)}$;

Output: For each $j \in [N]$, $\log N$ sumcheck messages $(a_{1,j},\dots,a_{\log N,j})$ for $\tilde{V}(\vec{j}) = \sum_{x\in\{0,1\}^{\log N}} \tilde{\beta}(\vec{j},\vec{x})\tilde{V}_0(\vec{x})$. Each message $a_{i,j}$ consists of 3 elements $(a_{i0,j},a_{i1,j},a_{i2,j})$;

```
Initialize $\mathsf{beta}_j = 1$ for all $j \in [N]$.
Initialize an array $V[B] = \tilde{V}_0(\vec{b})$ for all $\vec{b} \in \{0,1\}^{\log N}$.    // $\vec{b}$ is the binary representation of integer $B$.
for Round $i = 1,\dots,\log N$ do
    for Evaluation point $r = 0,1,2$ do
        for $\vec{b} \in \{0,1\}^{\log N - i}$ do
            $\tilde{V}_0(g_1^{(0)},\dots,g_{i-1}^{(0)},r,\vec{b}) = V[B]\cdot(1-r) + V[B+2^{\log N-i}]\cdot r$
            $V[B] = V[B]\cdot(1-g_i^{(0)}) + V[B+2^{\log N-i}]\cdot g_i^{(0)}$
        for $j = 0,\dots,N-1$ do
            $a_{it,j} = \mathsf{beta}_j \cdot [(1-j_i)\cdot(1-r) + j_i\cdot r]\cdot \tilde{V}_0(g_1^{(0)},\dots,g_{i-1}^{(0)},r,j_{i+1},\dots,j_{\log N})$    // $j_1 j_2 \dots j_{\log N}$ is the binary representation of $j$.
            $\mathsf{beta}_j = \mathsf{beta}_j \cdot [(1-j_i)\cdot(1-g_i^{(0)}) + j_i \cdot g_i^{(0)}]$
return $\{a_{1,0},\dots,a_{1,N-1},\dots,a_{\log N,0},\dots,a_{\log N,N-1}\}$;
```

Proof. **Completeness.** For each $\mathcal{V}_j$ and $\mathcal{P}$, the completeness is straightforward.

Soundness. For each j, if $\tilde{V}_0(j) \neq [C(\mathbf{in})]_j$, let $\tilde{V}_0^{\dagger}(\vec{g}^{(0)})$ be the correct value corresponding to C. If $\tilde{V}_0(\vec{g}^{(0)}) \neq \tilde{V}_0^{\dagger}(\vec{g}^{(0)})$, then $\mathcal{V}_j$ outputs 0 in Step 6 with the probability of $O(d\log|C|/|\mathbb{F}|)$ by the soundness of the GKR protocol. If $\tilde{V}_0(\vec{g}^{(0)}) = \tilde{V}_0^{\dagger}(\vec{g}^{(0)})$, $\mathcal{V}$ outputs 0 in Step 4 with probability of $O(\log N/|\mathbb{F}|)$ by the soundness of the sumcheck protocol. Thus, the total probability is bounded by $O(d\log|C|/|\mathbb{F}|)$ by the union bound.

Efficiency. For each verifier $\mathcal{V}_j$, the proof size and the verification time in Step 4 are $O(\log N)$ while the proof size and the verification time in Step 5 are $O(d\log|C|)$. If Protocol 2 employs the transparent MVPC scheme in [41], the proof size and the verification time are $O(\log^2 n)$ in Step 6. Thus the verification time and the proof size are $O(d\log|C| + \log^2 n)$ for an individual verifier. The prover runs in $O(N\log N)$ time for N verifiers in Step 4 by Algorithm 1. $\mathcal{P}$ also invokes the GKR protocol on C in Step 5. Given the same challenges, $\mathcal{P}$ costs $O(|C|)$ time in Step 5 to generate the common proof by Theorem 1. The prover time is $O(n\log n)$ for the zkMVPC scheme in [41]. Therefore, the total prover time is $O(|C| + N\log N + n\log n)$ asymptotically. □

It is not hard to see that our one-to-many zero knowledge argument scheme can be extended to support a subset of outputs per verifier in a straightforward way, and we omit the details in this paper. By instantiating the circuit C in Protocol 2 with the FFT circuit of size $|C| = O(N\log N)$ and depth $d = O(\log N)$, and the input **in** with the coefficients of the polynomial f and the N-th root of unity ω, we are able to construct a polynomial commitment scheme with prover batching. Each verifier $\mathcal{V}_j$ receives $\tilde{V}_0(\vec{j}) = f(\omega^j)$, and the prover generates all proofs in $O(N\log N)$ time. Suppose $f(x) = c_0 + c_1x + \ldots + c_tx^t$ and $t = \Theta(N)$, we have the following corollary:

Corollary 1. *For prover $\mathcal{P}$ and N verifiers $\mathcal{V}_j$ for $j \in [N]$, there exists an argument system for the function between every $\mathcal{P}$ and $\mathcal{V}_j$ that $\mathbf{out}_j = f(\omega^j)$ and $\mathbf{out} = \mathsf{FFT}(c_0,\dots,c_t)$ with soundness $O(\log^2 N/|\mathbb{F}|)$. The proof size is $O(\log^2 N)$ and the verifier time is $O(\log^2 N)$. If N verifiers have the common random challenge in every round, the total prover time is $O(N\log N)$ and communication is $O(N\log^2 N)$.*

3.2 A New Fiat-Shamir Transformation for Sharing Random Challenges

We now describe a new Fiat-Shamir-style transformation that makes Protocol 2 non-interactive in the random-oracle model, such that the N verifiers could effectively share the same random challenge in every round. A strawman approach is to use the standard Fiat-Shamir heuristic [16] for each verifier $\mathcal{V}_j$ and $\mathcal{P}$, separately. In the Fiat-Shamir heuristic [16], $\mathcal{P}$ generates $\mathcal{V}$'s random challenge by querying a random oracle on the entire transcript of messages with $\mathcal{V}$ so far. If we directly apply the Fiat-Shamir heuristic on Protocol 2 for each verifier separately, the random challenge will not be the same in every round of the protocol since each verifier $\mathcal{V}_j$ has the possibly different output $\tilde{V}_0(\vec{j})$ at the beginning of the protocol. Hence the previous transcript for $\mathcal{V}_j$ are divergent in any round, and thus the random oracle will output different challenges except with negligible probability.

Warmup: using a Merkle tree to merge random challenges into one. A better but still slightly inefficient approach is to use a Merkle tree to merge the random challenges into a single one. Precisely, in every round, the prover builds a Merkle tree on N random points generated by Fiat-Shamir-style transformation on N transcripts and uses the root as the unique challenge for all verifiers. $\mathcal{P}$ also attaches the corresponding Merkle path to convince each $\mathcal{V}_j$ of the correctness of the common randomness. Although the procedure guarantees the common random challenges, the Merkle tree approach will result in a multiplicative overhead of $O(\log N)$ on the prover time, proof size, and the verifier time of the whole protocol. Specifically, the $\log N$ blowup stems from the need to build the Merkle tree of size N and send a $\log N$-sized Merkle branch to every verifier in every round.

Our approach. We suggest a more efficient approach that achieves the same prover time as Protocol 2 and incurs only an additive overhead of $O(\log^2 N)$ on the proof size. We have the prover generate the verifier's random challenge by querying the random oracle at *only* the last round's challenge and message instead of the whole transcript. The advantage of this new heuristic approach is that all verifiers share the same challenge after Step 4 in Protocol 2 automatically without the

Merkle tree. As described in Protocol 2, as long as the random vector of $\vec{g}^{(0)}$ is identical in Step 4, all verifiers receive the same claim about $\tilde{V}_0(\vec{g}^{(0)})$ with the same random challenge. Our heuristic transformation assures that the transcript in Steps 5-6 will be the same for each verifier. Therefore, the prover only needs to insert Merkle trees in every round of Step 4 (i.e., the output layer). We provide the formal non-interactive protocol in Protocol 3.

In a general setting, the standard Fiat-Shamir transformation usually applies only to constant-round protocols (assuming only polynomial soundness loss in the security reduction). By contrast, we are applying our new Fiat-Shamir-style transformation to a non-constant-round protocol. Nonetheless, we can still prove standard polynomial soundness loss using techniques from Ben-Sasson, Chiesa, and Spooner [4]. We present the formal proof in Appendix B, which implies the theorem:

Theorem 3. *For each $\mathcal{V}_j$ and $\mathcal{P}$, Protocol 3 is a non-interactive argument system for the function $\textbf{out}_j = [C(\textbf{in})]_j$ such that $\textbf{out} = C(\textbf{in})$. The proof size is $O(d\log|C| + \log^2 n + \log^2 N)$ and the verifier time is $O(d\log|C| + \log^2 n + \log^2 N)$. The total prover time is $O(|C| + N\log N + n\log n)$.*

Efficiency. Compared to Protocol 2, the extra proof for each verifier is $\log N$ authentication paths each being of length $\log N$. Hence the proof size for each $\mathcal{V}_j$ has an extra term of $O(\log^2 N)$. The extra computation for each $\mathcal{V}_j$ is validating $\log N$ authentication paths contained in the proof by querying the random oracle $\log^2 N$ times. Thus the verification time for each verifier becomes $O(d\log|C| + \log^2 n + \log^2 N)$. The extra computation on the prover is building $\log N$ Merkle trees of size N by querying the random oracle $O(N\log N)$ times to merge the randomness in Step 4. Therefore, the total prover time is still $O(|C| + n\log n + N\log N)$ asymptotically.

Corollary 2. *For prover $\mathcal{P}$ and N verifiers $\mathcal{V}_j$ for $j \in [N]$, there exists a non-interactive argument system for the function between every $\mathcal{P}$ and $\mathcal{V}_j$ that $\textbf{out}_j = f(\omega^j)$ and $\textbf{out} = \mathsf{FFT}(c_0, \ldots, c_t)$ with the proof size of $O(\log^2 N)$ and the verification time of $O(\log^2 N)$. The prover time is $O(N\log N)$ and the total communication cost is $O(N\log^2 N)$ given $t = \Theta(N)$.*

4 KZG-Based Polynomial Commitment with Prover Batching

In this section, we propose a new scheme based on the KZG polynomial commitment with prover batching, such that generating all proofs only takes $O(N\log N)$ time, without introducing any overhead on the proof size and the verifier time. We first present the formal algorithms of the original KZG polynomial commitment and then introduce our new scheme.

4.1 KZG Polynomial Commitment

The KZG polynomial commitment relies on the bilinear map, which is defined below.

Bilinear map. Let $\mathbb{G}, \mathbb{G}_T$ be two groups of prime order p and let $g \in \mathbb{G}$ be a generator. $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ denotes a bilinear map and we use $\mathsf{bp} = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BilGen}(1^\lambda)$ for the generation of parameters for the bilinear map.

The KZG polynomial commitment is as follows.

- $\mathsf{pp} \leftarrow \mathsf{KeyGen}(1^\lambda, t)$: Given the security parameter and a bound on the degree of the polynomial, it runs $(p, g, \mathbb{G}, e, \mathbb{G}_T) \leftarrow \mathsf{BilGen}(1^\lambda)$. Output $\mathsf{pp} = [p, g, \mathbb{G}, e, \mathbb{G}_T, \{g^{\tau^0}, g^{\tau^1}, \ldots, g^{\tau^t}\}]$.
- $\mathsf{com}_f \leftarrow \mathsf{Commit}\ (f, \mathsf{pp})$: Given a polynomial $f(x) = \sum_{i=0}^{t} c_i x^i$, it computes $\mathsf{com}_f = g^{f(\tau)} = \Pi_{i=0}^{t}(g^{\tau^i})^{c_i}$.
- $\{(y, \pi)\} \leftarrow \mathsf{Open}\ (f, a, \mathsf{pp})$: For an evaluation point a, the prover computes $y = f(a)$ and polynomial $q(x) = \frac{f(x)-y}{x-a}$. Let the coefficients of q be $(q_0, q_1, \ldots, q_{t-1})$. The prover computes $\pi = g^{q(\tau)} = \Pi_{i=0}^{t-1}(g^{\tau^i})^{q_i}$.
- $\{1, 0\} \leftarrow \mathsf{Verify}\ (\mathsf{com}_f, a, y, \pi, \mathsf{pp})$: Given the commitment com_f, the evaluation point a, the answer y and the proof π, the verifier checks if $e(\mathsf{com}/g^y, g) \stackrel{?}{=} e(\pi, g^\tau/g^a)$. It outputs $\mathsf{1}$ if the check passes, and $\mathsf{0}$ otherwise.

The scheme is computationally-hiding under the discrete log assumption and computationally binding under the l-SBDH assumption. The prover time of the KZG commitment is $O(t)$ modular exponentiations, the proof size is $O(1)$, a single element in the base group, and the verifier time is $O(1)$, one bilinear pairing.

FFT on group elements. As the FFT algorithm only involves additions and scalar multiplications with the powers of the root of unity ω, the algorithm can be applied to a vector of elements in the base group of the bilinear map by replacing the additions with multiplications and the multiplications with exponentiations in the base group. In particular, let $A = (a_0, \ldots, a_N)$ and $g^A = (g^{a_0}, \ldots, g^{a_N})$, one can evaluate $\mathsf{FFT}(g^A) = (g^{f(\omega^0)}, \ldots, g^{f(\omega^{N-1})})$ for $f(x) = \sum_{i=0}^{N} a_i x^i$ in time $O(N\log N)$, without knowing $(a_0, \ldots, a_N)$. Similarly, one can also compute the convolution with a public vector $B = (b_0, \ldots, b_N)$ "on the exponent", i.e., $g^{A*B} = \mathsf{IFFT}(\mathsf{FFT}(g^A) \odot \mathsf{FFT}(B))$, where $\odot$ denotes element-wise exponentiation.

4.2 Our New Prover Batching Technique

In our new scheme, the public parameters pp, the commitment com_f and the proof π together with $\mathsf{Keygen}, \mathsf{Commit}, \mathsf{Verify}$ are exactly the same as the KZG commitment. The main contribution is that we present a new batched algorithm for the

Protocol 3. Making Protocol 2 non-interactive with a new Fiat-Shamir-style tranformation
*Let λ be the security parameter. Let C: $\mathbb{F}^n \to \mathbb{F}^N$ be a d-depth layered arithmetic circuit. For any $j \in [N]$, $\mathcal{P}$ needs to convince $\mathcal{V}_j$ that $\boldsymbol{out}_j = [C(\boldsymbol{in})]_j$ where $[C(\boldsymbol{in})]_j$ is the j-th output of the circuit given input **in**, and $\boldsymbol{out}_j$ is the the claimed result for $\mathcal{V}_j$. Without loss of generality, assume n and N are both powers of 2 and we can pad them if not. Let ρ be a random oracle.*

1. *Set* $\mathsf{pp} \leftarrow \mathsf{MVPC.KeyGen}(1^\lambda)$.
2. *$\mathcal{P}$ invokes* $\mathsf{MVPC.Commit}(\tilde{V}_d, \mathsf{pp})$ *to generate* $\mathsf{com}_{\tilde{V}_d}$ *and broadcasts* $\mathsf{com}_{\tilde{V}_d}$ *to all verifiers. $\tilde{V}_d$ is the multilinear extension of **in**, as defined in the GKR protocol.*
3. *For each $j \in [N]$, $\mathcal{P}$ sends $V_0(j)$ as $\boldsymbol{out}_j$ to $\mathcal{V}_j$ separately.*
4. *For each $j \in [N]$, $\mathcal{P}$ runs the sumcheck protocol on the equation in Step 4 of Protocol 2. For $i = 1,\dots,\log N$:*
   - *(a) Suppose $M_{i,j}$ is the i-th univariate polynomial $\mathcal{P}$ sends to $\mathcal{V}_j$ in the sumcheck. If $i = 1$, set $r_{i,j} = \rho(\mathsf{com}_{\tilde{V}_d}||V_0(j)||M_{1,j})$. If $i > 1$, set $r_{i,j} = \rho(g^{(0)}_{i-1}||M_{i,j})$.*
   - *(b) $\mathcal{P}$ builds a Merkle tree on the vector of $\vec{r}^{(i)} = (r_{i,0},\dots,r_{i,N-1})$. Let $g^{(0)}_i = \mathsf{MT.Commit}(\vec{r}^{(i)})$. Then $\mathcal{P}$ assigns $g^{(0)}_i$ as the common random challenge in the i-th round.*
   - *(c) $\mathcal{P}$ attaches $(r_{i,j}, \mathsf{path}_{i,j}) \leftarrow \mathsf{MT.Open}(j, g^{(0)}_i)$ in the proof.*

   In the last round of the sumcheck, $\mathcal{P}$ sends $\tilde{V}_0(\vec{g}^{(0)})$ to each $\mathcal{V}_j$ as they share the same random vector of $\vec{g}^{(0)}$.
5. *$\mathcal{P}$ invokes the GKR protocol with $\tilde{V}_0(\vec{g}^{(0)})$. In each round, $\mathcal{P}$ generates the random challenges by querying ρ on the last round's challenge and message. For all V_j, the random challenges and the transcript would be exactly the same because they share the same claim about $\tilde{V}_0(\vec{g}^{(0)})$ and the same random vector $\vec{g}^{(0)}$ from the first round of this step.*
6. *In the last round of the GKR protocol, all verifiers have the same claim about $\tilde{V}_d(\vec{g}^{(d)})$. $\mathcal{P}$ invokes* $\mathsf{zkMVPC.Open}(\tilde{V}_d, \vec{g}^{(d)}, \mathsf{pp})$ *to generate the proof for the claim.*
7. *For each $j \in [N]$, $\mathcal{V}_j$ checks the proof with random challenges provided by $\mathcal{P}$ and* $\mathsf{zkMVPC.Verify}$. *Then $\mathcal{V}_j$ checks all authenticated paths in the Merkle tree proof by* $\mathsf{MT.Verify}$. *In particular, given $\mathsf{path}_{i,j} = (\nu_1,\dots,\nu_{\log N})$, for $k = 1,\dots,\log N$: if $j_i = 0$, $\mathcal{V}_j$ computes $r_{i,j} = \rho(r_{i,j}||\nu_i||(i-1)(logN+1)+k)$; otherwise $\mathcal{V}_j$ computes $r_{i,j} = \rho(\nu_i||r_{i,j}||(i-1)(logN+1)+k)$. $\mathcal{V}_j$ checks that $r_{i,j} = g^{(0)}_i$. Finally, $\mathcal{V}_j$ queries ρ to check the generation process of random challenges.*

prover to generate proofs for N different evaluation points. The key idea of our scheme is to evaluate the polynomials at different powers of the N-th root of unity ω, which enables us to invoke the FFT algorithm to compute the proofs efficiently — but observing how to leverage the FFT algorithm is non-trivial. Recall that in the dealing round of the VSS scheme, for each party $i \in [N]$ the dealer computes $s_i = f(u_i)$ and $\pi_i = g^{q_i(\tau)} = g^{\frac{f(\tau)-f(u_i)}{\tau-u_i}}$. By setting the public evaluation point of party i as $u_i = \omega^i$, the dealer can compute all s_i in $O(N \log N)$ time using the FFT algorithm. However, computing the proofs π_i is more challenging, as τ is the secret key and is not explicitly given to the dealer. The dealer only has access to the public parameters $g^\tau, g^{\tau^2}, \dots, g^{\tau^t}$.

To solve this, we examine the structure of the polynomials $q_i(x)$ for $i \in [N]$. We define a bivariate polynomial $q(x,y)$ as

$$q(x,y) = \frac{f(x)-f(y)}{x-y}. \tag{2}$$

Then, $q_i(\tau) = q(\tau, \omega^i)$ and the proofs are $\pi_i = g^{q(\tau,y)}$ for $y = \omega^i, i \in [N]$. Let $f(x) = \sum_{j=0}^t c_j x^j$, we have:

$$q(\tau,y) = \frac{f(\tau)-f(y)}{\tau-y} = \frac{\sum_{j=0}^t c_j(\tau^j - y^j)}{(\tau-y)} = \sum_{j=1}^t c_j \sum_{k=1}^j y^{k-1}\tau^{j-k}, \tag{3}$$

as $\tau^j - y^j = (\tau-y)\cdot\sum_{k=1}^j y^{k-1}\tau^{j-k}$ for $j = 1,\dots,N$, and the constant term c_0 cancels out for $j = 0$. By changing the order of the summations, the equation above equals to

$$\sum_{k=1}^t y^{k-1} \sum_{j=k}^t c_j \tau^{j-k} = \sum_{k=1}^t h_k y^{k-1}, \tag{4}$$

where $h_k = \sum_{j=k}^t c_j \tau^{j-k}$. As shown by the equations above, $q(\tau,y)$ is a degree-$(t-1)$ polynomial of variable y. If we can precompute all g^{h_k} for $k = 1\dots,t$, we can evaluate $g^{q(\tau,y)}$ at $y = \omega^i$ for $i \in [N]$ in $O(N \log N)$ time using the FFT algorithm on the elements in the base group.

Precomputing g^{h_k}. We observe that h_k is in the form of a convolution, and we can precompute all g^{h_k} using FFT. Let

```
Algorithm 2 $(\pi_0,\ldots,\pi_{N-1}) \leftarrow \mathsf{multi\_proof}(f,\omega,N,\mathsf{pp})$
Input: Polynomial $f(x)=\sum_{i=0}^{t} c_i x^i$, the number of parties $N$, the $N$-th root of unity $\omega$ and the public parameter $\mathsf{pp}$ containing $g^{\tau}, g^{\tau^2},\ldots,g^{\tau^t}$ and $(p,g,e,\mathbb{G},\mathbb{G}_T)$.
Output: Proofs of the KZG commitment $\pi_i = g^{q_i(\tau)}$, for $i \in [N]$.
  Set $C=(c_1,c_2,...,c_t), g^T=(g^{\tau^{t-1}},g^{\tau^{t-2}},...,g^{\tau^0})$.
  Compute the convolution $g^H = g^{C*T} = \mathsf{IFFT}(\mathsf{FFT}(g^T)\odot\mathsf{FFT}(C))$, where $\odot$ denotes element-wise exponentiation.
  $g^{h_k} = g^{\sum_{j=k}^{t} c_j \tau^{j-k}} = g^{H[k+t-2]}$ for each $k=1,\ldots,t$.
  For polynomial $h(y)=\sum_{k=1}^{t} h_k y^{k-1}$, compute $(g^{h(\omega^0)},g^{h(\omega^1)},\ldots,g^{h(\omega^{N-1})}) = \mathsf{FFT}(g^{h_1},g^{h_2},\ldots,g^{h_t})$.
  Return $\pi_i = g^{q_i(\tau)} = g^{h(\omega^i)}$
```

$C=(c_1,c_2,...,c_t), T=(\tau^{t-1},\tau^{t-2},...,\tau^0)$, and let $H=C*T$ be their convolution. As described in Section 2, we have:

$$H[\ell]=\sum_{m=0}^{\ell} C[m]T[\ell-m]=\sum_{m=0}^{\ell} c_{m+1}\tau^{t-1-(\ell-m)}. \qquad (5)$$

By setting $\ell=k+t-2$ and $j=m+1$, $c_{m+1}=c_j$ and $\tau^{t-1-(\ell-m)}=\tau^{j-k}$ in Equation 5. Moreover, c_j is defined to be nonzero for $j\in[1,t]$, and τ^{j-k} is defined to be nonzero for $j\in[k,k+t-1]$. Therefore, $H[\ell]=\sum_{j=1}^{k+t-1} c_j\tau^{j-k}=\sum_{j=k}^{t} c_j\tau^{j-k}=h_k$. Thus, the dealer can precompute all g^{h_k} for $k=1,\ldots,t$ using FFT and IFFT on g^T and C in $O(t\log t)$ time without knowing τ.

Complexity analysis. We present the formal algorithm in Algorithm 2. Combining the two steps, the overall complexity of the dealer is $O(N\log N)$ modular exponentiations. The proof size and the verifier time remain $O(1)$ per evaluation. In fact, our scheme is a more efficient algorithm to generate multiple proofs, and each proof size and verification time remain exactly the same as the original KZG polynomial commitment. The security of our scheme follows directly from the security proofs in [25].

Note that both our new scheme and the original KZG scheme only achieve computationally-hiding and binding, but not the stronger notion of proof of knowledge and zero knowledge in Definition 1. However, they suffice to prove security for the application of VSS and DKG as shown in [25], as the secret and the polynomial are randomly generated. Follow-up works such as [43] propose variants that achieve proof of knowledge and zero knowledge using randomized commitment and opening, and knowledge assumptions. Our scheme with prover batching also works on these variants with minimal changes and achieves stronger notions. We sketch the algorithms in the full version.

5 Implementation and Evaluation

We fully implemented our proposed schemes with prover batching and present the experimental results in this section.

Implementation. We implemented our proposed schemes in C++ consisting of around 3000 lines of code. We used the ate-pairing library [1] for bilinear maps in the scheme with the trusted setup, and the GMP library [2] for large numbers and arithmetic on a finite field. The implementation of our transparent polynomial commitment was based on the open-source codebase of the scheme in [41]. We used the same extension field $\mathbb{F}_{p^2}$ for $p=2^{61}-1$, which provides 100+ bits of security.

Configuration. We ran the experiments on an AWS c5a.24xlarge instance, which was equipped with an AMD EYPC 7002 CPU with 96 cores, 187 GB RAM[4]. All parties were executed on the same machine. We only report the numbers for the optimistic case of the VSS and DKG schemes where all the proofs are generated honestly. The polynomial commitment takes the majority of the time in this case, which is the main focus of this paper. In all the 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}.

Counterpart comparison. In the VSS setting, we compare our KZG-based polynomial commitment scheme with two schemes that also require a trusted setup (Section 5.1): *(i)* naïvely running the KZG polynomial commitment for N verifiers, which incurs a prover time of $O(N^2)$; and *(ii)* the authenticated multipoint evaluation tree (AMT) scheme in [35]. We executed the open-source code of [35] on the same machine for a fair comparison. We then compare our transparent polynomial commitment scheme with running a transparent counterpart, named Virgo [41], N times to produce N proofs (Section 5.2). Finally, we evaluate the performance of our KZG-based and transparent schemes under DKG application, compared with AMT-DKG instantiation [35] (Section 5.3).

5.1 VSS with Trusted Setup

As shown in Figure 1, the running time of the dealer in our KZG-based scheme only grows quasi-linearly with the number of parties. It only takes 2.2s to generate the proofs for 2^{11} parties and takes 3,995s for 2^{21} parties. This is significantly faster than running the KZG commitment naïvely and the speedup is 100–58,000×. We could not run the naïve scheme beyond $N=2^{12}$ due to its long-running time. Therefore, we ran up to 2^{12} parties and extrapolated the result for the larger number of parties. Comparing to the AMT scheme [35], the prover time of our scheme is slightly worse. It is 2.2× slower than AMT for $N=2^{11}$ and 3× slower for $N=2^{21}$. This is because our scheme involves 3 FFTs on the base group of the bilinear map, and the constant in our asymptotic complexity is slightly larger than that in AMT.

[4] Our KZG-based scheme only takes 2.8GB of memory in the largest instance. In our transparent scheme, the memory usage can be reduced to several gigabytes with proper pipelining by streaming the proof to each verifier without affecting the prover time.

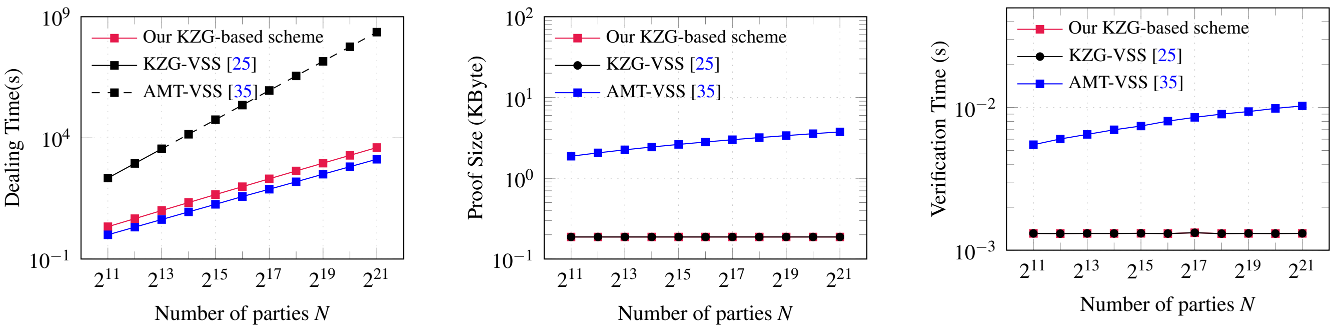

Figure 1: VSS Comparison, Trusted setup version

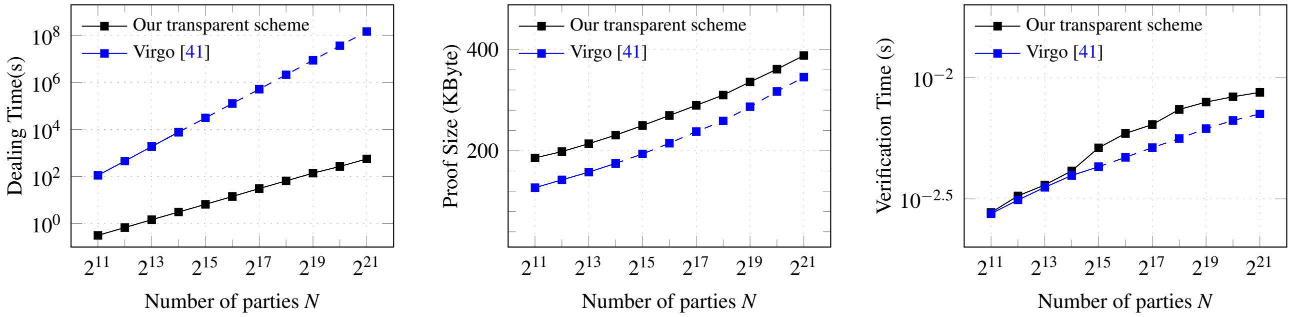

Figure 2: VSS Comparison, Transparent setup version

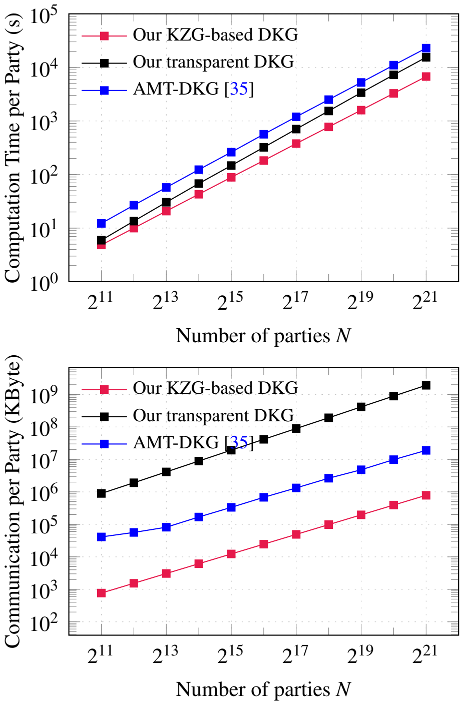

Figure 3: DKG Comparison

The proof size and the verifier time in our scheme are much smaller than AMT. They are always 192 bytes and 1.3ms *regardless of the number of parties*, which are the same as the original KZG scheme. By contrast, the proof size and the verifier time grow logarithmically in AMT. Specifically, the proof size is 20× larger than our scheme, and the verifier time is 4.2-7.8× slower. This shows that our schemes achieve much higher scalability than the state-of-the-art.

5.2 VSS with Transparent Setup

Figure 2 presents the performance of the VSS scheme with our scheme compared with Virgo. As shown in the figure, the dealing time of our transparent scheme is very fast. It only takes 0.3s to generate proofs for 2^{11} parties and 560s for 2^{21} parties. This is 700-260,000× faster than the naïve approach, which again takes a quadratic time and does not scale in practice. One may observe an interesting result that the dealing time of our transparent scheme is indeed an order of magnitude (i.e., 7-10.5×) faster than the schemes with trusted setup in Section 5.1. This is because our transparent scheme only incurs cheap symmetric-key operations such as hashing and field arithmetic instead of the costly modular exponentiation. This performance gain is significant even though it is generally not captured in the asymptotic cost.

The proof size and the verifier time of our transparent scheme are comparable to Virgo, as we merely introduce an additional sumcheck for each verifier. The proof size varies from 200KiB to 390KiB, and the verifier time varies from 2.8ms to 8.7ms. The proof size is larger than the KZG-based schemes because of the underlying techniques of interactive proofs. However, notice that our transparent scheme removes the trusted setup, which is critical in some applications.

5.3 Distributed Key Generation Experiment

We report the total computation time and communication for each party of the DKG schemes using our polynomial commitments in Figure 3, and compare it with AMT-DKG [35].

The overall computation time of our protocols grows quasilinearly with the number of parties. For example, it takes $15,400$s for our transparent scheme to run a DKG of 2^{21} participants, and $6,700$s for our KZG-based scheme. These are 1.5× and 3.3× faster than the AMT scheme respectively. This is because our transparent scheme only incurs cheap symmetric operations, as discussed above, despite being asymptotically logarithmically slower than AMT. On the other hand, our KZG-based scheme incurs a lower verification time for each party to verify the proofs from the other parties. Moreover, our transparent scheme is slower than our KZG-based scheme in the application of DKG. This is because although the prover time of our transparent scheme is faster, its verifier time is slower ($O(\log^2 N)$ vs. $O(1)$). In DKG, each party verifies the proof of every other party, which becomes the bottleneck of our transparent scheme.

The total communication of our KZG-based scheme is orders of magnitude smaller than AMT. Specifically, it is always $192 \cdot N$ bytes in our scheme, while the proof size of AMT-DKG grows quasilinearly. Concretely our KZG-based scheme achieves the communication of only 0.8GB for $N = 2^{21}$, which is 20× smaller than AMT. Due to techniques to remove the trusted setup, the communication in our transparent scheme is 100× larger than AMT, which matches their asymptotic cost difference (i.e., $O(N \log^2 N)$ vs. $O(N \log N)$).

Acknowledgments

Elaine Shi and Yupeng Zhang are supported by DARPA under Contract No. HR001120C0087. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA.

References

[1] Ate-pairing. `https://github.com/herumi/ate-pairing`.

[2] The GNU multiple precision arithmetic library. `https://gmplib.org/`.

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, 1988.

[4] E. Ben-Sasson, A. Chiesa, and N. Spooner. Interactive oracle proofs. In *TCC*. Springer, 2016.

[5] G. R. Blakley. Safeguarding cryptographic keys. In *Managing Requirements Knowledge, International Workshop on*. IEEE Computer Society, 1979.

[6] E. Boyle and R. Pass. Limits of extractability assumptions with distributional auxiliary input. In *ASIACRYPT 2015*, pages 236–261.

[7] B. Bünz, B. Fisch, and A. Szepieniec. Transparent snarks from dark compilers. Cryptology ePrint Archive, Report 2019/1229, 2019.

[8] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive security for threshold cryptosystems. In *CRYPTO*, 1999.

[9] D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *STOC*, 1988.

[10] A. Chiesa, M. A. Forbes, and N. Spooner. A Zero Knowledge Sumcheck and its Applications. *CoRR*, abs/1704.02086, 2017.

[11] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *FOCS*, 1985.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[13] G. Cormode, M. Mitzenmacher, and J. Thaler. Practical Verified Computation with Streaming Interactive Proofs. In *ITCS*, 2012.

[14] S. Das, Z. Xiang, and L. Ren. Asynchronous data dissemination and its applications. Cryptology ePrint Archive, Report 2021/777.

[15] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, 1987.

[16] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Crypto 1986*.

[17] Y. Gao, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang. Efficient asynchronous byzantine agreement without private setups. Cryptology ePrint Archive, Report 2021/810.

[18] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Eurocrypt*, 1999.

[19] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating Computation: Interactive Proofs for Muggles. *J. ACM*, 62(4):27:1–27:64, Sept. 2015.

[20] V. Goyal, A. Kothapalli, E. Masserova, B. Parno, and Y. Song. Storing and retrieving secrets on a blockchain. Cryptology ePrint Archive, Report 2020/504, 2020.

[21] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. Aggregatable distributed key generation. 2021.

[22] I. Ingemarsson and G. J. Simmons. A protocol to set up shared secret schemes without the assistance of a mutually trusted party. In *Workshop on the Theory and Application of of Cryptographic Techniques*, 1990.

[23] A. Kate. Distributed key generation and its applications. 2010.

[24] A. Kate and I. Goldberg. Distributed key generation for the internet. In *ICDCS*, 2009.

[25] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT 2010*, pages 177–194.

[26] E. Kokoris Kogias, D. Malkhi, and A. Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In *CCS*, 2020.

[27] J. Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. Cryptology ePrint Archive, Report 2020/1274, 2020.

[28] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic Methods for Interactive Proof Systems. *J. ACM*, 39(4):859–868, Oct. 1992.

[29] R. C. Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, 1987.

[30] W. Neji, K. Blibech, and N. Ben Rajeb. Distributed key generation protocol with a new complaint management strategy. *Security and communication networks*, 9(17):4585–4595, 2016.

[31] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1991.

[32] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *STOC*, 1989.

[33] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[34] J. Thaler. Time-Optimal Interactive Proofs for Circuit Evaluation. In R. Canetti and J. A. Garay, editors, *CRYPTO*, 2013.

[35] A. Tomescu, R. Chen, Y. Zheng, I. Abraham, B. Pinkas, G. G. Gueta, and S. Devadas. Towards scalable threshold cryptosystems. In *S & P*, 2020.

[36] A. Vlasov and K. Panarin. Transparent polynomial commitment scheme with polylogarithmic communication complexity. *IACR Cryptol. ePrint Arch.*, 2019.

[37] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. Doubly-efficient zkSNARKs without trusted setup. In *S & P*, 2018.

[38] C. J. Weinstein. Quantization effects in digital filters. Technical report, MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB, 1969.

[39] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *CRYPTO*, 2019.

[40] T. Yurek, L. Luo, J. Fairoze, A. Kate, and A. K. Miller. hbacss: How to robustly share many secrets. *IACR Cryptol. ePrint Arch.*, 2021, 2021.

[41] J. Zhang, T. Xie, Y. Zhang, and D. Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *S&P 2020*.

[42] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *S& P*, 2017.

[43] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. A Zero-Knowledge version of vSQL. Cryptology ePrint, 2017.

A Additional Preliminaries

Interactive proofs. An interactive proof allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ the validity of some statement through several rounds of interaction. We say that an interactive proof is public coin if $\mathcal{V}$'s challenge in each round is independent of $\mathcal{P}$'s messages in the previous rounds. The proof system is interesting when the running time of $\mathcal{V}$ is less than the time of directly computing the function F. We formalize the interactive proofs in the following:

Definition 4. *Let F be a function. A pair of interactive machines $\langle \mathcal{P}, \mathcal{V} \rangle$ is an interactive proof for $F(x) = y$ with negligible soundness if the following holds:*

- **Completeness.** *For $F(x) = y$ it holds that* $\Pr[\langle \mathcal{P}, \mathcal{V} \rangle(x,y) = 1] = 1$.
- **Soundness.** *For $F(x) \neq y$ and any $\mathcal{P}^*$ it holds that* $\Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle(x,y) = 1] \leq \mathsf{negl}(\lambda)$.

Verifiable Secret Sharing. An $(N, t+1)$ secret sharing scheme [5, 33] allows a dealer to split up a secret s among N verifiers in such a way that only the subset of $t+1$ or more

Protocol 4. $(N, t+1)$ **verifiable secret sharing scheme**
Suppose $\mathcal{P}$ is the dealer with a secret $s \in \mathbb{F}$ and $\mathcal{V}_0, \ldots, \mathcal{V}_{N-1}$ are N verifiers. Let $\mathsf{pp} \leftarrow \mathsf{KeyGen}(1^\lambda, t)$.

- ***Sh phase:***
  1. *$\mathcal{P}$ picks $f \in_R \mathbb{F}[X]$ of degree t such that $s = f(0)$, computes $s_j = f(u_j)$ for all $j \in [N]$.*
  2. *$\mathcal{P}$ runs $\mathsf{com}_f = \mathsf{Commit}(f, \mathsf{pp})$ and broadcasts com_f to all verifiers.*
  3. *$\mathcal{P}$ runs $(f(u_j), \pi_j) = \mathsf{Open}(f, u_j, \mathsf{pp})$ and sends $(f(u_j), \pi_j)$ to $\mathcal{V}_j$ for all $j \in [N]$.*
  4. *For each $j \in [N]$, $\mathcal{V}_j$ invokes $b \leftarrow \mathsf{Verify}(\mathsf{com}_f, u_j, f(u_j), \pi_j, \mathsf{pp})$. If $b = 0$, $\mathcal{V}_j$ broadcasts a complaint against the dealer.*
  5. *If the size of the set S of complaining players is larger than t, the dealer is disqualified. Otherwise, the dealer reveals the correct shares with proofs by broadcasting $\{f(u_j, \pi_j)\}_{j \in S}$. If any one proof does not verify (or dealer did not broadcast), the dealer is disqualified. Otherwise, each V_j now has her correct share $f(u_j)$.*
- ***Rec phase:*** *Given com_f and shares $(f(u_j), \pi_i)_{i \in T \subseteq [N]}$ such that $|T| > t$, the reconstructor runs $b_j \leftarrow \mathsf{Verify}(\mathsf{com}_f, u_j, f(u_j), \pi_j, \mathsf{pp})$ for all $j \in T$. If $b_j = 1$ for all $j \in T$, the reconstructor recovers f with $\{f(u_j)\}_{j \in T}$ by Lagrange interpolation and obtains $s = f(0)$.*

participants can recover the secret s, and the subset of t or fewer participants can not. An $(N, t+1)$ VSS scheme can be instantiated with a polynomial commitment scheme. It consists of two phases: the sharing (Sh) phase and the reconstruction (Rec) phase. In the sharing phase, the dealer $\mathcal{P}$ picks a random polynomial $f(x)$ of degree t such that the secret $s = f(0)$, and then commits to f. Then $\mathcal{P}$ sends the shared secret $s_j = f(u_j)$ and the corresponding proof π_j to the verifier $\mathcal{V}_j$ for all $j \in [N]$, where each u_j is a unique value. $\mathcal{V}_j$ accepts s_j by checking π_j. In the reconstruction phase, each verifier $\mathcal{V}_j$ reveals s_j and π_j to the reconstructor. The reconstructor uses Lagrange interpolation to recover s after receiving $t+1$ valid shares. The formal VSS protocol instantiated with the polynomial commitment is presented in Protocol 4.

B Proving Soundness of Our New Fiat-Shamir-Style Transformation

Below, we focus on Steps 3-6 of Protocol 3 on the circuit C and formally prove that the protocol is complete and sound. When combined with the MVPC in Steps 1,2 and 7, we are able to obtain a non-interactive argument by Definition of Zero-knowledge proofs.

There are two key differences between our transformation in Protocol 3 and the standard Fiat-Shamir transformation: (1) in every round, the randomness is generated by hashing the random challenge and the message in the previous round, instead of the entire transcript so far; (2) in Step 4 a Merkle tree is constructed on the hash of each verifier and the root is used as the common randomness in this round for all verifiers. However, when viewed from the perspective of a single verifier, e.g., $\mathcal{V}_0$, the second difference is actually very similar to the first one. This is because in each round, $\mathcal{V}_0$ receives $\log N$ messages from the prover (claimed to be the validation path of a Merkle tree). To perform the Merkle tree verification, as shown in Step 6 of Protocol 3, suppose $\mathcal{P}$ sends the authentication path of $(\nu_1, \ldots, \nu_{\log N})$ to $\mathcal{V}_0$ and r is $\mathcal{V}_0$'s random challenge in the current round, $\mathcal{V}_0$ computes $\mathsf{rt} = \rho(\ldots\rho(\rho(r||\nu_1)||\nu_2)\ldots||\nu_{\log N})$ and use rt as her random challenge in the next round. When proving soundness, as the prover is malicious, there is no guarantee that these messages are indeed from the same Merkle tree among all verifiers. Therefore, it is equivalent to extending one round of the interactive version of the sumcheck protocol in Step 4 of Protocol 2 to $\log N$ rounds. In these additional rounds, the prover does nothing but sending a dummy message ν_i to the verifier. The verifier ignores the dummy message and replies with fresh randomness. Finally, the prover and the verifier use the last randomness to proceed to the next round of the original sumcheck protocol. We give this interactive protocol with dummy messages for Step 4 of Protocol 2 in Protocol 5.

Observe that when we apply our Fiat-Shamir transformation of hashing only the previous random challenge and message to Protocol 5, it becomes Protocol 3 from the perspective of each verifier, if we model the hash function in the Merkle tree as the random oracle. Moreover, intuitively Protocol 5 is sound as long as the original interactive proof is sound, as the verifier simply picks some additional randomness and ignores some dummy messages from the prover. Therefore, our strategy to prove the soundness of Protocol 3 is: (1) we first show that as long as an interactive proof protocol is secure against state restoration attacks defined in [4], the non-interactive protocol by applying the transformation of hashing only the previous message and the random challenge is sound; (2) extending one round of an interactive proof protocol to multiple rounds with dummy messages as in Protocol 5 does not affect the security against state restoration attacks.

Formally, let T be an interactive proof protocol with η rounds for the statement of $F(x) = y$. Let $(m_1, r_1, \ldots, m_\eta, r_\eta)$ denote a complete transcript for T, where $m_i \in \mathbb{F}^*$ is the prover's message in round i while $r_i \in \mathbb{F}$ is the verifier's randomness in round i. Let $\rho : \mathbb{F}^* \to \mathbb{F}$ denote the random oracle. In our new heuristic algorithm, the prover sets required random points as $r_1 = \rho(x||y||m_1||0)$ and $r_i = \rho(r_{i-1}||m_i||i-1)$ or $r_i = \rho(m_i||r_{i-1}||i-1)$ for $i \in \{2, \ldots, \eta\}$ to make T non-interactive. We modify the transformation by taking the round number in T as the extra input to the random oracle and exchanging the order of r_{i-1} and m_i in certain rounds. We show that if T is sound against the prover with state restoration attacks, then the non-interactive protocol is sound after the heuristic transformation.

Protocol 5. The interactive proof with dummy messages for Step 4 of Protocol 2

For round $i = 1 : \dots, \log N$ of the sumcheck protocol:
For $k = 1, \dots, \log N$:

- *$\mathcal{P}$ sends $\mathsf{v}_{i,k}$ to $\mathcal{V}$.*
- *$\mathcal{V}$ responds with random number of $\mathsf{Du}_{i,k} \in \mathbb{F}$.*
- *If $j_k = 0$, set $r_i^{(0)} = \rho(r_i^{(0)}||\mathsf{v}_{i,k})$; otherwise, set $r_i^{(0)} = \rho(\mathsf{v}_{i,k}||r_i^{(0)})$.*

$\mathcal{P}$ and $\mathcal{V}_j$ use $r_i^{(0)}$ as the random challenge in round i and continue the sumcheck protocol.

Game 1. The game between a state-restoring prover $\mathcal{P}^*(x,y)$ and a verifier $\mathcal{V}(x,y)$.

1. *Given x,y satisfying $F(x) \neq y$, the game initializes the set of* SeenStates *to be* $\{\mathsf{null}\}$.
2. *Repeat the following at most T times, where $T = \mathsf{poly}(\lambda)$:*
   (a) *$\mathcal{P}^*$ chooses an element* cvs *in* SeenStates. *(*cvs *is short for complete verifier's state.)*
   (b) *The game sets $\mathcal{V}$'s state to* cvs.
   (c) *If* $\mathsf{cvs} = \mathsf{null}$*: $\mathcal{P}^*$ sends m_1 to $\mathcal{V}$. Then $\mathcal{V}$ returns a random point r_1 to $\mathcal{P}^*$; $\mathcal{P}^*$ adds $m_1||r_1$ into* SeenStates.
   (d) *If* $\mathsf{cvs} = m_1||r_1||\dots||m_{i-1}||r_{i-1}$ *for $1 < i \leq \eta$: $\mathcal{P}^*$ sets $\mathcal{V}$'s state to* cvs, *generates m_i according to* cvs *and sends it to $\mathcal{V}$. After receiving r_i randomly sampled by $\mathcal{V}$. $\mathcal{P}^*$ adds* $\mathsf{cvs}||m_i||r_i$ *into* SeenStates.
   (e) *If* $\mathsf{cvs} = m_1||r_1||\dots||m_\eta||r_\eta$, *the prover can choose to set $\mathcal{V}$'s state to* cvs. *$\mathcal{V}$ computes his decision b given $(x,y,m_1,r_1,\dots,m_\eta,r_\eta)$. Then, the game halts and outputs b.*
3. *The game halts and outputs* 0.

Definition 5. *An interactive protocol* T *for the statement $F(x') = y'$ with η rounds is secure against state restoration attacks if for every x and y such that $F(x) \neq y$, for every $\mathcal{P}^*$ and an honest $\mathcal{V}$, Game 1 outputs* 1 *with the probability of* $\mathsf{negl}(\lambda)$.

Theorem 4. *If* T *is an interactive proof for $F(x) = y$ with η rounds and it is secure against state restoration attacks in Definition 5, then after the transformation, the non-interactive protocol* T$'$ *satisfies the soundness in interactive proofs.*

Proof. We use $\mathbb{P}^\rho$ and $\mathbb{V}$ to represent the prover and the verifier in the non-interactive protocol separately. Suppose $\mathbb{P}$ can query a random oracle ρ at most Δ times. Given $\Delta = \mathsf{poly}(\lambda)$, we construct a prover $\mathcal{P}^*$ with state restoration attack ability against verifier $\mathcal{V}$ in the original interactive protocol.

Construction of $\mathcal{P}^*$. We use $\mathcal{P}^*$ to simulate the random oracle for $\mathbb{P}^\rho$ and $\mathcal{P}^*$ works as follows.

1. Let ρ be a table mapping $\mathbb{F}^* \to \mathbb{F}$ and let δ be a table mapping a random point in $\mathbb{F}$ to the verifier's state. Both tables are empty in the beginning and are filled with elements as $\mathcal{P}^*$ runs the protocol. Intuitively, we use ρ to simulate $\mathbb{P}^\rho$ access to a random oracle while we use δ to keep track of $\mathcal{V}$'s states that $\mathcal{P}^*$ has "seen in his mind". Given a verifier's state cvs, let $L(\mathsf{cvs})$ be the number of rounds contained in the state, which can be simply treated as the number of $||$ in cvs by the format of cvs. Suppose the vector $\vec{e} = (e_1, \dots, e_{\eta-1}) \in \{0,1\}^{\eta-1}$ is public.
2. Begin simulating $\mathbb{P}^\rho$ and, for $i = 1, \dots, \Delta$:
   (a) Let θ_i denote the i-th query by $\mathbb{P}^\rho$.
   (b) If θ_i has been inserted into the table ρ, $\mathcal{P}^*$ responds with $\rho(\theta_i)$. Go to next iteration for i.
   (c) If $i < j$, $\mathcal{P}^*$ draws a random number $r \in \mathbb{F}$, answers the query with r, then sets $\rho(\theta_i) := r$. Go to next iteration for i.
   (d) If $i = j$, $\mathcal{P}^*$ splits θ_i to $x||y||m_1||0$ ($\mathcal{P}^*$ aborts if he cannot split θ_i to $x||y||m_1||0$). $\mathcal{P}^*$ starts the game with $\mathcal{V}$ on $f(x) = y$. $\mathcal{P}^*$ sets $\mathcal{V}$'s state to (null), sends m_1 to $\mathcal{V}$, receives the first randomness of r_1 from $\mathcal{V}$. $\mathcal{P}$ sets $\rho(x||y||m_1||0) := r_1$ and $\delta(r_1) := m_1||r_1$. Go to next iteration for i.
   (e) If $i > j$, suppose the last element of θ_i is k.
      i. If $\theta_i = k$ or $k \geq \eta$ or $k = 0$, $\mathcal{P}^*$ draws a random number $r \in \mathbb{F}$, answers the query with r, then sets $\rho(\theta_i) := r$. Go to next iteration for i.
      ii. If $e_k = 0$, let $r_k \in \mathbb{F}$ be the first element of θ_i. $\mathcal{P}^*$ splits θ_i to $r_k||m_{k+1}||k$. If $\delta(r_k)$ is defined and $L(\delta(r_k)) = k-1$, $\mathcal{P}^*$ sets $\mathcal{V}$' state to $\mathsf{cvs} = \delta(r_k)$. Then $\mathcal{P}^*$ sends m_{k+1} to $\mathcal{V}$. After receiving r_{k+1} from $\mathcal{V}$, $\mathcal{P}^*$ answers $\mathbb{P}^\rho$ with r_{k+1}, sets $\rho(\theta_i) := r_{k+1}$, and sets $\delta(r_{k+1}) := \mathsf{cvs}||m_k||r_{k+1}$. If $\delta(r_k)$ is note defined or $L(\delta(r_k)) \neq k-1$, $\mathcal{P}^*$ draws a random number $r \in \mathbb{F}$, answers the query with r, then sets $\rho(\theta_i) := r$. Go to next iteration for i.
      iii. If $e_k = 1$, let $r_k \in \mathbb{F}$ be the last element ahead of k. $\mathcal{P}^*$ splits θ_i to $m_{k+1}||r_k||k$. If $\delta(r_k)$ is defined and $L(\delta(r_k)) = k-1$, $\mathcal{P}^*$ sets $\mathcal{V}$' state to $\mathsf{cvs} = \delta(r_k)$. Then $\mathcal{P}^*$ sends m_{k+1} to $\mathcal{V}$. After receiving r_{k+1} from $\mathcal{V}$, $\mathcal{P}^*$ answers $\mathbb{P}^\rho$ with r_{k+1}, sets $\rho(\theta_i) := r_{k+1}$, and sets $\delta(r_{k+1}) := \mathsf{cvs}||m_k||r_{k+1}$. If $\delta(r_k)$ is not defined or $L(\delta(r_k)) \neq k-1$, $\mathcal{P}^*$ draws a random number $r \in \mathbb{F}$, answers the query with r, then sets $\rho(\theta_i) := r$. Go to next iteration for i.

Our construction has two major differences from the construction in [6]. In the construction above, $\mathcal{P}^*$ guesses the statement of $f(x) = y$ that $\mathbb{P}$ would use in the proof by assuming that the j-th query to the random oracle is $x||y||m_1||0$, instead of knowing it in advance. This only introduces a polynomial loss on the probability. Moreover, the query to the random oracle contains the round number. This is because in our non-interactive argument in Protocol 3, to verify the Merkle tree path, the prover's message is sometimes on the left and sometimes on the right of the input of the hash. Our construction of $\mathcal{P}^*$ tracks this information by the round number in order to determine the ordering of the queries to the random oracle. In particular, $\mathcal{P}^*$ use k to detect which round is relevant to the query. For each k, with the public indicator e_k, $\mathcal{P}^*$ knows that $\mathbb{P}$'s message is in the head or the tail of the string.

Analysis of $\mathcal{P}^*$. We now analyze $\mathcal{P}^*$'s ability to cheat given $\mathbb{P}$'s ability to cheat.

Let $\mathcal{U}(\lambda)$ denote the uniform distribution over all functions on $\rho : \mathbb{F}^* \to \mathbb{F}$. If ρ is uniformly sampled from $\mathcal{U}(\lambda)$, then we write $\rho \leftarrow \mathcal{U}(\lambda)$ and say that ρ is a random oracle. We claim that $\mathcal{P}^*$ simulates a $\rho \in \mathcal{U}(\lambda)$ uniformly at random. That is because, given any new input, $\mathcal{P}^*$ responds either with a uniformly random point generated by himself, or a uniformly random point provided by $\mathcal{V}$. It is equivalent to draw ρ uniformly at random in the beginning of the non-interactive protocol.

We claim that if $\mathbb{P}^\rho$ outputs the proof of $(x, y, m_1, r_1, \ldots, m_\eta, r_\eta)$ that makes $\mathbb{V}$ accept with probability $\mathsf{nonegl}(\lambda)$, then $\mathcal{P}^*$ will have $\mathsf{cvs} = m_1||r_1||\ldots||m_\eta||r_\eta$ for $F(x) = y$ to win the game with probability $\mathsf{nonegl}(\lambda)$. The formal proof is provided in the following.

Without loss of generality, we suppose $e_k = 0$ for $1 \le k < \eta$. We define some events as follows.

1. E_1 represents that $\mathbb{P}^\rho$ outputs $(x, y, m_1, r_1, \ldots, m_\eta, r_\eta)$ that makes $\mathbb{V}$ accept. Then it satisfies $r_1 = \rho(x||y||m_1||0)$ and $r_i = \rho(r_{i-1}||m_i||i-1)$ for $1 < i \le \eta$.

2. E_2 represents that $\mathbb{P}^\rho$ queries $\mathcal{P}^*$ at $x||y||m_1||0, r_1||m_2||1, \ldots, r_{\eta-1}||m_\eta||\eta - 1$ in order and $\mathcal{P}^*$ does not return the same value for different queries during the entire query process.

3. E_3 represents that $\mathcal{P}^*$ predicts that $\mathbb{P}^\rho$ queries $x||y||m_1||0$ for the first time in the j-th query accurately.

4. E_4 represents that $\mathsf{cvs} = m_1||r_1||\ldots||m_\eta||r_\eta$ for $F(x) = y$ is in $\mathcal{P}^*$'s $\mathsf{SeenStates}$ set and $\mathcal{P}^*$ wins the game.

First, we prove $\Pr[E_1 \wedge \neg E_2] \le \mathsf{negl}(\lambda)$. Let r_0 denote $x||y$. There are three cases covering $E_1 \wedge \neg E_2$: (i) E_1 happens but $\mathbb{P}$ does not query $r_{i-1}||m_i||i-1$ for some $i \in \{1, \ldots, \eta - 1\}$; (ii) E_1 happens but $\mathbb{P}$ queries $r_i||m_{i+1}||i$ before querying $r_{i-1}||m_i||i-1$ for some $i \in \{1, \ldots, \eta - 1\}$; (iii) E_1 happens but $\mathcal{P}^*$ returns the same value for different queries. The probability of case (i) and the probability of case (ii) are both $\mathsf{negl}(\lambda)$ as $\mathbb{P}^\rho$ can not correctly guess the output of ρ for any input except with $\mathsf{negl}(\lambda)$. The probability of case (iii) is also $\mathsf{negl}(\lambda)$ as $\mathbb{P}^\rho$ can not find a collision of ρ except with $\mathsf{negl}(\lambda)$. By union bound, $\Pr[E_1 \wedge \neg E_2] \le \mathsf{negl}(\lambda)$. Suppose $\Pr[E_1] = p = \mathsf{nonegl}(\lambda)$, $\Pr[E_1 \wedge E_2] = \Pr[E_1] - \Pr[E_1 \wedge \neg E_2] = \mathsf{nonegl}(\lambda) - \mathsf{negl}(\lambda) = \mathsf{nonegl}(\lambda)$. Then we have $\Pr[E_1 \wedge E_2 \wedge E_3] = \Pr[E_3 | E_1 \wedge E_2] \cdot \Pr[E_1 \wedge E_2] \ge \frac{1}{\Delta} \cdot \mathsf{nonegl}(\lambda) = \mathsf{nonegl}(\lambda)$.

Next, we show $\Pr[E_4 | E_1 \wedge E_2 \wedge E_3] = 1$. We prove that if $E_1 \wedge E_2 \wedge E_3$ happens, $\delta(r_i) = m_1||r_1||\ldots||m_i||r_i$ for $1 \le i \le \eta$ by induction. For each i, $\delta(r_i)$ was included in δ only once as there is no collision during the query phase. For $i = 1$, when $\mathbb{P}$ queries $x||y||m_1||0$ in the j-th query, $\mathcal{P}^*$ sets $\delta(r_1) := m_1||r_1$. For $i = k$, suppose $\mathcal{P}^*$ sets $\delta(r_k) := m_1||r_1||\ldots||m_k||r_k$ when $\mathbb{P}$ queries $r_{k-1}||m_k||k-1$, when $\mathbb{P}$ queries $r_k||m_{k+1}||k$ hereafter, $\mathcal{P}^*$ sets $\delta(r_{k+1}) := \delta(r_k)||m_k||r_{k+1} = m_1||r_1||\ldots||m_{k+1}||r_{k+1}$. Hence $\mathsf{cvs} = m_1||r_1||\ldots||m_\eta||r_\eta$ for $F(x) = y$ will be in $\mathcal{P}^*$'s $\mathsf{SeenStates}$ set and $\mathcal{P}^*$ will win the game. $\Pr[E_4] \ge \Pr[E_1 \wedge E_2 \wedge E_3] = \mathsf{nonegl}(\lambda)$. □

Theorem 5. *If the interactive proof* T *for* $F(x) = y$ *with* η *rounds is secure against state restoration attacks, after inserting* $c = \mathsf{poly}(\lambda)$*-round interaction in the i-th round of* T *as in Protocol 5, the new interactive protocol* T_{du} *is also secure against state restoration attacks for* $F(x) = y$.

Proof. (sketch) Let $\mathcal{P}$ be the prover in T and $\mathcal{P}_{\mathsf{du}}$ be the prover in T_{du}. Suppose T_{du} inserts c-round interaction with arbitrary messages of $(\mathsf{du}_1, \nu_1, \ldots, \mathsf{du}_c, \nu_c)$ in the i-th round of T. If $\mathcal{P}_{\mathsf{du}}$ can win the game described in Definition 5 with probability p for x, y satisfying $F(x) \ne y$ by generating a cvs of $(m_1, r_1, \ldots, m_i, \mathsf{du}_1, \nu_1 \ldots, \mathsf{du}_c, \nu_c, r_i, m_{i+1}, r_{i+1}, \ldots, m_\eta, r_\eta)$, then $\mathcal{P}$ can invoke $\mathcal{P}_{\mathsf{du}}$ to generate the cvs of $(m_1, r_1, \ldots, m_i, r_i, m_{i+1}, r_{i+1}, \ldots, m_\eta, r_\eta)$ to win Game 1 with probability at least p. □

Replacing T with T_{du} and applying the non-interactive transformation to T_{du} indicate that we can integrate an authenticated path in the Merkle tree into such a protocol T at the cost of extra $\log N$ rounds, where N is the size of the Merkle tree. Therefore, for each verifier $\mathcal{V}_j$, Protocol 5 integrate $\log N$ Merkle paths into Protocol 2 at the cost of extra $\log^2 N$ rounds.

In Protocol 3, the statement $\mathcal{P}$ wants to convince V_j is equivalent to $F_j(\mathsf{com}_{\tilde{V}_d}) = [C(\mathbf{in})]_j$, where $\mathsf{com}_{\tilde{V}_d}$ is the commitment of $\tilde{V}_d$ and $\mathcal{P}$ broadcasts to all verifiers at the beginning. F_j represents that there exists a degree-t univariate polynomial $f(x) = c_0 + c_1 x + \ldots + c_t x^t$ such that $f(\omega^j) = [C(\mathbf{in})]_j$ and $\mathsf{com}_{\tilde{V}_d} = \mathsf{MVPC.Commit}(\tilde{c}, pp)$, where $\tilde{c}$ the multilinear extension of $(c_0, \ldots, c_t)$. For each $\mathcal{V}_j$, Protocol 3 practises our new heuristic transformation on Protocol 5 for $F_j(\mathsf{com}_{\tilde{V}_d}) = [C(\mathbf{in})]_j$ to make the proof non-interactive. The protocol will be sound after the transformation.