# Publicly Auditable MPC-as-a-Service with succinct verification and universal setup

Sanket Kanjalkar
*Blockstream Research*
*sanket1729@blockstream.com*

Ye Zhang
*New York University*
*yezhang@nyu.edu*

Shreyas Gandlur
*Princeton University*
*sgandlur@princeton.edu*

Andrew Miller
*University of Illinois, Urbana Champaign*
*soc1024@illinois.edu*

***Abstract*—In recent years, multiparty computation as a service (MPCaaS) has gained popularity as a way to build distributed privacy-preserving systems like blockchain trusted parameter setup ceremonies, and digital asset auctions. We argue that for many such applications, we should also require that the MPC protocol is *publicly auditable*, meaning that anyone can check the given computation is carried out correctly — even if the server nodes carrying out the computation are all corrupt. In a nutshell, the way to make an MPC protocol auditable is to combine an underlying MPC protocol with a verifiable computing proof (in particular, a SNARK). Building a general purpose MPCaaS from existing constructions would require us to perform a costly "trusted setup" every time we wish to run a new or modified application. To address this, we provide the first efficient construction for auditable MPC that has a one time universal setup. Despite improving the trusted setup, we match the state-of-the-art in asymptotic performance: the nodes incur a linear computation overhead and constant round communication overhead compared to the underlying MPC, and the audit size and verification are logarithmic in the application circuit size. We also provide an implementation and benchmarks that support our asymptotic analysis in example applications. Furthermore, compared with existing auditable MPC protocols, besides offering a universal setup our construction also has a 3x smaller proof, 3x faster verification time and comparable prover time.**

## 1. Introduction:

The past few years have seen increasing interest in the Secure-Multiparty-Computing-as-a-Service (MPCaaS) model. MPCaaS is a distributed system where a quorum of servers provide confidential computing service to clients. All its security guarantees (including confidentiality, integrity, and optionally availability), rely on an assumption that at least some of the servers are honest (either a majority of the servers or even just one, depending on the protocol). This model is flexible and well-suited to a range of applications including auctions and digital asset trading [1], [2], anonymous messaging systems [3], [4], computing statistics on confidential demographics data [5], [6], and for trusted parameter generation in other cryptography applications [7].

An important research focus in making MPCaaS practical has been to reduce the necessary trust assumptions to a minimum. Malicious-case security for confidentiality and integrity guarantees has now become a standard feature of most implementations [8], [9], [10], [11], and protocols like HoneybadgerMPC [4] and Blinder [12] furthermore guarantee availability in this setting as well.

**The need for public auditability.** The present work aims to reduce the trust assumptions for practical MPCaaS even further. Our focus is *publicly auditable MPC*, which can best be understood as a form of graceful degradation for MPC security properties, as summarized in Table 1. In the ordinary MPC setting, both confidentiality (Conf) and integrity (Int) only hold when the the number of corrupted parties is less than a threshold $t$; in Robust MPC [4][12], availability (Avail) holds under these conditions too. Let $f$ refer to the number of parties *actually* corrupted, such that $f > t$ means the ordinary assumptions fail to hold. Auditable MPC enables anyone to verify the correctness of the output, ensuring that the integrity guarantees hold even when $t < f$. Note that this notation describes equally well both the honest majority setting $t < n/2$ or $t < n/3$ (like Viff [13], HoneyBadgerMPC[4], HyperMPC [11], or any Shamir sharing based MPC), as well as the dishonest majority setting $t < n$ (like SPDZ [14] and related protocols). The complementary relation between auditability and other MPC qualities is summarized in Table 1. To give more context, the integrity guarantee is that the computation, if it completes, is performed correctly, i.e. the correct function is applied to the specified inputs. For blockchain applications which use SNARK, it is important ensure the setup ceremony[7] for parameter sampling is carried out correctly even if all participants are compromised. As another example, in an digital asset auction, we would want to know that the quantity of digital assets is conserved. Note that in these applications, integrity may matter even to users who did not themselves provide input (randomness or bids) to the service. Assuming a *robust offline phase*[12]( Section 2.2), we show a construction of robust auditable MPC.

**Auditable MPC with one-time trusted setup.** In a nutshell, auditable MPC is built from an underlying non-auditable MPC, composed with commitments and zero-knowledge proofs [15], [16]. The

TABLE 1. GRACEFUL DEGRADATION OF MPC PROTOCOLS DEPENDING ON NUMBER OF ACTUAL FAULTS ($f$), VERSUS FAULT TOLERANCE PARAMETER ($t$)

| | $f \leq t$ | $f > t$ |
|---|---|---|
| non-robust MPC | Conf, Int | |
| non-robust auditable MPC | Conf, Int | Int |
| robust MPC | Conf, Int, Avail | |
| robust auditable MPC | Conf, Int, Avail | Int |

resulting arrangement is illustrated in Figure 1. In addition to providing input to the servers, clients also publish commitments to their inputs to a public bulletin board that can be realized by a blockchain. The servers, in addition to computing MPC on the secret shared data, also produce a proof that resulting output is computed correctly. Any auditor can verify the proof against the input commitments to check the output is correct.

The initial version of auditable-MPC by Baum[16] examines the entire protocol transcript to audit the computation. Later Veeningen showed how to construct an efficient auditable MPC from an adaptive Commit-and-Prove zk-SNARK [17] (CP-SNARK) based on Pinocchio [18] which is more efficient than [16]. Adaptive roughly means that it is secure even when the relations are chosen after the inputs (statements) are committed to. However, like many SNARKs, Pinocchio relies on a difficult to carry out trusted setup to generate parameters. [19], [20], [21]. Since Pinocchio's trusted setup depends on the particular circuit, there is no way to update the program once the setup is complete. Any bug fix or feature enhancement to the MPC program would require performing the trusted setup ceremony again.

*The goal of our work is to remove this barrier to auditable MPC, by enabling a single trusted ceremony to last for the lifetime of a system, even if the programs are dynamically updated.* Our approach makes use of recent advances in zk-SNARKs, especially the Marlin zk-SNARK [22], and adapts it to the auditable MPC setting.

**Technical challenges and how we overcome them.** First, to summarize our approach, we follow Veeningen and build auditable MPC from an adaptive CP-SNARK. To achieve this, we follow the generic framework of LegoSNARK [23], and compose several existing CP-SNARK gadgets, namely ones for sum-check [24], linear relations, and opening of polynomial commitments, resulting in a new construction we call *Adaptive Marlin*.

From Adaptive Marlin, to build an auditable MPC requires two more steps. First, the prover algorithm must be replaced with a distributed MPC alternative, leaving the verifier routine essentially the same. Fortunately this turns out to be straightforward; Adaptive Marlin supports distributed computation in a natural way, and the soundness proof remains intact for the verifier. Second, we must provide a way to combine the input commitments contributed by different clients. This poses a greater challenge; in particular, Veeningen's approach to

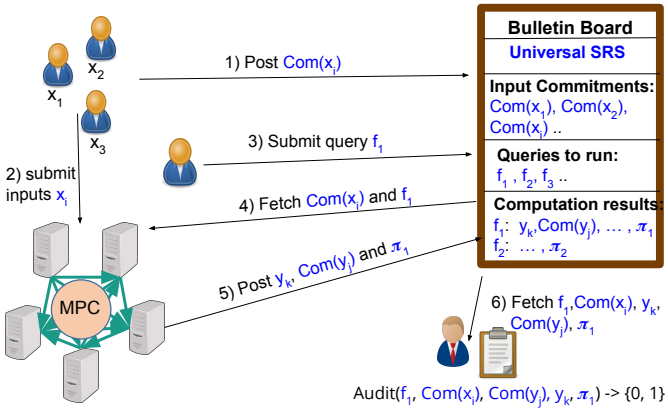


Figure 1. Auditable MPC-as-a-Service with one-time trusted setup. Clients with inputs $x_i$ post commitments $Com(x_i)$ and query $f$ to the bulletin board. The servers produce output commitments $Com(y_j)$ and SNARK proof $\pi_1$ onto the bulletin board. The auditor collects the data from bulletin board and verifies the correctness execution of $f$

combining input commitments does not work in Marlin since it makes use of the circuit-dependent structure of the Pinocchio CRS.

Our solution is based on a new primitive, *polynomial evaluation commitments*(PEC)4, polynomial commitments that can be assembled in a distributed fashion. Each party contributes one evaluation point and they jointly compute a commitment to the resulting interpolated polynomial. This primitive serves as a bridge between LegoSNARK and Veeningen's auditable MPC.

To summarize our contributions:

- **Adaptive zk-SNARK with universal reference string and constant verification time.** Adaptive Marlin is the first adaptive zk-SNARK for general arithmetic circuits that has $O(\log N)$ verification time, $O(1)$ proofs and relies on a universal reference string. This is an asymptotic improvement over LegoUAC, the only known adaptive zk-SNARK with universal reference string [23], which has $O(\log^2 N)$ sized proofs and verification time where $N$ is the size of the circuit.

- **Auditable Reactive MPC with one-time trusted setup.** Informally, reactive MPC is a type of MPC where the computations to perform may be determined dynamically, even after inputs are provided. By constructing Auditable MPC based on Adaptive Marlin, we avoid the need to run a new trusted setup each time a new program is defined, removing an important obstacle to deployment. We provide our formal security analysis using the same ideal functionality setting as Veeningen, except that we go further in considering the full universal composability environment.

- **Implementation.** We implement and evaluate our auditable MPC construction. In our experiments with 32 MPC servers, over 1 million constraints, and 8 statement size, our prover time is about 678 seconds, auditing time less than 40ms, proof size is $\approx$ 1.5Kb, total MPC communication overhead is a constant 700Kb with five additional rounds of communication. As an additional contributions, we also

implement and evaluate sample application workloads, including an auction and a statistical test (logrank). As a representative figure, in the auction application with 125 bidders (an R1CS with 9216 constraints), with 32 MPC servers, the auditor time is about 50ms, while the time to compute the proof is about 20 seconds, or a total of 4 minutes when including the underlying MPC computations — overall the auditable MPC is an overhead of 10% compared to plain (non-auditable) MPC.

In terms of performance, despite not relying on a circuit specific setup, our prover time is comparable to Veeningen's. In some settings, our auditor time and proof size show asymptotic and concrete improvements. For applications where each client contributes only a small input, our auditor has a constant pairing cost and constant proof size, which is asymptotically better than Veeningen's construction that has linear pairing cost and linear proof size in terms of number of clients (input commitments).

## 2. Preliminaries

### 2.1. Notation

Let $\langle group \rangle = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g, h, e)$ where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of a prime order $q$, $g$ generates $\mathbb{G}_1$, $h$ generates $\mathbb{G}_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a (non-degenerate) bilinear map with a security parameter $\lambda$. Bold letters like $\mathbf{v}$ denotes a vector of elements $[v_i]_{i=1}^n$. $|\mathbf{x}|$ denotes the cardinality of $\mathbf{x}$, while $|A|$ denotes the number of non-zeros elements when $A$ is a matrix. $\mathbf{x} \circ \mathbf{y}$ denotes the element wise product of $\mathbf{x}$ and $\mathbf{y}$. $\mathbb{F}_p$ denotes the finite field of prime order $p$ (usually we leave $p$ implicit and write $\mathbb{F}$), $g(X) \in \mathbb{F}^d[X]$ denotes a polynomial of degree at most $d$. If $g$ is a function from $H \to \mathbb{F}$, where $H \subseteq \mathbb{F}$ then $\hat{g}$ denotes a low degree extension of $g$ (the smallest polynomial that matches $g$ over all of $H$).

### 2.2. Secret Sharing and MPC

Secure multi-party computation (MPC) enables parties to jointly compute a function over secret shared inputs, while keeping those inputs confidential — only disclosing the result of the function.

We present our construction for Shamir Secret Sharing (for honest majority MPC), although it is also compatible with other linear secret sharing such as SPDZ (for dishonest majority MPC). For prime $p$ and a secret $s \in \mathbb{F}_p$, $[\![s]\!]$ denotes Shamir Secret Sharing [25](SSS) in a $(n, t)$ setting. We omit the superscript and/or subscript when it is clear from context. For a concrete instantiation in our benchmarks (Section 6), we assume a robust preprocessing MPC using Beaver multiplication [26] and batch reconstruction [27], [28], similar to HoneyBadgerMPC [4]

### 2.3. Extractable Commitments

Our construction for auditable MPC relies on a stronger variant of commitment schemes known as extractable trapdoor commitments. For space, we define these in the appendix.

### 2.4. Polynomial Commitments

Polynomial commitments [29] allow a prover to commit to a polynomial, and later reveal evaluations of the polynomial and prove they are correct without revealing any other information about the polynomial. Following Marlin [22], we define Polynomial Commitments($\mathsf{PC}$) over $\mathbb{F}$ by a set of algorithms $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$. We only state the definition for creating a hiding commitment to a single polynomial for a single evaluation point with only one maximum degree(Omitting the PC.Trim) bound that is necessary for our application.

#### 2.4.1. Polynomial Commitment Definitions.

- $\mathsf{Setup}(1^\lambda, D) \to \mathsf{ck}, \mathsf{rk}$ On input a security parameter $\lambda$ (in unary), and a maximum degree bound $D \in \mathbb{N}$, Setup samples some trapdoor $\mathsf{td}$ and outputs some public parameters $\mathsf{ck}, \mathsf{rk}$ for supporting maximum degree bound $\mathsf{D}$.
- $\mathsf{Commit}(\mathsf{ck}, \phi; \omega) \to c$ Given input committer key $\mathsf{ck}$, univariate polynomial $\phi$ over a field $\mathbb{F}$, Commit outputs commitments $c$ to the polynomial $\phi$ using randomness $\omega$.
- $\mathsf{Open}(\mathsf{ck}, \phi, q; \omega) \to v, \pi$ On inputs $\mathsf{ck}$, uni-variate polynomial $\phi$ over a field $\mathbb{F}$, a query point $q \in F$, Open outputs an evaluation $v$ and evaluation proof $\pi$. The $\omega$ used must be consistent with the one used in Commit.
- $\mathsf{Check}(\mathsf{rk}, c, q, v, \pi) \to \{0, 1\}$: On input reciever key $\mathsf{rk}$, commitment $c$, a query $q \in F$, claimed evaluation $v \in \mathbb{F}$ at $q$ and evaluation proof $\pi$, Check outputs 1 if $\pi$ attests that all the claimed evaluation corresponding the committed polynomial.

Note that we will later refer to this definition as "plain" polynomial commitments, in comparison to the polynomial evaluation commitments (Section 4).

**2.4.2. Construction in AGM.** We next describe the Polynomial Commitment construction from Marlin, which is a variation of KZG [29] adapted for the Algebraic Group Model (AGM) [30]. In particular it relies on a pairing-based group with the Strong Diffie-Hellman assumption ($\mathsf{SDH}$), for which a formal definition is given in the Appendix C.

- Setup : Upon input $\lambda$ and $D$, Setup samples random elements in $\mathbb{F}_q$ and outputs $\mathsf{ck} := (\langle \mathsf{group} \rangle, \mathbf{\Sigma})$ and $\mathsf{rk} := (D, \langle \mathsf{group} \rangle, g^\gamma, h^\alpha)$ where $\mathbf{\Sigma}$ is sampled as follows:

$$\Sigma := \begin{pmatrix} g & g^{\alpha} & g^{\alpha^2} & \cdots & g^{\alpha^D} \\ g^{\gamma} & g^{\alpha\gamma} & g^{\gamma\alpha^2} & \cdots & g^{\gamma\alpha^D} \end{pmatrix} \quad (1)$$

- Commit: On input $\mathsf{ck}$, univariate polynomial $\phi$ and randomness $\omega$, Commit operates as follows: If $deg(\phi) > D$, abort. Else, sample a random polynomial $\bar{\phi}$ of $\deg(\phi)$ according to randomness $\omega$. Output $c := g^{\phi(\alpha)} g^{\gamma\bar{\phi}(\alpha)}$
- Open: On inputs $\mathsf{ck}$, uni-variate polynomial $\phi$ over a field $\mathbb{F}$, a query point $q \in \mathbb{F}$, Open outputs an evaluation $v := \phi(q)$ and evaluation proof $\pi := (\mathsf{w}, \bar{v})$ as follows: Compute $w(x) = \frac{\phi(X) - \phi(q)}{X - q}$ and

$\bar{w}(x) = \frac{\bar{\phi}(X)-\bar{\phi}(q)}{X-q}$ and set $\mathsf{w} := g^{w(X)} g^{\gamma \bar{w}(X)}$, $\bar{v} := \bar{\phi}(q)$.

- Check: On input receiver key rk, commitment $c$, a query $q \in \mathbb{F}$, claimed evaluation $v \in \mathbb{F}$ at $q$ and evaluation proof $(\mathsf{w}, \bar{v}) = \pi$, Check outputs as $e(c/(g^v g^{\gamma \bar{v}}), h) \stackrel{?}{=} e(\mathsf{w}, h^{\alpha}/h^{q})$

### 2.5. zkSNARKs for R1CS Indexed Relations:

A zkSNARK is an efficient proof system where a prover demonstrates knowledge of a satisfying witness for some statement in an NP language. We focus on zkSNARKs for a generic family of computations, based on R1CS relations, a well known generalization of arithmetic circuits. For performance, we are interested in succinct schemes where the proof size and verification time are sublinear (or indeed constant, as with our construction) in the number of gates or constraints.

Following Marlin, we define indexed relations $\mathcal{R}$ as a set of triples $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ where $\mathbb{i}$ is the index, $\mathbb{x}$ is the statement instance and $\mathbb{w}$ is the corresponding witness. The corresponding language $\mathcal{L}(\mathcal{R})$ is then defined by the set of pairs $(\mathbb{i}, \mathbb{x})$ for which there exists a witness $\mathbb{w}$ such that $((\mathbb{i}, \mathbb{x}), \mathbb{w}) \in \mathcal{R}$. In standard circuit satisfaction case, the $\mathbb{i}$ corresponds to the description of the circuit, $\mathbb{x}$ corresponds to the partial assignment of wires (also known as public input) and $\mathbb{w}$ corresponds to the witness.

### 2.6. Universal Structured Reference Strings

The vast majority of zkSNARK schemes rely on a common reference string crs, which must be sampled from a given distribution at the outset of the protocol. In a perfect world, we would only need to sample reference strings from the uniform distribution over a field (a urs), in which case it can be sampled using public randomness [31]. However, most practical SNARKs require sampling the reference string from a structured distribution (an srs), which requires a (possibly distributed) trusted setup process [19], [21], [20].

As a practical compromise, we aim to use a universal structured reference string (u-srs), which allows a single setup to support all circuits of some bounded size. A deterministic or public coin procedure can specialize the trusted setup to a given circuit. This avoids the need to perform the trusted setup each time a new circuit is desired. Some u-srs constructions (like the one we use) are also updatable[32], meaning an open and dynamic set of participants can contribute secret randomness to it indefinitely. Throughout this paper, we refer to u-srs as srs as the universality is clear from the context.

A zkSNARK with an srs is a tuple of algorithms $\mathsf{ARG} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V})$. The setup G samples the srs, supporting arbitrary circuits up to a fixed size. The indexer I is a deterministic polynomial-time algorithm that uses srs and circuit index $\mathbb{i}$ satisfying the srs constraint bound, outputs an index proving key ipk and a verification key ivk. The Prover P uses ipk to provide a proof $\pi$ for indexed relation $\mathcal{R}$. The Verifier V then checks $\pi$ using ivk.

**2.6.1. Review of Marlin's construction.** As our construction closely builds on Marlin, we reuse most of its notation, and review its construction here. The Marlin construction is centered around an interactive "holographic proof" technique [33], combined with polynomial commitments and Fiat-Shamir to make it non-interactive. In a holographic proof, the verifier does not receive the circuit description as an input but, rather, makes a small number of queries to an encoding of it. This deterministic algorithm responsible for this encoding is referred to as the indexer I. Marlin focuses on the setting where the encoding of the circuit description and the proofs consist of low-degree polynomials. Another way to look at this is that this imposes a requirement that honest and malicious provers are "algebraic" (See Appendix C.2).

In brief, the Marlin protocol proceeds in four rounds, where in each round the verifier V sends a challenge and prover P responds back with one or more polynomials; after the interaction, V probabilistically queries the polynomials output by the indexer I as well as those polynomials output by the prover P, and then accepts or rejects. The verifier does not receive circuit index $\mathbb{i}$ as input, but instead queries the polynomials output by I that encode $\mathbb{i}$. For our construction, we require a MPC version of Marlin protocol shown in Appendix G.

## 3. Overview of Our Construction

### 3.1. Motivating application: Auction

We start by explaining an auction application that we use as a running example throughout. We envision a distributed service that accepts private bids from users, and keeps a running tally of the current best price, but both the bids and the price are only stored in secret shared form. Finally after all users have submitted bids, the servers publish the winning price. This application can be summarized with the following two procedures, where secret sharing notation $[\![x]\!]$ indicates that $x$ is confidential:

- Initialize state: $[\![\mathsf{bestprice}]\!] := [\![0]\!]$
- ProcessBids(inputs: $[\![x_i]\!]$ from $P_i$, state:$[\![\mathsf{bestprice}]\!]$):
  for each $[\![x_i]\!]$ that has not been processed:
  $[\![\mathsf{newprice}]\!] := \max([\![x_i]\!], [\![\mathsf{bestprice}]\!])$
  return OK, $[\![\mathsf{newprice}]\!]$
- Finalize(inputs: $\emptyset$, state=$[\![\mathsf{bestprice}]\!]$):
  return $[\![\mathsf{bestprice}]\!]$.open(), $\perp$

Note that we write our example to process arbitrary-size batches of user-submitted bids at a time. This is to illustrate the flexibility of our construction, since it supports reactive computations (each computation can provide public output as well as secret shared output carried over to the next operation) as well as support for large circuits. In

our example, the Finalize procedure also discloses the current state. In general, each procedure can be characterized by the following quantities: $X$, the total size of secret inputs; $K$, the number of distinct clients providing input in each invocation; and $M$, the total number of gates needed to express the procedure as an arithmetic circuit. In our example, each ProcessBid invocation receives $K$ constant-size bids from different parties, so $X = \Theta(K)$, and the circuit comprises a comparison for each bid, so $M = \Theta(Kc)$ where $c$ is the number of gates for each comparison subcircuit. Since we are building auditable MPC from SNARKs, we are primarily interested in *witness succinctness*, meaning the verification cost is independent of the circuit size $M$, although it will in general depend on $K$ and $X$ (as we explain more in Section 5). When the verification cost is also independent of $X$, we call it *statement succinct*.

### 3.2. System overview of Auditable MPCaaS

Auditable MPC is a distributed system architecture for performing secure computations over inputs provided by clients. The computation is organized into several phases. For simplicity we describe these as occurring one after the other, though in the general (reactive) setting each phase can occur multiple times and may run concurrently with each other.

**One-time Setup Phase.** The offline phase of our auditable MPC consists of two components. First is the one-time setup for the underlying SNARK and client input commitment scheme; this setup needs only be carried out once, regardless of the circuit programs to evaluate. The second is translating a circuit description into an index format. This is deterministic and anyone can publicly recompute and check this computation.

**Commit Inputs.** We have data client parties $I_i$ which provide inputs $x_i$ to the computation. The data-input parties $I_i$ provide commitments of their input on the bulletin board for availability.

**Define Program.** The input party $I_{inp}$ which provides the computation function $f$. We model this as a separate party, but in general this would be chosen through a transparent process, such as through a smart contracts. Marlin Indexed circuit generated indexer prover and verification keys: The indexer should be run every time there is a request for a new computation indexing or an update to an existing computation.

**MPC Pre-processing.** In order to facilitate fast online multiplication of MPC servers, it is typically necessary to prepare offline Beaver triples and random element shares [26].

**Compute phase.** Next, data clients post secret shares $[\![x_i]\!]$ their input values to the MPC servers. The online phase includes interaction between MPC servers $P_j$ to compute the desired user function and generation of proof of correct execution. The auditor can collect the proofs from the bulletin board and verify that the computation was carried out correctly. Figure 1 shows the high-level overview describing the online phase of auditable MPC. The servers carry out MPC protocols to compute the function $f(x_i, y_{prev}) = y_k, \mathsf{Com}(y_j)$, where $y_k$ is the public output and $\mathsf{Com}(y_j)$ is a commitment to a secret output along with a SNARK proof $\pi$.

**3.2.1. Audit phase.** The auditor receives the output and verifies that it is correct. Finally, the auditor verifies computation was carried out correctly by collecting all input commitments $\mathsf{Com}(x_i)$, secret outputs $\mathsf{Com}(y_j)$, public outputs $y_k$ and the proof $\pi$. To completely audit the computation, one would need to verify the MPC pre-processing, circuit indexing along with execution proof. We only consider the costs for verifying proofs because the indexing cost be amortized over multiple uses and because we operate robust offline pre-processing model.

## 4. Polynomial Evaluation Commitments

The main building block for our auditable MPC construction is a new variant of polynomial commitments called Polynomial Evaluation Commitments (PEC). In the original polynomial commitment definition [29], the committer must have chosen a polynomial before calling the Commit procedure. To adapt these for use in MPC, our extended PEC definition supports an alternative, distributed way to create the polynomial commitments: Each party starts with a commitment to just an evaluation point on the polynomial. Next, the evaluation commitments are combined and interpolated to form the overall polynomial commitment. In our definition below, the changes to plain polynomial commitments (in Section 2.4) are highlighted *blue*. Briefly, these are 1) Setup outputs additional commitment evaluation keys $\mathbf{ck_e}$ and 2) the Commit operation is split into $\mathsf{Commit_{eval}}$ and Interpolate. More formally, our polynomial evaluation commitment scheme over a field $\mathbb{F}$ is defined by the following set of algorithms $\mathsf{PEC} = (\mathsf{Setup}, \mathsf{Commit_{eval}}, \mathsf{Interpolate}, \mathsf{Open}, \mathsf{Check})$. We index parties by $i$ and the evaluation of the polynomial by $j$, $e_{ij}$ denotes the $j$th evaluation by the $i$th party.

- $\mathsf{Setup}(1^\lambda, D, K) \rightarrow \mathbf{ck_e}, \mathsf{ck_p}, \mathsf{rk_p}, \mathsf{td}$ : On input a security parameter $\lambda$, and a maximum degree bound $D \in \mathbb{N}$, number of parties $K \in \mathbb{N}$ , Setup samples some trapdoor td and outputs public parameters $\mathbf{ck_e}, \mathsf{ck_p}, \mathsf{rk_p}$.
- $\mathsf{Commit_{eval}}(\mathsf{ck_{e_i}}, \boldsymbol{e_i}, \boldsymbol{p_i}; \omega) \rightarrow c_{e_i}$ Given input evaluation committer key $\mathsf{ck_{e_i}}$, univariate polynomial evaluations $\boldsymbol{e_i} = [e_{ij}]_{j=1}^{d}$ at evaluation point $\boldsymbol{p_i} = [p_{ij}]_{j=1}^{d} (d \leq \frac{D}{K})$ over a field $\mathbb{F}$, $\mathsf{Commit_{eval}}$ outputs commitment $c_{e_i}$ to the evaluations $\boldsymbol{e_i}$ using randomness $\omega$.
- $\mathsf{Interpolate}(\mathsf{ck_p}, \mathbf{c_e}, \mathbf{p}) \rightarrow c$: On input $\mathsf{ck_p}$ and commitment to evaluations $\mathbf{c_e} = [c_{e_i}]_{i=1}^{k}$ at $\mathbf{p} = [\mathbf{p_i}]_{i=1}^{k}$, Interpolate outputs a commitment $c$ to the interpolated polynomial $\phi(.)$ corresponding the evaluations of the committed in $\mathbf{c_e}$.
- $\mathsf{Open}(\mathsf{ck_p}, \phi, q; \boldsymbol{\omega}) \rightarrow v, \pi$ Same as the PC.Open as discussed in Section 2.4 where $\phi$ is the polynomial interpolated by the evaluations $\mathbf{e} = [[e_{ij}]_{j=1}^{d}]_{i=1}^{k}$.

Note that the $\boldsymbol{\omega} = [\omega_i]$ where $\omega_i$ must be the same as the one used in $\mathsf{Commit}_{\mathsf{eval}}$ for point at index $i$.

- $\mathsf{Check}(\mathsf{rk_p}, c, q, v, \pi) \rightarrow \{0,1\}$: Same as the PC.Check as discussed in Section 2.4.

Additionally, a PEC must satisfy the following properties.

- **Perfect Completeness**: Consider an adversary which chooses evaluations $\mathbf{e} = [e_i]_{i=1}^k$ at evaluation points $\mathbf{p} = [\mathbf{p_i}]_{i=1}^k$ randomness $[\boldsymbol{\omega}]_{i=1}^k$ and query point $q$. Let $c_{e_i}$ denote the commitments to the evaluations $\mathbf{e}$, $\phi$ the interpolated polynomial at $(\mathbf{e}, \mathbf{p})$ and $\boldsymbol{V_p}$ the Vandermonde matrix at evaluation points $\mathbf{p}$ respectively. We say that PEC is complete if the evaluation proofs created by Open for $\phi$ at $q$ are correctly verified by Check with respect to the interpolated commitment $c$ that is generated by $\mathsf{Interpolate}(c_{e_i})$. More formally, we say that PEC is perfect complete if the following probability is 1($\Downarrow$ denotes logic implication).

$$\Pr\left[\begin{array}{c} deg(\phi) \leq \mathsf{D} \\ \Downarrow \\ \mathsf{Check}(\mathsf{rk_p}, c, q, v, \pi) \end{array} \middle| \begin{array}{c} \mathsf{ck_e}, \mathsf{ck_p}, \mathsf{rk_p}, \mathsf{td} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{D}, \mathsf{K}) \\ [\mathbf{e_i}, \mathbf{p_i}, \boldsymbol{\omega}]_{i=1}^k, q \leftarrow \mathcal{A}(\mathbf{ck_e}, \mathsf{ck_p}, \mathsf{rk_p}) \\ [c_{e_i} \leftarrow \mathsf{Com_e}(\mathsf{ck_{e_i}}, \mathbf{e_i}, \mathbf{p_i}; \boldsymbol{\omega_i})]_{i=1}^k \\ c \leftarrow \mathsf{Interpolate}(\mathsf{ck_p}, \mathbf{p}, \mathbf{c_e}) \\ \phi \leftarrow \boldsymbol{V_p}\mathbf{e}, v \leftarrow \phi(q) \\ \pi \leftarrow \mathsf{Open}(ck_p, \phi, q; \boldsymbol{\omega}) \end{array}\right]$$

- **Extractable:** First, consider an adversary $\mathcal{A}_0$ that chooses $k', (k' \leq k \leq K)$ points $\mathbf{p}' = [\mathbf{p}]_{i=1}^{k'}$ and their evaluations $\mathbf{e}' = [\mathbf{e}]_{i=1}^{k'}$. Next, consider an $\mathcal{A}_1$ which upon input setup material $(\mathbf{ck_e}, \mathsf{ck_p}, \mathsf{rk_p})$ and evaluation commitments $\mathbf{c'_e}$(for $\mathbf{e}'$ and $\mathbf{p}'$) chooses commitments to evaluations $\boldsymbol{c_e}$ at evaluation points $\mathbf{p}$. Let $c$ denote the interpolated commitment from $\mathbf{ck_e}$. Finally, consider an adversary $\mathcal{A}_2$ which upon input state st from $\mathcal{A}_1$ outputs a claimed evaluation $v$ at query point $q$ with a proof $\pi$. We say that PEC is extractable if evaluations of the polynomial can be extracted from an adversary $(\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3))$ generating a valid proof. More formally, PEC is extractable if for every size bound $\mathsf{D}, \mathsf{K} \in \mathbb{N}$, every efficient adversary $\mathcal{A}_1, \mathcal{A}_2$ there exists an efficient extractor $\mathcal{E}$ such that for every $\mathcal{A}_3$ the following probability is $\mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{c} c_{e_i} = \mathsf{Com_e}(\phi(p_i); \omega_i) \\ \wedge \mathsf{Check}(\mathsf{rk_p}, c, q, v, \pi) \\ \Downarrow \\ \begin{pmatrix} deg(\phi) \leq \mathsf{D} \wedge \\ v = \phi(q) \wedge \\ [\phi(\mathbf{p'_i}) = \mathbf{e'_i}]^{k'} \end{pmatrix} \end{array} \middle| \begin{array}{c} \mathbf{ck_e}, \mathsf{ck_p}, \mathsf{rk_p}, \mathsf{td} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{D}, \mathsf{K}) \\ [(\mathbf{p'_i}, \mathbf{e'_i})]_{i=1}^{k'} \leftarrow \mathcal{A}_1(\mathbf{ck_e}, \mathsf{ck_p}; z) \\ [c_{e_i} \leftarrow \mathsf{Com_e}(\mathsf{ck_{e_i}}, \mathbf{e'_i}, \mathbf{p'_i}; \boldsymbol{\omega_i})]_{i=1}^{k'} \\ (\mathbf{p}, \mathbf{c_e}, \mathsf{st}) \leftarrow \mathcal{A}_2(\mathbf{ck_e}, \mathsf{ck_p}, \mathbf{c'_e}; z) \\ c \leftarrow \mathsf{Interpolate}(\mathsf{ck_p}, \mathbf{p} || \mathbf{p}', \mathbf{c_e} || \mathbf{c_{e'}}) \\ \phi, \boldsymbol{\omega} \leftarrow \mathcal{E}^{\mathcal{A}_1}(\mathbf{ck_e}, \mathsf{ck_p}, \mathsf{rk_p}; z) \\ v, \pi, q \leftarrow \mathcal{A}_3(\mathbf{ck_e}, \mathsf{ck_p}, \mathsf{rk_p}, \mathbf{c_e}, \mathsf{st}) \end{array}\right]$$

Note that for brevity in the definition, we represent $[c_{e_i} = \mathsf{Commit}_{\mathsf{eval}}(\mathsf{ck_i}, \phi(p_i), p_i; \omega_i)]_{i=1}^k$ as $c_{e_i} = \mathsf{Com_e}(\phi(p_i); \omega_i)$

- **Zero knowledge:** We say that PEC is zero knowledge if the adversary cannot distinguish whether it is interacting with the honest prover or a simulator with trapdoors. More formally, there exists a polynomial-time simulator $\mathsf{S} = (\mathsf{Setup}, \mathsf{Commit}_{\mathsf{eval}}, \mathsf{Open})$ such that, for every maximum degree bound $\mathsf{D}, \mathsf{K} \in \mathbb{N}$, and efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following distributions are indistinguishable:

Real World:

$$\left[\mathsf{st}_2, \pi \middle| \begin{array}{l} \mathsf{ck_e}, \mathsf{ck_p}, \mathsf{rk_p} \leftarrow \mathsf{PEC.Setup}(1^\lambda, \mathsf{D}, \mathsf{K}) \\ [\mathbf{e_i}, \mathbf{p_i}]_{i=1}^k, \mathsf{st}_1 \leftarrow \mathcal{A}_1(\mathsf{ck_e}, \mathsf{ck_p}, \mathsf{rk_p}) \\ [c_{e_i} \leftarrow \mathsf{Commit}_{\mathsf{eval}}(\mathsf{ck_e}, \mathbf{e_i}, \mathbf{p_i}; \boldsymbol{\omega_i})]_{i=1}^k \\ \mathbf{c_e} := [c_{e_i}]_{i=1}^k, \boldsymbol{\omega} := [\boldsymbol{\omega_i}]_{i=1}^k, \phi := \boldsymbol{V_p}(\mathbf{e}, \mathbf{p}) \\ c \leftarrow \mathsf{PEC.Interpolate}(\mathsf{ck_e}, \mathbf{p}, \mathbf{c_e}) \\ q, \mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c) \\ \pi \leftarrow \mathsf{PEC.Open}(ck_p, \phi, q; \boldsymbol{\omega}) \end{array}\right]$$

Ideal World:

$$\left[\mathsf{st}_2, \pi \middle| \begin{array}{l} \mathsf{ck_e}, \mathsf{ck_p}, \mathsf{rk_p}, \mathsf{td} \leftarrow \mathsf{S.Setup}(1^\lambda, \mathsf{D}, \mathsf{K}) \\ [\mathbf{e_i}, \mathbf{p_i}]_{i=1}^k, \mathsf{st}_1 \leftarrow \mathcal{A}_1(\mathsf{ck_e}, \mathsf{ck_p}, \mathsf{rk_p}) \\ [c_{e_i} \leftarrow \mathsf{S.Commit}_{\mathsf{eval}}(\mathsf{ck_e}, \mathbf{p_i}; \boldsymbol{\omega_i})]_{i=1}^k \\ \mathbf{c_e} := [c_{e_i}]_{i=1}^k, \boldsymbol{\omega} := [\boldsymbol{\omega_i}]_{i=1}^k, \phi := \boldsymbol{V_p}(\mathbf{e}, \mathbf{p}) \\ c \leftarrow \mathsf{PEC.interpolate}(\mathsf{ck_e}, \mathbf{p}, \mathbf{c_e}) \\ q, \mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c) \\ \pi \leftarrow \mathsf{S.Open}(ck_{poly}, \mathsf{td}, \phi(q), q; \boldsymbol{\omega}) \end{array}\right]$$

### 4.1. PEC Constructions

We discuss three constructions for PEC schemes. The first, based on Pedersen commitments, is a straightforward approach that involves a commitment to each coefficient of the polynomial. Naturally, this results in commitments and evaluation proofs that are linear in the degree of the polynomials. Even still, when used to instantiate auditable MPC in Section 5, this results in a proof and verification time that is *circuit-succinct* (i.e., independent of the circuit size $M$). We defer the details of this scheme to the Appendix.

Towards constructing an auditable MPC that is additionally *statement-succinct*, our second approach is to adapt an efficient polynomial commitment scheme such as KZG [29]. However, this turns out to be non-trivial. Our first attempt was to simply transport the KZG Commit routine "into the exponent." Briefly, (and ignoring zero-knowledge to illustrate the problem even in the simple case) this involves committing to the evaluation $\phi(i)$ with a group element $g^{\phi(i)}$. However, we then have now way to obtain $g^{\phi(\alpha)}$, the desired KZG polynomial commitment form. We can use the CRS to compute interpolation factors $g^{\ell_i(\alpha)}$ for Lagrange polynomials $\ell_i$, but still we cannot combine these with $g^{\phi(i)}$ to get $g^{\phi(\alpha)}$ without breaking the Computational Diffie Hellman assumption in our group (which KZG relies on). In particular, the CRS does not allow us to compute $\ell_i(\alpha)$ outside the exponent. To solve this problem, our idea is to have each evaluation commitment take the form $g^{\ell_i(\alpha)\phi(i)}$, i.e., to have each party precompute their Lagrange polynomials when

committing to their evaluation points. We also need to ensure that corrupt parties cannot perturb the evaluation points committed by honest parties; we address this by creating separate CRS elements to be used by each party, and proving that each evaluation lies in the span its their assigned CRS elements. Finally our construction incorporates hiding polynomials to maintain zero-knowledge. Our succinct PEC construction, PEC.Succ, is defined as follows:

- $\mathsf{Setup}(1^\lambda, D, K)$ : Sample $\mathbf{\Sigma} = [\Sigma_i]_{i=1}^K$ as follows

$$\Sigma_i := \begin{pmatrix} g^{K*i} & g^{\alpha+K*i} & \dots & g^{\alpha^{n_K-1}+K*i} \\ g^{\gamma_i K*i} & g^{\gamma_i(\alpha+K*i)} & \dots & g^{\gamma_i(\alpha^{n_K-1}+K*i)} \\ g^{\tau K*i} & g^{\tau\alpha+K*i} & \dots & g^{\tau\alpha^{n_K-1}+K*i} \\ g^{\tau\gamma_i K*i} & g^{\tau\gamma_i(\alpha+K*i)} & \dots & g^{\tau\gamma_i(\alpha^{n_K-1}+K*i)} \end{pmatrix} \tag{2}$$

$$\begin{aligned} \mathsf{ck_p} &:= (\Sigma_K, n_K), \mathsf{ck_e} := (\mathbf{\Sigma} = [\Sigma_i]_{i=1}^K), \\ \mathsf{rk_p} &:= (D, g^{\gamma_K}, g^{\tau\gamma_K}, h^\alpha), \mathsf{td} := (\tau, \alpha, [\gamma_i]_0^K) \end{aligned} \tag{3}$$

- $\mathsf{Commit_{eval}}(\mathsf{ck}_{e_i}, \boldsymbol{e_i}, \boldsymbol{p_i}; \omega) \rightarrow c_{e_i}$: With input $\mathsf{ck}_{e_i}, \mathsf{ck}^{\mathsf{k}}_{e_i} = \Sigma_i$, points $\mathbf{p_i} = [p_{ij}]_{j=1}^d$ with evaluations $\mathbf{e_i} = [e_{ij}]_{j=1}^d$, compute computing $\phi$ at atmost degree $d$ by interpolating $\mathbf{e_i}$ at points $\mathbf{p_i}$. Compute $c_e = \mathsf{KZG.Commit}(\mathsf{ck}_{e_i}, \phi; \omega)$. This computes a shifted polynomial commitment using CRS $g^{i*K} \dots g^{i*K+n_K-1}$. Similarly, compute $c_e^k = \mathsf{Commit}(\mathsf{ck}^{\mathsf{k}}_{e_i}, \phi; \omega)$. Return $(c_e, c_e^k)$
- $\mathsf{Interpolate}(\mathsf{ck_p}, \mathbf{c_e}, \mathbf{p}) \rightarrow c$ : Parse $\mathbf{c_e} = [c_{e_i}]$. Check the knowledge component: $\forall i \ (e(g, c_e^k)), g \stackrel{?}{=} e(g_i^\gamma, c_e)$. If check fails, abort, otherwise return $c = \Pi_{i=1}^k c_{e_i}$.
- $\mathsf{Open}(\mathsf{ck_p}, \phi, q; \omega) \rightarrow v, \pi$: Same as KZG polycommit open operation as described in Section 2.4
- $\mathsf{Check}(\mathsf{rk_p}, c, q, v, \pi) \rightarrow \{0,1\}$: Same as KZG polycommit operation as described in Section 2.4.

***Theorem 4.1.*** If $\langle group \rangle$ satisfies the SDH assumption(Appendix C.1), then the above construction for PEC.Succ is a PEC scheme (definition 4)

In Appendix H.2, we provide a proof for the theorem. In Appendix F we show a third PEC scheme that offers concrete performance improvements based on Lipmaa's commitments [34].

## 5. Our Auditable MPC Construction

### 5.1. Adaptive Preprocessing arguments with universal SRS

We first give a formal security definition for Adaptive Marlin, our main construction. Although our final goal is a non-interactive protocol, we follow Chiesa et al. and give an interactive definition and remove interaction with Fiat-Shamir at the end [22]. We use angle brackets $\langle P(...)V(...)\rangle$ to denote the output of $V(...)$ when interacting with $P(...)$.

We extend the indexed relations defined in Section 2.5 to indexed commitment relations. Let $\mathsf{Setup}, \mathsf{Comm}$ be a extractable trapdoor commitment scheme as shown in section C.3. Given indexed relation $\mathcal{R}$ and a commitment key $\mathbf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$

$$\begin{aligned} \mathcal{R}_{\mathbf{ck}} := \{(\mathbf{C_x}, \mathbb{i}, \mathbf{x}, \mathbf{r}, \mathbf{w})) : \\ \forall i.\ C_{x_i} = \mathsf{Comm}(\mathsf{ck}_i, x_i, r_i) \wedge (\mathbb{i}, \mathbf{x}, \mathbf{w}) \in \mathcal{R}\} \end{aligned}$$

Informally, an adaptive preprocessing argument (also refered as adaptive SNARK) for indexed relation is a preprocessing argument for the relation $\mathcal{R}_{\mathbf{ck}}$. We next give the formal definitions for adaptive preprocessing arguments with universal SRS.

Let $\mathsf{C} = (\mathsf{Setup}, \mathsf{Com})$ be an extractable commitment scheme. Further, let $\mathsf{C.Setup}(1^\lambda) \rightarrow \mathbf{ck}, \mathsf{rk}, \mathsf{td}$. We define an adaptive SNARKs (referred as preprocessing arguments with universal SRS in Marlin) for extractable trapdoor commitment scheme $\mathsf{C}$ and relation $\mathsf{R_{ck}}$ as a tuple of four algorithms $(\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V})$:

- $\mathsf{G}(1^\lambda, \mathsf{N}) \rightarrow \mathsf{srs}, \mathsf{td}$: generator $\mathsf{G}$ is a ppt which when given a size bound $\mathsf{N} \in \mathbb{N}$, outputs an srs that supports indices of size up to $\mathsf{N}$ and a trapdoor $\mathsf{td}$.
- $\mathsf{I}^{\mathsf{srs}}(\mathbb{i}) \rightarrow \mathsf{ipk}, \mathsf{ivk}$: The indexer $\mathsf{I}$ is a deterministic algorithm that with oracle access to srs takes in a circuit index $\mathbb{i} < \mathsf{N}$ outputs proving key ipk and verification key ivk specific to the index $\mathbb{i}$.
- $\mathsf{P}(\mathsf{ipk}, \mathbf{ck}, \mathbf{C_x}, \mathbf{x}, \mathbf{r}, \mathbf{w}) \rightarrow \pi$: The prover $\mathsf{P}$ is a ppt which on input prover key ipk, committer keys $\mathbf{ck}$, statement $\mathbf{x}$, commitment randomness $\mathbf{r}$ and witness $\mathbf{w}$ outputs a proof $\pi$.
- $\mathsf{V}(\mathsf{ivk}, \mathsf{rk}, \mathbf{C_x}, \pi) \rightarrow \{0,1\}$ : Verifier $\mathsf{V}$ is a ppt which upon input index verification key ivk, receiver key rk, polynomial evaluation commitments $\mathbf{C_x}$ and a proof $\pi$ outputs either 0 or 1.

Furthermore, we want adaptive preprocessing agruments to satisfy the following properties:

- Perfect Completeness: We say that our adaptive preprocessing argument is complete if all adversaries choosing the a tuple $(\mathbf{C_x}, \mathbb{i}, \mathbf{x}, \mathbf{r}, \mathbf{w}) \in \mathcal{R}_{\mathbf{ck}}$, the interaction between honest prover is always able to convince the honest verifier.

$$\Pr\left[\begin{array}{c|c} (\mathbf{C_x}, \mathbb{i}, \mathbf{x}, \mathbf{r}, \mathbf{w}) \notin \mathcal{R}_{\mathbf{ck}} & \mathsf{srs} \leftarrow \mathsf{G}(1^\lambda, \mathsf{N}) \\ \vee & \mathbf{ck} \leftarrow \mathsf{C.Setup}(1^\lambda) \\ \left\langle \mathsf{P}(\mathsf{ipk}, \mathbf{ck}, \mathbf{C_x}, \mathbf{x}, \mathbf{r}, \mathbf{w}) \right\rangle & (\mathbf{C_x}, \mathbb{i}, \mathbf{x}, \mathbf{r}, \mathbf{w}) \leftarrow \mathcal{A}(\mathsf{srs}) \\ \mathsf{V}(\mathsf{ivk}, \mathsf{rk}, \mathbf{C_x}) & \mathsf{ipk}, \mathsf{ivk} \leftarrow \mathsf{I}^{\mathsf{srs}}(\mathbb{i}) \end{array}\right]$$

- Extractable: We say that our adaptive preprocessing argument is extractable if for every size bound $\mathsf{N} \in \mathbb{N}$ and efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists an efficient extractor $\mathcal{E}$ such that the following probability is $\mathsf{negl}(\lambda)$.

$$\Pr\left[\begin{array}{c|c} & \mathsf{srs} \leftarrow \mathsf{G}(1^\lambda, \mathsf{N}) \\ (\mathbf{C_x}, \mathbb{i}, \mathbf{x}, \mathbf{r}, \mathbf{w}) \notin \mathcal{R}_{\mathbf{ck}} & \mathbf{ck} \leftarrow \mathsf{Setup}(1^\lambda) \\ \wedge & (\mathbf{C_x}, \mathbb{i}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{srs}, \mathbf{ck}; \mathsf{z}) \\ \langle \mathcal{A}_2(\mathsf{st}), \mathsf{V}(\mathsf{ivk}, \mathsf{rk}, \mathbf{C_x}) \rangle & (\mathbf{x}, \mathbf{r}, \mathbf{w}) \leftarrow \mathcal{E}^{\mathcal{A}_1, \mathcal{A}_2}(\mathsf{srs}; \mathsf{z}) \\ & \mathsf{ipk}, \mathsf{ivk} \leftarrow \mathsf{I}^{\mathsf{srs}}(\mathbb{i}) \end{array}\right]$$

- Zero Knowledge: We say that our adaptive preprocessing argument is zero knowledge if the adversary is not able to distinguish whether it is interacting with a honest prover or a simulator.

More formally, ARG is zero knowledge if for every size bound $\mathsf{N} \in \mathbb{N}$ there exists a simulator $\mathsf{S} = (\mathsf{Setup}_1, \mathsf{Setup}_2, \mathsf{Prove})$ such that for every efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ the probabilities shown below are equal

$$\Pr\left[\begin{array}{c|c} (\mathbf{C_x}, \mathbb{i}, \mathbf{x}, \mathbf{r}, \mathbf{w}) \in \mathcal{R}_{\mathbf{ck}} \\ \wedge \\ \left\langle \mathsf{P}(\mathsf{ipk}, \mathbf{ck}, \mathbf{C_x}, \mathbf{x}, \mathbf{r}, \mathbf{w}) \right\rangle \\ \mathcal{A}_2(\mathsf{st}) \end{array} \middle| \begin{array}{c} \mathsf{srs} \leftarrow \mathsf{G}(1^\lambda, \mathsf{N}) \\ \mathbf{ck} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathbb{i}, \mathbf{x}, \mathbf{r}, \mathbf{w}, \mathsf{st} \leftarrow \mathcal{A}_1(\mathsf{srs}, \mathsf{ck}) \\ \mathsf{ipk}, \mathsf{ivk} \leftarrow \mathsf{I}^{\mathsf{srs}}(\mathbb{i}) \\ \mathbf{C_x} \leftarrow [\mathsf{Com}(\mathsf{ck_i}, \mathsf{x_i}, \mathsf{r_i}))]_{i=1}^{|\mathbf{x}|} \end{array}\right]$$

$$\Pr\left[\begin{array}{c|c} (\mathbb{i}, \mathbf{x}, \mathbf{r}, \mathbf{w}) \in \mathcal{R}_{\mathsf{ck}} \\ \wedge \\ \left\langle \mathsf{S.Prove}(\mathsf{td_s}, \mathsf{td_c}, \mathbf{C_x}, \mathbb{i}) \right\rangle \\ \mathcal{A}_2(\mathsf{st}) \end{array} \middle| \begin{array}{c} \mathsf{srs}, \mathsf{td_s} \leftarrow \mathsf{S.Setup}_1(1^\lambda, \mathsf{N}) \\ \mathbf{ck}, \mathsf{td_c} \leftarrow \mathsf{S.Setup}_2(1^\lambda) \\ \mathbb{i}, \mathbf{x}, \mathbf{r}, \mathbf{w}, \mathsf{st} \leftarrow \mathcal{A}_1(\mathsf{srs}, \mathsf{ck}) \\ \mathsf{ipk}, \mathsf{ivk} \leftarrow \mathsf{I}^{\mathsf{srs}}(\mathbb{i}) \\ \mathbf{C_x} \leftarrow [\mathsf{Com}(\mathsf{ck_i}, \mathsf{x_i}, \mathsf{r_i}))]_{i=1}^{|\mathbf{x}|} \end{array}\right]$$

**Adaptive Zk-Snark based on Marlin**

Let $\mathsf{PEC} = \mathsf{Setup}, \mathsf{Commit}_{\mathsf{eval}}, \mathsf{Interpolate}, \mathsf{Open}, \mathsf{Check}$ and $\mathsf{PEC.Setup}(1^\lambda, D) \rightarrow \mathsf{ck_e}, \mathsf{ck_p}, \mathsf{rk_p}$. Let $(\mathsf{G_m}, \mathsf{I_m}, \mathsf{P'_m}, \mathsf{V'_m})$ be a augmented pre-processing argument constructed from Marlin for the relation $\mathcal{R}'_{\mathbf{ck}}$. Let $\mathsf{P_m}$ be same as $\mathsf{P'_m}$ with the difference that it knows public commitment $\mathbf{C_x}$ along with the secret $\mathbf{x}$. Similarly, let $\mathsf{V_m}$ be the same as $\mathsf{V'_m}$ with the difference that $\mathsf{V'_m}$ only knows $\mathbf{C'_x}$ instead of $\mathbf{x}$.

- $\mathsf{G}(\mathsf{N})$: If $\mathsf{N} > \mathsf{D}$, abort. Return $\mathsf{srs} = \Sigma$ from $\mathsf{ck_e}$
- $\mathsf{I}^{\mathsf{srs}}(\mathbb{i}) \rightarrow (\mathsf{ipk}, \mathsf{ivk})$: Let $\mathbb{i}' = (\mathbb{F}, n+1, m, A', B', C')$ where $(A', B', C') = \mathsf{Pad}(A, B, C)$. Return $\mathsf{I}^{\mathsf{srs}}_{\mathsf{m}}(\mathbb{i}')$.
- $\mathsf{P}(\mathsf{ipk}, \mathsf{ck_e}, \mathbf{C_x}, \mathbf{x}, \mathbf{r}, \mathbf{w}) \rightarrow \pi$: Sample $x^{\mathsf{b}} \xleftarrow{\$} \mathbb{F}$; $\mathbf{C}^{\mathsf{b}}_{\mathbf{x}} \leftarrow \mathsf{Commit}_{\mathsf{eval}}(\mathsf{ck_e}, x^{\mathsf{b}}, .; \omega)$. $\mathbf{C'_x} = (\mathbf{C_x}, \mathbf{C}^{\mathsf{b}}_{\mathbf{x}})$; $\mathbf{x}' = \mathbf{x}||x^{\mathsf{b}}$. call $\mathsf{P_m}(\mathsf{ipk}, x||x^{\mathsf{b}}, w) \rightarrow \pi_m$. Let $\hat{x}'(X) = V_{\mathbf{p}}\mathbf{x}'$, $\beta_1$ be second verifier challenge in Marlin execution. $\mathsf{PEC.open}(\mathsf{ck_p}, \hat{x}'(X), \beta_1; \omega) \rightarrow \hat{x}'(\beta_1), \pi_c$. return $\pi = (\pi_m, \pi_c, \hat{x}'(\beta_1), \mathbf{C}^{\mathsf{b}}_{\mathbf{x}})$.
- $\mathsf{V}(\mathsf{ivk}, \mathsf{rk_p}, \mathbf{C_x}, \pi)$: Parse $\pi = (\pi_m, \pi_c, \hat{x}'(\beta_1), \mathbf{C}^{\mathsf{b}}_{\mathbf{x}})$; $\mathbf{C'_x} = (\mathbf{C_x}, \mathbf{C}^{\mathsf{b}}_{\mathbf{x}})$ Invoke the Marlin verifier routine by replacing $\hat{x}$ with constant function $\hat{x}(\beta_1)$ as $b_m \leftarrow \mathsf{V_m}(\mathsf{ivk}, \hat{x}'(\beta_1), \pi_m)$. $C_{\hat{x}} \leftarrow \mathsf{PEC.Interpolate}(\mathsf{ck_p}\mathbf{C'_x}, .)$ Invoke $b_c \leftarrow \mathsf{PEC.check}(\mathsf{vk_{poly}}, C_{\hat{x}'}, \beta_1, \hat{x}'(\beta_1), \pi_c)$.

  return $b_m \stackrel{?}{=} 1$ and $b_c \stackrel{?}{=} 1$.

Figure 2. Adaptive preprocessing arguments using Marlin

## 5.2. Construction of Adaptive Preprocessing arguments with Universal SRS

Our construction closely follows Marlin's except for two main modifications to the underlying Algebraic Holographic Proof (AHP). The full dsecription of Marlin is in Appendix G, so here we only highlight the differences. In the Marlin prover algorithm, the verifier is assumed to have the entire statement $\mathbf{x}$ and hence it can construct for itself $\hat{x}$, the polynomial encoding of the statement (querying this polynomial at random challenge points is roughly what makes the scheme "holographic). In our setting, the verifier does not have $\mathbf{x}$, only a commitment to it $\mathbf{C_x}$, so the prover must additionally supply $\hat{x}$. We must check that the prover supplied $\hat{x}$ matches the commitment $\mathbf{C_x}$, which can be addressed using PEC. Additionally, statement $\mathbf{x}$ must be kept zero knowledge. We can achieve this the same way as Marlin keeps the witness zero knowledge, namely by padding the degree of $\hat{x}$ by a margin of $\mathsf{b}$ so that learning $\mathsf{b}$ challenge points of $\hat{x}$ reveals nothing about $\mathbf{x}$. As with Marlin, it suffices to set $\mathsf{b} = 1$, but we stick to $\mathsf{b}$ for consistency of notation.

In more detail, we consider an augmented relation $\mathcal{R}'_{\mathbf{ck}} = \{(\mathbf{C'_x}, \mathbb{i}', \mathbf{x}', \mathbf{r}', \mathbf{w})\}$: $(\mathbf{C_x}, \mathbb{i}, \mathbf{x}, \mathbf{r}, \mathbf{w}) \in \mathcal{R}_{\mathbf{ck}}$, $\mathbf{C'_x} = \mathbf{C_x}||\mathbf{C}^{\mathsf{b}}_{\mathbf{x}}$, $\mathbb{x}' = \mathbf{x}||x^{\mathsf{b}}$, $\mathbf{r}' = \mathbf{r}||r^{\mathsf{b}}$, $\mathbf{C'_x} = \mathsf{Commit}_{\mathsf{eval}}(\mathsf{ck}_{|\mathsf{x}|+1}, x^{\mathsf{b}}, r^{\mathsf{b}})$ and $\mathbb{i}' = \mathsf{Pad}(\mathbb{i})$ by padding $\mathbb{i}$ with dummy constraint. In more detail, let $\mathbb{i} = (\mathbb{F}, n, m, A, B, C)$ such that $\mathbf{z} := (\mathbf{x}, \mathbf{w}) \in \mathbb{F}^n$ such that $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z}$, compute $\mathbb{i}' = (\mathbb{F}, n+1, m, A', B', C')$ $\mathbf{z}' := (\mathbf{x}', \mathbf{w})$ is a vector in $\mathbb{F}^{n+1}$ such that $A'\mathbf{z}' \circ B'\mathbf{z}' = C'\mathbf{z}'$. This is done by padding matrices $A, B, C$ with dummy constraint $(0 \times 0 = 0)$ on free variable $x^{\mathsf{b}}$ to obtain $A', B', C'$. In simpler words, we add a free statement variable to the indexed constraint system.

Finally, we use the compiler from Marlin to compile the above modified AHP and polynomial commitment scheme from Marlin(different from PEC) to result in pre-processing arguments that are adaptive.

Our construction for an adaptive Preprocessing arguments $\mathsf{ARG} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V})$ with universal SRS for extractable trapdoor commitment scheme PEC and relation $\mathsf{R_{ck}}$ is shown in 2.

We state our construction with a generic PEC, it is possible to instantiate with any of PEC.Ped, PEC.Lipmaa or PEC.Succ. Our routine Generator G uses the same srs from the PEC scheme whereas our Indexer I operates on the $\mathbb{i}'$. Our prover algorithm first samples additional element $x^{\mathsf{b}} \in \mathbb{F}_q$ element to compute the augmented statement $\mathbf{x}'$. First, the prover computes $\mathbf{C}^{\mathsf{b}}_{\mathbf{x}} \leftarrow \mathsf{PEC.Commit}_{\mathsf{eval}}(\mathsf{PEC.ck}_{\mathsf{eval}}, \mathsf{x}^{\mathsf{b}}, p_{|x|+1}; \omega)$ to compute the evaluation commitment at point with index $|x|+1$ keeping the randomness $\omega$. It then runs the modified Marlin prover $\mathsf{P_m}(\mathsf{ipk}, \mathbf{x}', \mathbf{w})$ to obtain a proof $\pi_m$. Let $\hat{x}'$ be the low degree extension(LDE) of $\mathbf{x}'$ and $\beta_1$ be the second round challenge in the Marlin protocol. Then, our prover routine computes $\mathsf{PEC.Open}(\mathsf{ck_p}, \hat{x}', \beta_1, \omega)$ obtain a evaluation $\hat{x}'(\beta_1)$ and opening proof $\pi_c$. Finally, the proof is returned as $\pi = (\pi_m, \pi_c, \hat{x}'(\beta_1), \mathbf{C}^{\mathsf{b}}_{\mathbf{x}})$

The auditor reconstructs the augmented statement from $\mathbf{C'_x} = (\mathbf{C_x}, \mathbf{C}^{\mathsf{b}}_{\mathbf{x}})$ and verifies that $\pi_m$ and $\pi_c$ are correct.

***Theorem 5.1.*** If PEC is a extractable PEC scheme then the above construction $\mathsf{ARG} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V})$ is an adaptive zkSNARK according to definition 5.1.

We prove this theorem in Appendix H.3

### 5.3. Prover algorithm using MPC

Next, we describe how to implement the above prover P algorithm by using MPC where the servers compute the proof from shares of the witness and statement. Our prover, just like Marlin, proceeds in multiple rounds. In each round, the verifier sends some challenges and the prover responds back commitments to polynomials. After the conclusion of rounds, the prover provides the evaluations proofs of the committed polynomials. In the MPC setting, this translates to servers knowing shares of the polynomials and computing the evaluation proofs through MPC. We provide our constructions for MPC versions of $\mathsf{PC.Open_{MPC}}$, $\mathsf{PC.Commit_{MPC}}$ and $\mathsf{Eval_{MPC}}$ (Evaluate a polynomial) in Appendix E.

We note that only the first two rounds in Marlin rely on secret shared values; the last two rounds operate on public values and so do not require MPC. To initialize the prover, the servers first compute the commitment to the statement polynomial from shares of that polynomial using $\mathsf{PC.Commit_{MPC}}$. We again use protocol $\mathsf{PC.Commit_{MPC}}$ for the first two Marlin rounds to obtain commitments to polynomials from two rounds from their respective secret shares. After the rounds are concluded, the servers provide evaluations the secret shared polynomials using $\mathsf{Eval_{MPC}}$ and provide an evaluation proof using $\mathsf{PC.Open_{MPC}}$. Full details are in Appendix G.

### 5.4. Construction of auditable MPC

Figure 3 shows the auditable MPC protocol using the constructions for adaptive zk-SNARKS, Marlin based MPC prover and Polynomial Evaluation commitment schemes as the underlying commitment scheme. In Appendix I we show a UC [35] proof that the construction follows the functionality shown previously in Figure 8.

In Step 1, a trusted party performs the setup for PEC(same as the setup for Marlin) the evaluation commitment keys $\mathsf{ck_e}$ to input parties, verification key $\mathsf{vk_p}$ for the polynomial commitment scheme to auditor and $\mathsf{ck_p}$ to the MPC servers. Each input client $I_i$ samples an input $x_i$ with randomness $r_i$ and creates a commitment $C_{x_i} = \mathsf{Commit_{eval}}(\mathsf{ck_{e_i}}, x_i, r_i)$. Because each client has it's own committer key, we can gurantee input independence (Refer H.4) as clients can only use thier own commitment keys. Next, the input client $I_{inp}$ provides the computation function $f$ to MPC system. For simplicity, we assume that computation $f$ itself specifies the desired R1CS computation. If the circuit is not already preprocessed, the MPC servers compute the indexer proving and verification keys, $\mathsf{index_{pk}}$ and $\mathsf{index_{vk}}$ and post them on the bulletin board.

In step 4, the clients again submit the inputs to the MPC system $[\![x_i]\!]$ and randomness $[\![r_i]\!]$. The servers check that the $C_{x_i}$ is consistent with the shares $[\![x_i]\!]$ and $[\![r_i]\!]$ using MPC polycommit check as listed in Appendix E. If the commitments do not match, the servers abort the computation. Looking in the UC proof (Appendix I) for $f < t$ case, this aborting helps us guarantee Zero knowledge property as the auditor might learn the instance is under Indexed relations with Commitments 2.5.

**Auditable MPC using universal SRS**

Consider Auditor $A$, $P_1, \ldots, P_n$ servers with preprocessed values, data-clients $I_1, I_2, \ldots, I_m$ with input $\mathbf{x_i} \in \mathbb{F}$ respectively, Input-Client $I_{inp}$ with $f$ and bulletin board $BB$

1) Perform PEC.Setup for the PEC scheme and universal CRS, send $\mathsf{ck_{e_i}}$ to $I_i$, $\mathsf{vk_p}$ to $A$ and the crs, $\mathbf{ck_e}$, $\mathsf{ck_p}$ to the all servers $P_j$.
2) Each $I_i$ commits $C_{x_i}$ as $\mathsf{PEC.Commit_{eval}}(\mathsf{ck_{e_i}}, x_i, r_i)$ and post it to $BB$.
3) $I_{inp}$ provides $f$ to servers all servers $P_j$. Servers run Indexer I to compute ipk, ivk for $f$.
4) Each $I_i$ provides inputs $[\![x_i]\!]$ and randomness $[\![r_i]\!]$. Check $C_{x_i} \stackrel{?}{=} \mathsf{PEC.Commit_{eval}}(\mathsf{ck_{e_i}}, [\![x_i]\!], [\![r_i]\!])$ using MPC polycommit protocols Appendix E.1. Abort if check fails.
5) Carry out the MPC protocol $(o_1, \ldots, o_k) = f(x_1, \ldots, x_m, y_1, \ldots, y_l)$. Computing witness $\mathbf{w}$ and commit outputs $o_1 \ldots o_k$ using $\mathsf{PEC.Commit_{eval}}$ as $[C_o]_{i=1}^k$ and post to $BB$.
6) Run $\mathsf{P}(\mathsf{ipk}, \mathsf{ck_e}, \mathbf{C_{stmt}}, \mathbf{x}, \mathbf{r}, \mathbf{w}) \rightarrow \pi$ in MPC protocol 5.3 where $\mathbf{C_{stmt}} = ([C_{x_i}]_{i=1}^{m+l}, [C_o]_{i=1}^k)$. Post $\pi$
7) The auditor gets $C_{x_i}$, $C_{o_k}$, ivk, proof $\pi$ from $BB$ and recomputes $\mathbf{C_{stmt}}$ and verifies proof as $\mathsf{V}(\mathsf{ivk}, \mathsf{rk_p}, \mathbf{C_{stmt}}, \pi)$ as shown in Fig 2.

Figure 3. Auditable MPC with universal SRS

In Step 5, the servers compute the MPC operation to get the output of the computation $o_k$ and post evaluation commitments to the bulletin board. These can be treated as inputs for the next round of MPC computations, allowing us the reactive functionality. Finally, the MPC servers compute a adaptive Snark proof $\pi$ by using our prover algorithm construction detailed in Figure 2 and post it on the bulletin board. The auditor collects the evaluation commitments to the inputs and outputs from the bulletin board to construct the commitment to $\hat{x}$. The auditor then uses the verify routine in from our adaptive Snark construction as listed in Figure 2.

## 6. Evaluation

Our rust implementation for auditable MPC based on Marlin [36] can found at https://github.com/randomcyrptobuddy/auditablempc.

We simulate the MPC behaviour as described in section 2.2 by using artificial delays(Appendix A) in communication latency of 200ms and a uplink speed of 200Mbits/per second [4]. We report our performance numbers on a single thread machine with a modern CPU processor with 1200 MHz and use the bls12-381 curve[37] for pairing friendly and fft optimizations. For all the experiments, we use the faster PEC.Lipmaa (Appendix F) version of PEC scheme. For sampling random R1CS circuits with $m$

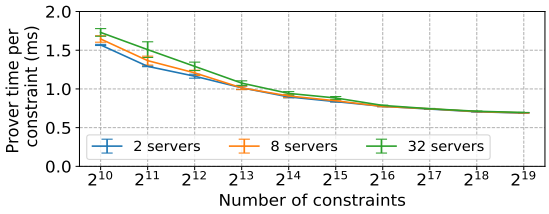


Figure 4. Prover cost per constraint as a function of number of constraints. The different lines indicate number of MPC servers. Error bars report 95% confidence interval over 10 trials for constraints less than $2^{16}$, 3 trails for $2^{17}$, $2^{18}$ and single trail stretch run for $2^{19}$ and $2^{20}$.

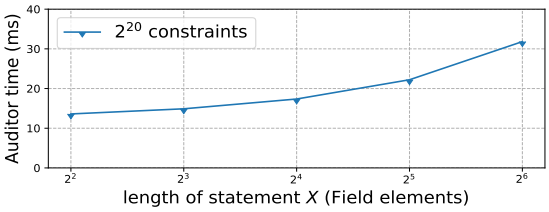


Figure 5. Auditor cost as a function of statement length. Values are averaged over 10 iterations

gates, we sample matrices $A$, $B$ and solve for $C$ to obtain a solvable QAP of $m$ constraints.

We first report the performance of our construction over random circuits in terms of prover cost, auditor cost, communication cost, proof size. We vary our statement size from $[2^2, 2^6]$, our constraint size from $[2^{10}, 2^{20}]$ and number of MPC servers from $[2^2, 2^5]$. In the Appendix we also implement and evaluate two applications: 1) Publicly auditable auction and 2) Logrank Test from Vee'17 [17] for direct comparison with previous work.

**6.0.1. Prover cost.** Server computation consists of two main components: 1) Computing the witness and output wire values at each server and 2) computing the Marlin proof by doing another MPC amongst the servers. When we report prover time benchmarks, we only consider the time for the second component. As our prover works in round-wise synchronous fashion, we consider a round time for MPC as the worst time amongst all provers in that round. As shown in Figure 4 prover cost decreases logarithmically ($\frac{1}{\log m}$). The different lines show the prover time with different number of servers. Because of the linear communication cost in statement size, the prover time overhead due to the extra communication cost is marginal compared to the Marlin prover cost. This is the reason why the different lines start separate and eventually converge with additional constraints.

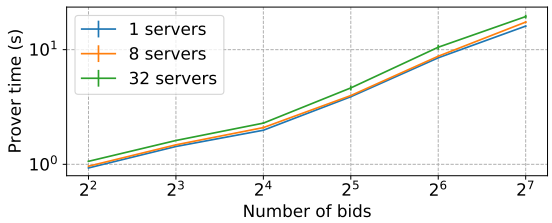


Figure 6. Auction Application Results: Prover cost as a function of number of bids for the auction application. Multiple lines denote the number of MPC servers involved in proof creation. Error bars show 95% Confidence intervals over 10 iterations.

**6.0.2. Auditor cost.** The auditor computation mainly consists of two parts: 1) Verifying the marlin proof and 2) Carrying out the input consistency check. The first involves a constant number of pairings to verify that the claimed evaluations are consistent with the commitments, while the second involves interpolating a polynomial in the exponent which is linear in statement size. Neither of these computations depend on the number of MPC servers used, or on the circuit size. Figure 5 shows the auditor cost as a function of statement length.[1]

**6.0.3. Proof size.** Our auditable MPC proof is just a Marlin proof combined with statement commitments. Normally in SNARKS, the statement is not considered a part of proof because the verifier is assumed to have access to it. Similarly when reporting proof size, we assume that auditor already has client input and server output commitments. Concretely speaking our proof size is 17 $\mathcal{G}_1$ elements and 23 $\mathbb{F}_q$ elements which for bls12-381 [37] corresponds to 1.5 KB (1552 bytes). (See Table 3 for details)

**6.0.4. Communication Cost.** Our construction involves sequential(round-wise) communication; one round for checking client inputs are correct against commitments and four additional rounds for the marlin prover algorithm. Recall that before executing the prover, the servers execute an MPC to check whether the inputs are consistent with the commitments provided by the clients. This step involves opening $|X|$ partial commitments while Marlin prover incurs an additional 9 openings in $\mathbb{G}_1$ and 9 openings in $\mathbb{F}_q$ across 4 sequential rounds where a round communication between $n$ parties. For bls381 curve, with $n = 32$ and a naive broadcast reconstruction algorithm, the total communication cost across all servers amounts to $\approx 700$ KB.

## 7. Conclusion and future work

Our work[2] shows that public auditability can be practically added to existing MPC protocols, by adapting Marlin, a SNARK construction with universal trusted setup, to the MPC setting. Future work is to explore other SNARKs with different tradeoffs, including transparent SNARKs that avoid trusted setup altogether. Also in our prototype implementation we had to write our application program twice: once for MPC and again for the SNARK that checks the MPC's work. Publicly auditable MPC would benefit from a unified programming framework that targets both MPC and SNARKs in one program.

1. The x-label on graph shows statement size of $2^k$, it is actually $2^k$ - 2

2. Acknowledgements: This work was partially supported by NSF awards #1801321 and #1943499.

## References


[1] F. Massacci, C. N. Ngo, J. Nie, D. Venturi, and J. Williams, "Futuresmex: secure, distributed futures market exchange," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 335–353.

[2] J. Cartlidge, N. P. Smart, and Y. Talibi Alaoui, "Mpc joins the dark side," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, 2019, pp. 148–159.

[3] N. Alexopoulos, A. Kiayias, R. Talviste, and T. Zacharias, "Mcmix: Anonymous messaging via secure multiparty computation," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 1217–1234.

[4] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller, "Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 887–903.

[5] A. Lapets, N. Volgushev, A. Bestavros, F. Jansen, and M. Varia, "Secure mpc for analytics as a web application," in *2016 IEEE Cybersecurity Development (SecDev)*. IEEE, 2016, pp. 73–74.

[6] A. Rajan, L. Qin, D. W. Archer, D. Boneh, T. Lepoint, and M. Varia, "Callisto: A cryptographic approach to detecting serial perpetrators of sexual misconduct," in *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, 2018, pp. 1–4.

[7] Z. J. Williamson, "The aztec protocol," *URL: https://github.com/AztecProtocol/AZTEC*, 2018.

[8] M. Keller, "Mp-spdz: A versatile framework for multi-party computation." *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 521, 2020.

[9] K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell, and A. Nof, "Fast large-scale honest-majority mpc for malicious adversaries," in *Annual International Cryptology Conference*. Springer, 2018, pp. 34–64.

[10] X. Wang, S. Ranellucci, and J. Katz, "Global-scale secure multiparty computation," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 39–56.

[11] A. Barak, M. Hirt, L. Koskas, and Y. Lindell, "An end-to-end system for large scale p2p mpc-as-a-service and low-bandwidth mpc for weak participants," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 695–712.

[12] I. Abraham, B. Pinkas, and A. Yanai. (2020) Blinder–mpc based scalable and robust anonymous committed broadcast.

[13] I. Damgård, M. Geisler, M. Krøigaard, and J. B. Nielsen, "Asynchronous multiparty computation: Theory and implementation," in *International workshop on public key cryptography*. Springer, 2009, pp. 160–179.

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

[15] M. Stadler, "Publicly verifiable secret sharing," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1996, pp. 190–199.

[16] C. Baum, I. Damgård, and C. Orlandi, "Publicly auditable secure multi-party computation," in *International Conference on Security and Cryptography for Networks*. Springer, 2014, pp. 175–196.

[17] M. Veeningen, "Pinocchio-based adaptive zk-snarks and secure/correct adaptive function evaluation," in *International Conference on Cryptology in Africa*. Springer, 2017, pp. 21–39.

[18] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 238–252.

[19] S. Bowe, A. Gabizon, and I. Miers, "Scalable multi-party computation for zk-snark parameters in the random beacon model." *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 1050, 2017.

[20] S. Bowe, A. Gabizon, and M. D. Green, "A multi-party protocol for constructing the public parameters of the pinocchio zk-snark," in *International Conference on Financial Cryptography and Data Security*. Springer, 2018, pp. 64–77.

[21] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza, "Secure sampling of public parameters for succinct zero knowledge proofs," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 287–304.

[22] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zksnarks with universal and updatable srs," Cryptology ePrint Archive, Report 2019/1047, 2019, https://eprint. iacr. org ..., Tech. Rep., 2019.

[23] M. Campanelli, D. Fiore, and A. Querol, "Legosnark: modular design and composition of succinct zero-knowledge proofs," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2075–2092.

[24] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," *Journal of the ACM (JACM)*, vol. 39, no. 4, pp. 859–868, 1992.

[25] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[26] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Annual International Cryptology Conference*. Springer, 1991, pp. 420–432.

[27] Z. Beerliová-Trubíniová and M. Hirt, "Perfectly-secure mpc with linear communication complexity," in *Theory of Cryptography Conference*. Springer, 2008, pp. 213–230.

[28] I. Damgård and J. B. Nielsen, "Scalable and unconditionally secure multiparty computation," in *Annual International Cryptology Conference*. Springer, 2007, pp. 572–590.

[29] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2010, pp. 177–194.

[30] G. Fuchsbauer, E. Kiltz, and J. Loss, "The algebraic group model and its applications," in *Annual International Cryptology Conference*. Springer, 2018, pp. 33–62.

[31] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *2017 IEEE Symposium on Security and Privacy (SP)*. Ieee, 2017, pp. 444–460.

[32] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers, "Updatable and universal common reference strings with applications to zk-snarks," in *Annual International Cryptology Conference*. Springer, 2018, pp. 698–728.

[33] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy, "Checking computations in polylogarithmic time," in *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, 1991, pp. 21–32.

[34] H. Lipmaa, "Prover-efficient commit-and-prove zero-knowledge snarks," *International Journal of Applied Cryptography*, vol. 3, no. 4, pp. 344–362, 2017.

[35] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 136–145.

[36] "Marlin: rust library for preprocessing zksnarks," 2019. [Online]. Available: https://github.com/scipr-lab/marlin

[37] S. Bowe. Faster subgroup checks for bls12-381.

[38] M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele, "Secure computation on floating point numbers." in *NDSS*, 2013.

[39] S. J. A. de Hoogh. (2012) Design of large scale applications of secure multiparty computation: secure linear programming.

[40] N. Mantel, "Evaluation of survival data and two new rank order statistics arising in its consideration," *Cancer Chemother. Rep.*, vol. 50, pp. 163–170, 1966.

[41] D. Boneh and X. Boyen, "Short signatures without random oracles," in *International conference on the theory and applications of cryptographic techniques*. Springer, 2004, pp. 56–73.

[42] D. J. Bernstein. (2002) Pippenger[3]s exponentiation algorithm.

[43] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual international cryptology conference*. Springer, 1991, pp. 129–140.

[44] L. Rotem and G. Segev. Algebraic distinguishers: From discrete logarithms to decisional uber assumptions.

[45] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward, "Aurora: Transparent succinct arguments for r1cs," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2019, pp. 103–128.

# Appendix A. Applications:

For 32 servers with circuits over 1 million multiplication gates, the pre-processing takes less than a minute. The benchmarks do not consider the offline cost of pre-processing as it can be carried out previously without knowledge of any particular computation description. A full auditable MPC should also carry out the audit the offline phase, but we believe our ideas can be directly combined with Blinder[12] to obtain a full robust specification.

**A.0.1. Auction Application.** We also evaluate our implementation on the motivation auction application functionality listed in section 3.1. Our auction application critically relies on comparison operation for which we implement a comparison circuit. We use techniques from [38][39] to compute a share of $[\![a > b]\!]$ and prove that it was computed correctly.

Our application proceeds in a reactive manner processing a fixed of number of bids $k$ each round. At the end of each round, the servers maintain a secret shared state of the highest bid which can be then used as an input for subsequent rounds. Figure 6 show the prover cost for the auction application when varied across $k$, the number of bids processed in one round. As expected, the prover cost increases almost linearly because an increase $k$ results in a linear increase in the number of constraints. Finally, the auditor cost exhibits a similar graph as figure 5.

**A.0.2. Logrank Test.** - Table 3 shows a theoretical comparison of our work with Veeningen, while Table 2 shows the performance comparison for the Logrank Test (See Appendix B) involving fixed point operations. $\mathbf{C_x}$ denotes the total number of statement commitments. The first shows performance numbers reported by Veeningen [17].

Our prover cost is empirically close to Veeningen's which is surprising because the Marlin prover is more expensive Pinocchio prover. We think this is likely because of our use of FFT friendly curves like BLS12-381[37]. We expect our prover to be about 3-4 times slower than adaptive SNARKs with circuit specific setup. Our auditor performs significantly better than Veeningen [17] because of constant pairing cost compared to linear pairing cost from Veeningen. Our auditor(PEC.Lipmaa) also incurs a linear cost in group exponentiation operations, but in practice the cost of those operations is small compared to pairing operations.

TABLE 2. PROVER AND AUDITOR COST COMPARISON OF THIS WORK WITH VEENINGEN.

| | | $\lvert\mathbf{C_x}\rvert = 1$ | $\lvert\mathbf{C_x}\rvert = 7$ | $\lvert\mathbf{C_x}\rvert = 175$ |
|---|---|---|---|---|
| Vee'17 | Prove | 0.4s | 3.2s | 73.5s |
| | Audit | 0.0s | 0.3s | 4.9s |
| This work | Prove | 1.46 ± 0.02s | 2.23 ± 0.02 | 85.74 ± 0.98s |
| | Audit | 12.4 ± 0.04 ms | 18.8 ± 0.42 ms | 40.5 ± 0.37 ms |

# Appendix B. Logrank

## B.1. Logrank Test

Mantel-Haenzel Logrank test[40] is a statistical test to decide whether there is a significant difference in survival rate between the two populations. It is widely used in clinical trials to establish the efficacy of a new treatment in comparison with a control treatment. The survival data about a population is represented by a set of tuples $(n_j, d_j)$, where $n_j$ is the number of patients still in the study just before time $j$ and $d_j$ is the number of deaths at time $j$. The populations are distributed across multiple hospitals and each hospital commits to it's value of $(n_j, d_j)$ tuple.

The null hypothesis for the logrank test, i.e., the distributions represent the same "survival function", corresponds to $X \sim \chi^2_1$. This null hypothesis is rejected if $1 - \mathsf{cdf}(X) > \alpha$, where cdf is the cumulative density function of the $\chi^2_1$ distribution. Logrank test involves fixed point operations of division, multiplication. Similar to Geppetri [17], we use MPC to compute $X$, and then apply the cdf in the clear. Figure B.1.1 show the algorithms implemented. Veeningen also had a block-size parameter that controlled a a tradeoff between prover cost and auditor cost. Their total cost total cost for the application across all blocks, in table 2 we report the performance numbers in per-block basis for a more direct comparison of programs of similar complexity.

**B.1.1. Logrank Algorithm.**

$$E_{j,1} = \frac{(d_{j,1} + d_{j,2}) \cdot n_{j,1}}{n_{j,1} + n_{j,2}}$$

$$V_j = \frac{n_{j,1} n_{j,2} \left(d_{j,1} + d_{j,2}\right) \left(n_{j,1} + n_{j,2} - d_{j,1} - d_{j,2}\right)}{\left(n_{j,1} + n_{j,2}\right)^2 \cdot \left(n_{j,1} + n_{j,2} - 1\right)}$$

$$X = \frac{\sum_j E_{j,1} - \sum_j d_{j,1}}{\sum_j V_j}$$

**Algorithm 1** Logrank computation for each time step

**Require:** $[\![\mathbf{d}_{i,1}]\!]$ , $[\![\mathbf{d}_{i,1}]\!]$ , $[\![\mathbf{d}_{i,2}]\!]$ , $[\![\mathbf{n}_{i,1}]\!]$ , $[\![\mathbf{n}_{i,2}]\!]$ survival data at time point $i$
**Ensure:** $([\![e_i]\!]^f, [\![v_i]\!]^f, [\![d_i]\!])$ contributions to $\sum_j E_{j,1}, \sum_j V_j, \sum_j d_{j,1}$ for test statistic

**function** BLOCK( $[\![\mathbf{d}_{i,1}]\!]$ , $[\![\mathbf{d}_{i,2}]\!]$ , $[\![\mathbf{n}_{i,1}]\!]$ , $[\![\mathbf{n}_{i,2}]\!]$ )
  $[\![ac]\!] \leftarrow [\![\mathbf{d}_{i,1}]\!] + [\![\mathbf{d}_{i,2}]\!]$
  $[\![bd]\!] \leftarrow [\![\mathbf{n}_{i,1}]\!] + [\![\mathbf{n}_{i,2}]\!]$
  $[\![frc]\!]^f \leftarrow [\![ac]\!] / [\![bd]\!]$
  $[\![e_i]\!]^f \leftarrow [\![frc]\!]^f \cdot [\![\mathbf{n}_{i,1}]\!]$
  $[\![vn]\!] \leftarrow [\![\mathbf{n}_{i,1}]\!] \cdot [\![\mathbf{n}_{i,2}]\!] \cdot [\![ac]\!] \cdot ([\![bd]\!] - [\![ac]\!])$
  $[\![vd]\!] \leftarrow [\![bd]\!] \cdot [\![bd]\!] \cdot ([\![bd]\!] - 1)$
  $[\![v_i]\!]^f \leftarrow [\![vn]\!] / [\![vd]\!]$
  **return** $([\![e_i]\!]^f, [\![v_i]\!]^f, [\![d_i]\!])$
**end function**

**Algorithm 2** Logrank final computation

**Require:** $[\![es]\!], [\![vs]\!], [\![ds]\!]$ : summed-up values required to compute $X$
**Ensure:** $[\![chi]\!]^f$ test statistic comparing two curves; supposedly $chi \sim \chi^2_1$

**function** FIN( $[\![es]\!]$ , $[\![vs]\!]$ , $[\![ds]\!]$ )
  $[\![ds]\!]^f \leftarrow [\![ds]\!] \ll \text{PRECISION}$
  $[\![dmi]\!]^f \leftarrow [\![ds]\!]^f - [\![vs]\!]^f$
  $[\![chi]\!]^f \leftarrow [\![dmi]\!]^f / [\![vs]\!]^f$
  $[\![chi]\!]^f \leftarrow [\![chi]\!]^f \cdot [\![dmi]\!]^f$
  **return** $[\![chi]\!]^f$
**end function**

# Appendix C. Cryptographic Assumptions:

## C.1. Strong Diffie-Hellman:

Let $\langle \mathsf{group} \rangle = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g, h, e)$ where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of a prime order $q$, $g$ generates $\mathbb{G}_1$, $h$ generates $\mathbb{G}_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a (non-degenerate) bilinear map with security parameter $\lambda$. The **Strong Diffie-Hellman**[41][22] assumption states that for every efficient adversary $\mathcal{A}$ and a degree $d \in \mathbb{N}$ the following holds:

$$\Pr\left[ C = g^{\frac{1}{\alpha+c}} \;\middle|\; \begin{array}{c} \alpha \leftarrow \mathbb{F}_q \\ \mathbf{\Sigma} \leftarrow (\{g^{\alpha^i}\}_{i=0}^{d}, h^{\alpha}) \\ c, g^c \leftarrow \mathcal{A}(\langle \mathsf{group} \rangle, \mathbf{\Sigma}) \end{array} \right] = \mathsf{negl}(\lambda)$$

We note that this assumption is a stronger assumption that the $q$ dlog assumption as it additionally allows the adversary to pick $c$. If c is prespecified SDH reduces to q-dlog.

## C.2. Algebraic group Model:

In order to achieve additional efficiency, marlin papers shows how to construct polynomial commitment schemes in the Algebraic Group Model(AGM) [30], which replaces specific knowledge assumptions (such as Power Knowledge of Exponent assumptions) with simpler assumptions like SDH. Let $\mathbb{G}$ be a cyclic group of prime order $p$. An algorithm $A_{alg}$ algebraic if whenever $A_{alg}$ outputs a group element $Z \in \mathbb{G}$, it also outputs a "representation" $\boldsymbol{z} = (z_1, \ldots, z_t) \in \mathbb{Z}_p^t$ such that $\mathbf{Z} = \Pi_i L_i^{z_i}$ where $\mathbf{L} = (L_1, \ldots, L_t)$ is the list of all group elements that were given to $A_{alg}$ during it's execution so far.

In AGM, we model adversaries as algebraic, which means that whenever an adversary $\mathcal{A}$ outputs a group element $G$, $\mathcal{A}$ must also output an "explanation" or "representation" of $G$ in terms of the group elements that it has seen beforehand.

## C.3. Extractable Commitments

An extractable commitment scheme consists of a pair of probabilistic polynomial time algorithms $\mathsf{Setup}, \mathsf{Comm}$. The setup algorithm $\mathsf{Setup}(1^\lambda, \mathsf{D}) \rightarrow \mathsf{crs}, [\mathsf{ck_i}]_{\mathsf{i}=1}^{\mathsf{D}}, \mathsf{rk}, \mathsf{td}$ generates committer keys $\mathsf{ck_i}$ and a receiver key $\mathsf{rk}$ for the scheme and some trapdoor $\mathsf{td}$ for a given security parameter $\lambda$ and a bound $\mathsf{D}$. The commitment algorithm $\mathsf{Comm}$ defines a function $\mathsf{Comm}(\mathsf{ck_i}, m; r)$ outputs a commitment $c$ to the message $m$ with randomness $r$ using committer key $\mathsf{ck_i}$. Additionally, the trapdoor commitment scheme must satisfy the following properties:

**Computational Binding:** For all ppt $\mathcal{A}$ $\Pr[(\mathsf{crs}, \mathbf{ck}, .) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{D}), (m, r, m', r, \mathsf{i}) \leftarrow \mathcal{A}(\mathsf{crs}, \mathbf{ck}), \mathsf{Comm}(\mathsf{ck_i}, m; r) = \mathsf{Comm}(\mathsf{ck_i}, m'; r')] \approx \mathsf{negl}(\lambda)$.

**Trapdoor property:** There exist ppts $T_1, T_2$ such that: $(\mathsf{crs}, \mathbf{ck}, \mathsf{rk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{D})$, $(c_1, \tau) \leftarrow T_1(crs, td)$, $r \leftarrow T_2(c_1, \tau, m)$, then $c_1$ is identically distributed to real commitments and $\mathsf{Comm}(\mathsf{ck_i}, \mathsf{m}; \mathsf{r}) = c_1$.

**Perfect hiding:** $(\mathsf{crs}, \mathbf{ck}, .) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{D})$, $(r, r') \xleftarrow{\$} \mathbb{F}$, $\forall (m, m') : \mathsf{Comm}(\mathsf{ck_i}, m; r)$ is identically distributed to $\mathsf{Comm}(\mathsf{ck_i}, m'; r')$.

**Extractability:** $\forall$ ppt $\mathcal{A}$, there exists ppt $\mathcal{E}^{\mathcal{A}}$ such that $\Pr[(\mathsf{crs}, \mathbf{ck}, .) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{D}), (c||m;r) \leftarrow (\mathcal{A}||\mathcal{E}^{\mathcal{A}})(\mathsf{crs}, \mathbf{ck}), c \in \mathsf{Range}(\mathsf{Comm}(\mathsf{ck_i}, .)) \wedge \mathsf{c} \neq \mathsf{Comm}(\mathbf{ck_i}, \mathsf{m}; \mathsf{r})] \approx \mathsf{negl}(\lambda)$. Range denotes the range of commitments.

# Appendix D. Cost analysis

Table 3 shows the theoretical comparison between our constructions(PEC.Ped and PEC.Succ) and the construction from Veeningen [17]. Our construction for PEC.Ped and PEC.Succ incurs a five round communication overhead compared to one round in Veeningen's construction. As shown in Table 3 the prover cost is proportional to (Variable base Multi-scalar exponentiation) $v - MSM(m)$ that can be calculated in $\frac{m}{\log m}$ by using Peppinger's algorithm [42].

# Appendix E. Polycommits using MPC

We observe that all polycommit operations are MPC friendly, that is one can create commitments, provide evaluation proofs for local secret shared polynomials and then later interpolate them. $\mathsf{Eval}_{\mathsf{MPC}}$ thus requires regular Lagrange interpolation while providing evaluation proofs, polynomial commitments requires interpolating in the exponent. For simplicity, we state the protocols for single commitment and single evaluation, although batching can be supported. Infact, our implementation makes use of such batching.

## E.1. $\mathsf{PC.Commit}_{\mathsf{MPC}}$: PolyCommits from shares of evaluations

**Input:** shares of evaluation $[\![x_{i_1}]\!], \ldots, [\![x_{i_k}]\!]$ of $\hat{x}$
**Output:** Hiding polynomial commitment $C_{\hat{x}} = g^{\hat{x}} h^{\hat{r}}$. **Procedure:**(For each server $P_i$)

1) $[\![\hat{x}]\!] = \mathbf{V}([\![x_{i_1}]\!], \ldots, [\![x_{i_k}]\!])$ where $\mathbf{V}$ is the Vandermonde matrix over the evaluation domain corresponding to $k$.
2) Similarly, Compute $[\![\hat{r}]\!] = \mathbf{V}([\![r_{i_1}]\!], \ldots, [\![r_{i_k}]\!])$ where $[\![r_{i_j}]\!]$ are pre-processed random shares.
3) Compute $C_{[\![\hat{x}]\!]} = \mathsf{PC.commit}(\mathsf{ck}, [\![\hat{x}]\!], [\![\hat{r}]\!])$ and send the commitment $C_{[\![\hat{x}]\!]}$ to all other parties.
4) After receiving all the shares of commitments $C_{[\![\hat{x}]\!]}$, we check easily the evaluate the commitment to candidate interpolation result polynomial $\Pi_{i=1}^{t+1}(C_{[\![\hat{x}]\!]})^{\ell_i(v)}$ at any point $v$ we desire. We can use that to check whether $2t+1$ shares agree on some polynomial in the exponent. After finding such polynomial, we evaluate it at 0 in the exponent to obtain the $C_{\hat{x}}$.

## E.2. $\mathsf{PC.Open}_{\mathsf{MPC}}$: MPC Polynomial evaluation proofs

The protocol for $\mathsf{PC.Open}_{\mathsf{MPC}}$ proceeds similar to protocol for creating polynomial commitments as in Appendix E.1 with two minor changes. Instead of PC.Commit in step 3, we use PC.Open and instead of sampling randomness in step 2, we use the same randomness as the ones used in PC.Commit.

# Appendix F. Constructions of Polynomial Commitments

The versions of PEC.Ped and PEC.Lipmaa are useful only when there is a single input per party(as with the auction application). Hence, while stating the below constructions, we state them with $K = D$.

## F.1. Construction using Pedersen Commitments

Our polynomial evaluation commitment scheme PEC.Ped over a cyclic group $\mathbb{G}$ is constructed as follows:

- $\mathsf{Setup}(1^\lambda, D) \rightarrow \mathsf{ck_e}, \mathsf{ck_p}, \mathsf{rk_p}, \mathsf{td}$: Sample random generators $g$ and $h = g^{\mathsf{td}}$ and return $\mathsf{ck_{eval}} = (g,h), \mathsf{ck_{poly}} = (g,h), \mathsf{rk_{poly}} = (g,h), \mathsf{td}$.
- $\mathsf{Commit_{eval}}(\mathsf{ck_e}, e, p; \omega) \rightarrow c_e$: Parse $\mathsf{ck_e} = (g,h)$. Then the commitment to a evaluation $e$ is $c_e = g^{e_i} h^{r_i}$ where $r_i$ is sampled randomly according to randomness $\omega$.
- $\mathsf{Interpolate}(\mathsf{ck_p}, \mathbf{c_e}) \rightarrow \mathbf{c}$: $\mathbf{c} = \mathbf{c_e}$. The interpolate operation is same as collection of commitments and thus is not a succinct commitment.
- $\mathsf{Open}(\mathsf{ck_p}, \phi, q; \boldsymbol{\omega}) \rightarrow v, \pi$: Obtain the interpolated randomness $r_i$ from $\boldsymbol{\omega} = [\omega_i]$ (must the be same as the one used for $\mathsf{Commit_{eval}}$ at evaluation point $p_i$) and must be interpolation of committed values $e_i$. Interpolate $\mathbf{r}$, polynomial with evaluations $r_i$ and return $(v, \pi) = (\mathbf{p}(q), \mathbf{r}(q))$.
- $\mathsf{Check}(\mathsf{rk_{poly}}, \mathbf{c}, q, v, \pi) \rightarrow \{0,1\}$: Parse $\mathsf{rk_{poly}} = (g,h)$ and check

$$\Pi_{i=1}^{n} (c_i)^{\ell_i(q)} \stackrel{?}{=} g^v h^\pi \tag{4}$$

where $\ell_i(X)$ denotes the Lagrange polynomial at evaluation point $i$ (or $\omega^i$ in case of FFT).

We defer the proof of *Pec.Ped* to the original paper from Pedersen[43] for secret sharing.

## F.2. Auditor based on Lipmaa commitments

Our key idea is to commit to an evaluation (share) of the polynomial by committing to a Lagrange polynomial multiplied by the share. We can later homomorphically combine the commitments to shares to obtain a commitment to the polynomial, which we can provide polynomial evaluation proofs.

We first list some preliminaries that we use in this scheme. Let $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ be a evaluation vector of length $n$ which we wish to commit. For simplicity, we assume $n$ is a power of two, and let $\omega$ be the n-th primitive root of unity in a field $\mathbb{F}_p$. Further, let $\ell_i(X)$ be $\Pi_{j \neq i} \frac{X - \omega^j}{\omega i - \omega^j}$ be $ith$ Lagarange polynomial that

TABLE 3. $N$ : NUMBER OF MPC PARTIES; $K$ : NUMBER OF CLIENTS; $X$ :TOTAL STATEMENT SIZE(ALL INPUTS + OUTPUT); $m$ : TOTAL NUMBER OF GATES, $n$ : NUMBER OF MULTIPLICATION GATES; COMM: COMMUNICATION COST, $\pi$ PROOF SIZE. V-MSM($m$) DENOTES VARIABLE-BASE MULTI-SCALAR MULTIPLICATIONS (MSM) EACH OF SIZE $m$.

| | | size/cost(bytes) | | Time Complexity | | |
|---|---|---|---|---|---|---|
| | | Comm | $\|\pi\|$ | Prover | Auditor | Setup |
| Vee'17 | $\mathbb{G}_1$ | $K$ open | $3K+8$ | $\mathcal{O}(n)$ | $6K+12$ pair. | circ |
| | $\mathbb{G}_2$ | - | - | - | - | |
| | $\mathbb{F}_q$ | - | - | $\mathcal{O}(m+n\log n)$ | - | |
| Ours (Pec.Ped) | $\mathbb{G}_1$ | $X+9$ open | 17 | 21 v-MSM($3m$) + 9 v-MSM($N$) | $X+2$ grp operations | univ |
| | $\mathbb{G}_2$ | - | - | - | 2 pairings | |
| | $\mathbb{F}_q$ | 9 open | 23 | $\mathcal{O}(N+m\log m)$ | $\mathcal{O}(\log m)$ | |
| Ours (Pec.Suc) | $\mathbb{G}_1$ | $K+9$ open | 17 | 21 v-MSM($3m$) + 9 v-MSM($N$) | $K+3$ pair | univ |
| | $\mathbb{G}_2$ | - | - | - | - | |
| | $\mathbb{F}_q$ | 9 open | 23 | $\mathcal{O}(N+m\log m)$ | $\mathcal{O}(\log m)$ | |

is unique and has degree $n-1$ such that $\ell_i(\omega^i)=1$ and for $\ell_i(\omega^j)=0$ for $j\neq i$.

Clearly, we can evaluate the interpolated polynomial by viewing the $a_i$ as evaluations of the polynomials. $L_{\mathbf{a}}(X)=\Sigma_{i=1}^{n}a_i\ell_i(X)$. Lipmaa's construction provided an extractable interpolating commitment scheme, which we extend to create an extractable polynomial commitment. We define our scheme based on Lip'16 [34]'s construction with two major adaptations. First, any combination of linearly independent polynomials can be chosen for creating the CRS for a vector commitment scheme. For interpolating efficiency, Lip'16 schemes use evaluations of $\ell_i(\beta)$ for a trapdoor secret $\beta$. We instead use the more $1,\beta,\beta^2$ and create a polynomial commitment using Kate style polynomial commitments. Since our ZK-snark for Marlin uses the same CRS, it allows us to use common polycommit proof batching techniques for efficiency. Second, the Lip'16 scheme is based on PKE assumptions and hence requires double the elements in CRS and double the commitment size. Although it is possible to adapt our scheme to use plain model under knowledge assumptions, similar to Marlin, we use AGM to obtain an efficient construction.

We define PEC.Lipmaa as follows:

- $\mathsf{Setup}(1^\lambda, D)$ : Sample $\mathbf{\Sigma}$ as follows

$$\Sigma := \begin{pmatrix} g & g^{\alpha} & g^{\alpha^2} & \cdots & g^{\alpha^D} \\ g^{\gamma} & g^{\alpha\gamma} & g^{\gamma\alpha^2} & \cdots & g^{\gamma\alpha^D} \end{pmatrix}. \quad (5)$$

$$\mathsf{ck_p} := (\Sigma, D), \mathsf{ck_e} := ([g^{\ell_i(\alpha)}]_{i=1}^{n}, \mathbf{\Sigma}), \quad \mathsf{rk_p} := (D, g^{\gamma}, h^{\alpha}), \mathsf{td} := (\alpha, \gamma) \quad (6)$$

- $\mathsf{Commit}_{\mathsf{eval}}(\mathsf{ck_e}, e, p_i; \omega) \rightarrow c_e$: With input $\mathsf{ck}_{\mathsf{e}_i} = g^{\ell_i(\alpha)}$, point $p_i$ at index $i$, compute the commitment to evaluation $e$ as:

$$c_e := ((g^{\ell_i(\alpha)})^e g^{\gamma\bar{\phi}(\alpha)}, K, s, \overline{\phi_s})$$

where $\bar{\phi}(X)$ is a polynomial sampled according to randomness $\omega$ such that $\mathsf{deg}(\ell_i(X)) = \mathsf{deg}(\bar{\phi}(X))$, and

$$K := (g^{\ell_i(\alpha)})^r g^{\gamma\overline{\phi_r}(\alpha)}; \quad s := r+\beta e; \quad \overline{\phi_s} = \overline{\phi_r} + \beta\bar{\phi}$$

for a random challenge $\beta$ (which can be sampled by Fiat Shamir in a non-interactive version).

- $\mathsf{Interpolate}(\mathsf{ck_p}, \mathbf{c_e}) \rightarrow c$: Parse $\mathbf{c_e} = [c_{e_i}, K, s, \overline{\phi_s}]_{i=1}^{n}$. Check $\forall i \quad (g^{\ell_i(\alpha)})^s g^{\gamma\overline{\phi_s}(\alpha)} \stackrel{?}{=} K(c_{e_i})^{\beta}$ where $\beta$ is a verifier chosen challenge(or Fiat Shamir in Non-interactive setting). If check fails, abort, otherwise return $c = \Pi_{i=1}^{n} c_{e_i}$ which is a polynomial commitment to the polynomial whose shares are committed by $\mathbf{c_e}$.
- $\mathsf{Open}(\mathsf{ck_p}, \phi, q; \omega) \rightarrow v, \pi$: Same as regular polycommit open operation as described in Section 2.4
- $\mathsf{Check}(\mathsf{rk_p}, c, q, v, \pi) \rightarrow \{0,1\}$: Same as regular polynomial commitment operation as described in Section 2.4.

***Theorem F.1.*** If $\langle group \rangle$ satisfies the SDH assumption(Appendix C.1), then the above construction for PEC.Lipmaa is a PEC scheme (definition 4)

We prove this theorem in Appendix H.1

## Appendix G. Prover algorithm in MPC

In this section, we describe how the Marlin prover algorithm is computed using MPC. We first describe a round the Marlin protocol without the MPC version, then we describe the corresponding round in it's MPC version. All the public values are written in *blue* color.

### G.1. Prover Initialize

1) MPC servers engage in protocol to create commitments to PEC.Poly to the outputs from the shares of the output. The servers post the output commitments to the bulletin board. The bulletin board now has commitments for all statement elements(input and output).
2) Each servers computes a share of solution vector $[\![z_i]\!] = ([\![x_i]\!], [\![w_i]\!])$ to the QAP derived from R1CS.
3) For achieving zero knowledge purposes of underlying Holographic proof, we additionally need to have a additional degree $\mathsf{b}=1$ into the $\hat{x}$ (x interpolated) polynomial. We do this by

adding an extra dummy output statement chosen by the server. Note that $\hat{x}(X) \in \mathbb{F}^{|I|+|O|+b}[X]$ where $|I|$, $|O|$ denote the input domain size and output domain size of the MPC. Similar to $\hat{x}(X)$, the provers also construct a $\hat{r}(X)$ polynomial(with a hiding bound) consisting of random masks given by the client.

### G.2. Marlin First Round

With the prover initialized, we now engage in Marlin protocol. Instead of the statement being the public value, in this case, our public input is a commitment to the statement elements.

In the Marlin protocol, the prover first engages in a rowcheck protocol to attest the relation $\hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) = h_0(X)v_H(X)$. The prover computes $z := (x, w)$, $z_A := Az$, $z_B := Bz$ and $z_C := Cz$. It computes a $\hat{w}(X) \in \mathbb{F}^{|w|+b}[X]$, $\hat{z}_A(X) \in \mathbb{F}^{|H|+b}[X]$, $\hat{z}_B(X) \in \mathbb{F}^{|H|+b}[X]$, $\hat{z}_C(X) \in \mathbb{F}^{|H|+b}[X]$.

The prover then computes $h_0(X)$ such that

$$\hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) = h_0(X)v_H(X) \tag{7}$$

Sample $s(X) \in \mathbb{F}^{2|H|+b-1}[X]$ and compute $\sigma_1 := \sum_{\kappa \in H} s(\kappa)$. This $s(X)$ would help us in achieving zero knowledge.

In an MPC setting however, all servers have access to shares of $[\![z]\!] = ([\![x]\!], [\![w]\!])$ which was previously computed in prover Initialize step. Therefore, we directly operate on shares locally to compute shares of polynomials. Each server computes $[\![z_A]\!] := A[\![z]\!]$, $[\![z_B]\!] := B[\![z]\!]$ and $[\![z_C]\!] := C[\![z]\!]$. It computes a $[\![\hat{w}(X)]\!] \in \mathbb{F}^{|w|+b}[X]$, $[\![\hat{z}_A(X)]\!] \in \mathbb{F}^{|H|+b}[X]$, $[\![\hat{z}_B(X)]\!] \in \mathbb{F}^{|H|+b}[X]$, $[\![\hat{z}_C(X)]\!] \in \mathbb{F}^{|H|+b}[X]$. Recall the b is an additional degree added for zero-knowledge.

The MPC servers then computes $[\![h_0(X)]\!]$ such that

$$[\![h_0(X)]\!] = \frac{[\![\hat{z}_A(X)]\!][\![\hat{z}_B(X)]\!] - [\![\hat{z}_C(X)]\!]}{v_H(X)} \tag{8}$$

Ideally, the multiplication for $[\![\hat{z}_A(X)]\!][\![\hat{z}_B(X)]\!]$ would require beaver multiplication. But we the same optimization from Marlin to force $\hat{z}_C(X)$ to be equal to $\hat{z}_A(X)\hat{z}_B(X)$. The way to think about this is that all MPC servers combined together act as a single prover with each server having individual share elements. So, all optimizations to Marlin are still applicable to at MPC servers as a whole, but not an individual share level. Similarly, we use another optimization in Marlin to sample $[\![s(X)]\!]$ such that $[\![\sigma_1]\!]$ is 0. Both of these optimizations combined allow us to skip the above row check and $\sigma_1$ round.

The servers use protocol from App E to create the commitments $C_{w(X)}, C_{z_A(X)}$, $C_{z_B(X)}, C_{s(X)}$ from local evaluations of the respective polynomial shares and publish it to the blockchain.

To Summarise, in the first round, the servers:

1) Create the polynomials $[\![\hat{z}_A(X)]\!], [\![\hat{z}_B(X)]\!]$, $[\![\hat{w}(X)]\!], [\![\hat{s}(X)]\!]$ using the methods described above.
2) Send the commitments $C_{w(X)}, C_{z_A(X)}$, $C_{z_B(X)}, C_{s(X)}$ to the bulletin board.
3) Verifier sends a challenge $\alpha$, $\eta_A, \eta_B, \eta_C \in \mathbb{F}$. In non-interactive proof, the prover computes himself using random oracle using a transcript that contains the commitments to the statement elements and the above four polynomial commitments.

### G.3. Marlin Second Round

Prover computes the polynomial

$$q_1(X) = s(X) + r(\alpha, X)\left(\sum_M \eta_M \hat{z}_M(X)\right) - \left(\sum_M \eta_M r_m(\alpha, X)\right)\hat{z}(X)) \tag{9}$$

Prover then divides $q_1(X)$ by $v_H(X)$ to get $h_1(X)$ and $g_1(X)$ such that

$$q_1(X) = h_1(X)v_H(X) + g_1(X)X \tag{10}$$

In the marlin variant, each MPC server computes the share of the polynomial $[\![q_1(X)]\!]$ as follows:

$$[\![q_1(X)]\!] = [\![s(X)]\!] + r(\alpha, X)\left(\sum_M \eta_M [\![\hat{z}_M(X)]\!]\right) - \left(\sum_M \eta_M r_M(\alpha, X)\right)[\![\hat{z}(X)]\!] \tag{11}$$

The quantities in blue represent the public polynomials which don't rely on any secret data. Note that $r(X, Y) = \frac{X^{|H|} - Y^{|H|}}{X - Y}$ is the derivative polynomial as defined earlier, $\alpha, \eta_A, \eta_B, \eta_C$ are challenges which are public and finally $r_M(X, Y) = \sum_{\kappa \in H} r(X, \kappa)\hat{M}(\kappa, Y)$ which is also a public polynomial since $r(X, Y)$, $\hat{M}(X, Y)$ are both public polynomials. Recall that $\hat{M}$ is a low degree extension of the R1CS matrix M where $M \in \{A, B, C\}$ and hence public.

Finally, each server then divides $[\![q_1(X)]\!]$ by $v_H(X)$ using the divmod algorithm in fft to get $[\![h_1(X)]\!]$ and $[\![g_1(X)]\!]$ such that:

$$[\![q_1(X)]\!] = [\![h_1(X)]\!]v_H(X) + [\![g_1(X)]\!]X \tag{12}$$

and $deg([\![g_1(X)]\!]X) < deg(v_H(X))$. Recall that $\sigma_1$ was chosen to be zero as an optimization in the previous round. Again, as before the MPC servers compute $C_{h_1(X)}, C_{g_1(X)}$ using protocol from App E from the local shares of $[\![h_1(X)]\!], [\![g_1(X)]\!]$

To summarize, the MPC servers in the second round.

1) Prover carries our Marlin locally to compute $h_1(X), g_1(X)$ using the challenges from previous round.
2) Use Appendix E commit operation to create $C_{h_1(X)}, C_{g_1(X)}$ from local polynomial shares.
3) Server challenge $\beta_1$ is sampled from $\mathbb{F} \backslash H$ based on the transcript of first transcript plus the commitments $C_{h_1(X)}, C_{g_1(X)}$.

### G.4. Marlin Third Round

Note that the third and fourth rounds do not use any secret shared input, and thus this protocol can be thoroughly carried out in the open. All the polynomials in this round $r(X,Y), M(\hat{X},Y)$ are all public, and hence there is no difference between the marlin protocol and the secret shared version. We state the third and fourth rounds from Marlin for completeness. Each MPC computes the polynomial

$$q_2(X) = r(\alpha, X)\left(\sum_M \eta_M \hat{M}(X, \beta_1)\right) \tag{13}$$

and the sum-check result

$$\sigma_2 = \sum_{\kappa \in H} r(\alpha, \kappa)\left(\sum_{M \in A,B,C} \eta_M \hat{M}(\kappa, \beta_1)\right) \tag{14}$$

Each server then divides $q_2(X)$ by $v_H(X)$ to get $h_2(X)$ and $g_2(X)$ such that

$$q_2(X) = h_2(X)v_H(X) + g_2(X)X + \sigma_2/|H| \tag{15}$$

and $deg(g_2(X)X) < deg(v_H(X))$. Such a division is similar to protocol for divmod except that it is carried out in the open instead of secret shared form. Servers can then use standard PC.commit() to create commitments $C_{h_2(X)}, C_{g_2(X)}$ which are extractable and hiding. MPC servers sample $\beta_2$ from $\mathbb{F}\backslash H$ using Fiat Shamir using transcript upto the current round.

### G.5. Marlin Fourth Round

Again, as with the previous round, all the operations in this round are public and carried out in the open. So, everything is the same as the Marlin fourth round. Sum-check for the term:

$$\sum_{M \in \{A,B,C\}} \eta_M \frac{v_H(\beta_2)v_H(\beta_1)\hat{val}_M(X)}{(\beta_2 - \hat{row}_M(X))(\beta_1 - \hat{col}_M(X))} \tag{16}$$

$$\sigma_3 = \sum_{\kappa \in K} \sum_{M \in \{A,B,C\}} \eta_M \frac{v_H(\beta_2)v_H(\beta_1)\hat{val}_M(\kappa)}{(\beta_2 - \hat{row}_M(\kappa))(\beta_1 - \hat{col}_M(\kappa))} \tag{17}$$

Compute $a(X)$ and $b(X)$ deterministically from indexed $\hat{row}(X)$, $\hat{col}(X)$, $\hat{val}(X)$. We ignore the exact details for now, but this is done publicly based on challenges and public indexer values.

Find $h_3(X)$ and $g_3(X)$ such that

$$h_3(X)v_K(X) = a(X) - b(X)(Xg_3(X) + \sigma_3/|K|) \tag{18}$$

. Finally, compute $C_{h_3(X)}, C_{g_3(X)}$ using PC.commit() since all polynomials are public. The prover samples sends a challenge $\beta_3$ sampled from $\mathbb{F}$ according to fiat sharmir.

### G.6. Prover Poly Evaluation proofs

After the four rounds, the prover needs to provide proofs of evaluation of polycommits. The prover posts a proof for all the polynomials $p_i(X)$ with commitments $C_{p_i(X)}$ at evaluation points $\beta_j$ using KZG.Open($\beta_j$, $C_{p_i(X)}$, $p_i(X)$). To create a proof for public polynomials $p(X)$, we would standard KZG.open $(\beta_j, C_{p_i(X)}, p_i(X))$. If we want to create an evaluation proof on a secret shared polynomial, we use a create witness protocol described in Appendix E.

In standard marlin protocol, the prover and the verifier both had access to the statement, but in our scenario, the auditor does not have that access. Instead, we additionally need to provide the value $\hat{x}(\beta_1)$ proof that the value was correct which is exactly done by PEC.Open.

## Appendix H. Security Proofs

In this section, we will show that completeness, extraction, and zero-knowledge properties of PEC scheme construction F.2 and our adaptive zk-snarks described in section 2.

### H.1. Proof of theorem F.1

**Perfect Completeness** By inspection

**Extractability:** We first provide a construction for extractor and then claim that if the adversary is able to pass the extractability game(see 4) with non-negligible probability, the our construction for extractor fails with only negligible probability. Recall that in AGM C.2, the adversary would output a representation of $\mathbf{c_e}$ in terms of crs elements from $\mathbf{\Sigma}$.

The idea is to extract thw polynomial from the Sigma proof with AGM. This proof is similar to Uber assumptions in AGM [44], but repeat it here with our notation. For each commitment$(c_{e_i}, K_i, s_i, \overline{\phi_s})$, $c_{e_i} = (g^{\phi_i \alpha})^{e_i} g^{\gamma \overline{\phi_i}(\alpha)}$ and $K = (g^{\phi_{r_i} \alpha}) g^{\gamma \overline{\phi_{r_i}}(\alpha)}$. Obtain $r_i$ from the adversary representation. Then compute $(\phi_s(X))/\ell_i(X)$, if the division fails, our extractor aborts. Finally, compute $e_i = \frac{s-r}{\beta}$. Next, the outputs $\phi(X)$ as the interpolated polynomial corresponding to evaluations $e_i$ at $p_i$. The extractor also outputs randomness $\omega$ as $\omega = \Sigma \overline{\phi_i(X)}$.

Next, we need show to that if the adversary wins the game then our extractor will only fail with negligible probability and that values output by the extractor are satisfy the evaluation binding property. The first reason the extractor might fail is if the division $\phi_i(X)/\ell_i(X)$ fails. Note that since the adversary succeeds with non-negligible probability, it must satisfy the relation interpolation was carried out correctly. Which implies that the $\forall i$ $(g^{\ell_i(\alpha)})^s g^{\gamma \overline{\phi_s}(\alpha)} = K * (c_{e_i})^{\beta_c}$ for a randomly sampled $\beta_c$. Next, consider the algebraic $s * \ell_i(X) = r_i \phi_r(X) + Y * \phi(X)$ which must hold true for all $Y$(with soundness error $1/|\mathbb{F}|$), which implies both $\phi_r(X)$ and $\phi(X)$ must be a multiple of $\ell_i(X)$. Which means that $\phi(X)$ interpolated by $e_i$ indeed corresponds to the interpolated polycommit $c$.

Now, this polycommit is exactly like any other polycommit generated by KGZ10, but instead of single party creating it, we had different parties create

a commitment. Note that Opening and The next part of the proof is to show that the extracted polynomial is indeed evaluation binding such that the evaluation $v$ at query point $q$ is the same as [22] and KGZ10 [29]. At a high level, consider an adversary outputs two proofs $\pi = (\bar{v}, \mathbf{w})$ and $\pi' = (\overline{v'}, \mathbf{w})$ for two different evaluations $v$ and $v'$ such that both proofs satisfy the verification equation. Output a pair

$$(-q, \frac{1}{v' - v + \gamma(\overline{v'} - \bar{v})}(\mathbf{w} - \mathbf{w}'))$$

where $q$ is the query point.

We can show the if $q \neq \alpha$(the trapdoor), the pair breaks the SDH assumption C.1. The probability that $q = \alpha$, is $\frac{1}{|F|}$ since the trapdoor is not known to the adversary. For detailed version of the proof, we defer the reader to Appendix B of Marlin [22].

**Zero Knowledge:** We need show that views of the adversary in Ideal world(when interacting with simulator having trapdoors) and real prover are identically distributed. Following is the construction for our simulator.

- S.Setup $\rightarrow$ samples $\mathsf{td} = \alpha, \gamma$ and computes $\Sigma$(same as powers of $\alpha$ in equation 1).
- S.Commit$_{\mathsf{eval}}$: Sample random evaluation points $e_{r_i}$ and create simulated commitments to it randomness $\omega_i$. Let $\phi_r, c$ denote the hiding polynomial corresponding to interpolated randomness and interpolated commitment $c$.
- S.Open: Using the trapdoor $\alpha$, provide an evaluation proof for the polynomial opening at $\phi(q)$ at $q$ with randomness $\boldsymbol{\omega}$ as follows: Compute $\bar{v} = \phi_r(q) - \phi(q)/\gamma$ and $\mathbf{w}$ as $(c/(g^{\phi(q)-\gamma\bar{v}}))^{1/(\alpha - z)}$. Return $\pi = (\mathbf{w}, \bar{v})$

The view of the adversary consists of $(\pi, \mathbf{C_e}, \mathsf{ck_e})$ and public values $q, \phi(q), \mathbf{p}$. Since, S.Setup uses PEC.Setup the output is distributed identically in both worlds. Similarly, the commitments are identically distributed in both worlds because of blinding polynomial. $c$ is the output of interpolate method which is a deterministic function and thus is identically distributed in both worlds. Finally, substituting the above expression of simulated $\pi$ in Check, it is easy that proofs are also indistinguishable in both worlds.

## H.2. Proof of theorem 4.1

**Perfect Completeness** By inspection

**Extractability:** We first provide a construction for extractor and then claim that if the adversary is able to pass the extractability game(see 4) with non-negligible probability, the our construction for extractor fails with only negligible probability. Recall that in AGM C.2, the adversary would output a representation of $\mathbf{c_e}$ in terms of crs elements from $\mathbf{\Sigma}$.

The extraction proceeds similar to the extraction proof for the *PEC.Lipmaa* H.1, so we only highlight the differences here. The first part of the proof is to extract the polynomials from the evaluations. Since, we have distributed different trapdoors to different parties, the validity pairing check in interpolate ensures that the parties only use those terms in CRS.

For each commitment $(c_{e_i}, c^k_{e_i}) = (g^{\phi_i \alpha})^{e_i} g^{\gamma_i \tau \overline{\phi_i}(\alpha)}$. Furthermore, we also have the validity condition from the interpolate algorithm $\forall i \ \ (e(g, c^k_e)), g = e(g^\gamma_i, c_e)$. Looking at the terms on the right hand side, we see that $c_e$ must only contain CRS terms with the corresponding powers of which have a $\gamma_i$ component. This is because the knowledge components only have terms corresponding to powers of degrees of $\alpha^i$. Since, the interpolate check passes, we can conclude that parties can only commit to the certain degrees of $\alpha_i$ assigned to them by $\mathsf{ck_i}$. Since, all parties have different degrees of $\alpha_i$ allotted by $\mathsf{ck_i}$, the multiplication of those terms produces a commitment to a polynomial of a larger degree D. The first $d$ terms come party with $g^{\alpha^0}$, $g^{\alpha^1}$ ... $g^{\alpha^d}$, the next $d$ terms from the second party and so on. Finally, a single combined commitment $c$ of a degree $\mathsf{D} = d * K$ is created by multiplying the commitments.

Now, this polycommit is exactly like any other polycommit generated by KGZ10, but instead of single party creating it, we had different parties create a commitment. Note that Opening and The next part of the proof is to show that the extracted polynomial is indeed evaluation binding such that the evaluation $v$ at query point $q$ is the same as [22] and KGZ10 [29]. At a high level, consider an adversary outputs two proofs $\pi = (\bar{v}, \mathbf{w})$ and $\pi' = (\overline{v'}, \mathbf{w})$ for two different evaluations $v$ and $v'$ such that both proofs satisfy the verification equation. Output a pair

$$(-q, \frac{1}{v' - v + \gamma(\overline{v'} - \bar{v})}(\mathbf{w} - \mathbf{w}'))$$

where $q$ is the query point.

We can show the if $q \neq \alpha$(the trapdoor), the pair breaks the SDH assumption C.1. The probability that $q = \alpha$, is $\frac{1}{|F|}$ since the trapdoor is not known to the adversary. For detailed version of the proof, we defer the reader to Appendix B of Marlin [22].

**Zero Knowledge:** We need show that views of the adversary in Ideal world(when interacting with simulator having trapdoors) and real prover are identically distributed. Following is the construction for our simulator.

- S.Setup $\rightarrow$ samples $\mathsf{td} = \alpha, \gamma$ and computes $\Sigma$(same as powers of $\alpha$ in equation 1).
- S.Commit$_{\mathsf{eval}}$: Sample random evaluation points $e_{r_i}$ and create simulated commitments to it randomness $\omega_i$. Let $\phi_r, c$ denote the hiding polynomial corresponding to interpolated randomness and interpolated commitment $c$.
- S.Open: Using the trapdoor $\alpha$, provide an evaluation proof for the polynomial opening at $\phi(q)$ at $q$ with randomness $\boldsymbol{\omega}$ as follows: Compute $\bar{v} = \phi_r(q) - \phi(q)/\gamma$ and $\mathbf{w}$ as $(c/(g^{\phi(q)-\gamma\bar{v}}))^{1/(\alpha - z)}$. Return $\pi = (\mathbf{w}, \bar{v})$

The view of the adversary consists of $(\pi, \mathbf{C_e}, \mathsf{ck_e})$ and public values $q, \phi(q), \mathbf{p}$. Since, S.Setup uses PEC.Setup the output is distributed identically in

both worlds. Similarly, the commitments are identically distributed in both worlds because of blinding polynomial. $c$ is the output of interpolate method which is a deterministic function and thus is identically distributed in both worlds. Finally, substituting the above expression of simulated $\pi$ in Check, it is easy that proofs are also indistinguishable in both worlds.

### H.3. Proof of theorem 5.1

**Perfect Completeness** By inspection

**Extractability:** The output proof from $\mathcal{A}_2$ would be of the form $(\mathsf{prf_{mar}}, \mathsf{v}, \mathsf{ß_{comm}}, \mathbf{C}_{\mathbf{x}}^{\mathbb{b}})$. We construct our extractor $\mathcal{E}$ as follows:

- Use the Commitment Extractor $\mathcal{E}_{comm}$ to extract inputs and randomness $\mathbb{x}, \mathbb{r}$ from augment statement commitments $\mathbf{C_x}, \mathbf{C}_{\mathbf{x}}^{\mathbb{b}}$.
- Use the marlin extractor $\mathcal{E}_{Mar}$ with above extracted statement $\mathbb{x}$ and $\mathsf{prf_{mar}}$ to get witness $\mathbb{w}$. The constructions of this extractor is described in Section 8.3 of Marlin [22] paper.
- Use the same $\mathbb{i}$ as the one output by the $\mathcal{A}_1$ and output $(\mathbb{x}, \mathbb{r}, \mathbb{w}, \mathbb{i})$

We need to show that, if $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ produces a verifying proof, the extractor fails only with negligible probability and the returned values are in $\mathcal{R}_{\mathsf{ck}_{eval}}$.

Suppose that $\mathcal{E}$ fails with non-negligible probability $\epsilon$, then either two extractors ($\mathcal{E}_{comm}$ or $\mathcal{E}_{Mar}$) fail or the output $(\mathbb{x}, \mathbb{r}, \mathbb{w}, \mathbb{i}) \notin \mathcal{R}_{\mathsf{ck}_{eval}}$

- If $\mathcal{E}_{Mar}$ fails, then we can construct an adversary that makes the extractor fail to either break 1) the soundness of underlying Algebraic Holographic Proof(AHP) or 2) succeed in the extractability game for the PC scheme. We defer the reader to Section 8.3 of Marlin paper [22] for details.
- If $\mathcal{E}_{comm}$ fails with non-negligible probability, then we can break the extraction game for the PEC scheme.

It remains to show that the values returned by the extractor are in with high probability in $\mathcal{R}_{\mathsf{ck}}$(see section 2.5)

By properties of the extractor $\mathcal{E}_{comm}$, we know that openings to the commitments $\mathbf{C_x}$ are $(\mathbb{x}, \mathbb{r})$ and now we need to show that $(\mathbb{x}, \mathbb{i}, \mathbb{w}) \in \mathcal{R}$. Let the interpolated polynomial for all $\mathbb{x}$ be $\tilde{x}(X)$.

Note that in regular Marlin execution, the value $\mathsf{v}$ is computed the verifier himself based on the statement $x$. We need show that with high probability that the $\mathsf{v}$ is exactly the same of $\hat{x}(\beta_1)$. Suppose that this were not the same, $\mathsf{v} \neq \hat{x}(\beta_1)$. i.e $\tilde{x}(\beta_1) \neq \hat{x}(\beta_1)$ which means that $\hat{x}(X) \neq \tilde{x}(X)$.

From the underlying AHP Marlin protocol, we know that

$$\hat{z}(X) := \hat{w}(X)v_H(X) + \hat{x}(X) \tag{19}$$

is true for all the values of $X$. Therefore, the equation can only hold true for $\tilde{x}$ with probability $|X|/|\mathbb{F}|$ for a randomly sampled challenge $\beta_1$ from the verifier. $|X|$ is the statement length and also the degree of the polynomial $\hat{x}$.

**Zero Knowledge:** We need to the simulator with access to the trapdoor can generate proofs that can indistinguishable from real proofs. In other words, our simulator $\mathsf{S}$ is given given commitments to the $\mathsf{C_x}$ and needs to construct proofs using the trapdoors from commitment scheme and trapdoor from marlin. Our simulator works as follows:

The figure 7 shows the construction for simulator satisfying the above definition for our construction in 2. Next, we provide a proof about the indistinguishably for the ideal and real world for the adversary w.r.t our simulator.

First, we note that in the underlying AHP for $x$ polynomial, we also have introduced a query bound $\mathbb{b}$ in order to ensure zero knowledge of the evaluations of $x$ upto $\mathbb{b}$ queries. Informally, since the prover sampled $\hat{x}$ such that $\mathsf{deg}(\hat{x}) = \mathsf{deg}(x) + \mathbb{b}$ queries less than the query bound $\mathbb{b}$ information theoretically does not reveal any information about $x$.

Similar to the construction in [45], we would construct a simulator $\mathsf{AHP}'$ for the underlying AHP with changes that the first message of the prover also includes an encoding of statement along with the encoding of witness and encoding of its linear combinations. These encodings are protected against up to $\mathbb{b}$ queries because the encodings have degree $\mathbb{b}$more than corresponding encodings. The rest of the simulator for the subsequent rounds proceeds similarly to what is described in Marlin [22]. At the high level, all the subsequent messages are hidden by adding the additional $\mathsf{s}$ polynomial and hence do not reveal any information.

From the marlin simulator with the trapdoors to $\mathsf{srs}$ which uses $\mathsf{AHP}'$ instead of AHP described in marlin, we know that $\mathsf{prf}'_{\mathsf{mar}}$ and $\mathsf{prf_{mar}}$ are indistinguishable. For the last proof element $\mathbf{C}'^{\mathbb{b}}_{\mathbf{x}}$ is indistinguishable from $\mathbf{C}^{\mathbb{b}}_{\mathbf{x}}$. Similarly, using the trapdoors $\mathsf{td_{comm}}$ for PEC, we can simulated proofs for $(\pi'_{comm}, \hat{x}(\beta_1))$ are indistinguishable from $(\mathsf{ß_{comm}}, \mathsf{v})$.

Although, we have argued about the individual distributions of $\mathsf{prf}'_{\mathsf{mar}}$ and $\mathsf{prf_{mar}}$ are same and distributions for $(\mathsf{ß_{comm}}, \mathsf{v},,$ and $(\pi'_{comm}, \hat{x}(\beta_1))$. Similarly, we also showed that $\mathbf{C}'^{\mathbb{b}}_{\mathbf{x}}$ is indistinguishable from $\mathbf{C}^{\mathbb{b}}_{\mathbf{x}}$. The hiding property of the $\mathsf{PEC}$ scheme ensures that the simulator by using the trapdoor $\mathsf{trap_{comm}}$ can perfectly simulate the evaluation and the commitments. Since both of these distributions are independent and can individually be simulated we argue that the joint distribution views of prover and simulator are identical.

### H.4. Auditable MPC ideal functionality

We provide an ideal functionality for auditable MPC in the universal composability (UC) framework [35] in Figure 8. Our modeling is guided by a few goals: first, the protocol should allow programs to be chosen adaptively, without having to conduct additional trusted setups. Second, the ideal functionality to be simple and self-contained. While we provide a proof that our protocol realizes this ideal functionality in Appendix I, here we focus mainly on

**Simulator for Adaptive ZK Construction in 2**

- Recall that our proof consists of four elements $(\mathsf{prf}_{\mathsf{mar}}, \mathsf{v}, ß_{\mathsf{comm}}, \mathbf{C}_{\mathbf{x}}^{\mathbb{b}})$. First, the simulator samples a random polynomial $\hat{x}$ of $\mathsf{deg}(|\mathsf{C}_{\mathsf{x}}|) + \mathsf{deg}(|C_x^{\mathbb{b}}|)$. Note that all the information simulator needs is the length of the augmented statement.
- Using the trapdoors of the PEC, open the commitments $\mathbf{C_x}$ to corresponding evaluations of $\hat{x'}$ over a pre-selected domain. Extend the polynomial by degree $\mathbb{b}$ to get a resultant polynomial $\hat{x}$. Create a commitment to the additional $\mathbb{b}$evaluations as $(|C_x'^{\mathbb{b}}|)$.
- Run the marlin simulator with $\hat{x}$ to obtain a fake proof $\mathsf{prf}'_{\mathsf{mar}}$ and obtain the evaluation $\hat{x}(\beta_1)$.
- Interpolate the commitments $\mathbf{C_x}$ using PEC.interpolate to obtain a polynomial commitment $c_p$ for the $\hat{x}$ and provide a evaluation proof $ß'_{\mathsf{comm}}$ for the correct value $\hat{x}(\beta_1)$.
- return the proof $(\mathsf{prf}'_{\mathsf{mar}}, \hat{x}(\beta_1)\ ß'_{\mathsf{comm}}, (|C_x'^{\mathbb{b}}|))$.

Figure 7. Simulator construction for adaptive zk-snark

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

(01) On init: inputs $\leftarrow [\,]$, new $\leftarrow [\,]$
(02) On input $x_i$ from data-client $C_i$:
(03) Append $x_i$ to new
(04) Send $C_i$ to $\mathcal{A}$
(05) On input circuit **F** from Auditor:
(06) If $f \leq t$:
(07) Append new to inputs and set new $\leftarrow [\,]$
(08) Compute $o \leftarrow \mathbf{F}(\mathsf{inputs})$
(09) Optionally: Send $(\mathbf{F}, o)$ to Auditor
(10) Send $(\mathbf{F}, o)$ to $\mathcal{A}$
(11) Else:
(12) Optionally:
(13) Append new to inputs
(14) Compute $o \leftarrow \mathbf{F}(\mathsf{inputs})$
(15) Optionally: Send $(\mathbf{F}, o)$ to Auditor
(16) Send $(\mathbf{F}, \mathsf{new})$ to $\mathcal{A}$ and set new $\leftarrow [\,]$
(17) Send **F** to $\mathcal{A}$

Figure 8. UC ideal functionality for Auditable MPC

explaining what security properties the ideal functionality expresses. First, notice that while we allow any clients to submit secret inputs, a separate "Auditor" party is responsible for choosing the arbitrary application circuits and is the only party that receives the output. This is just for simplicity, but in a real system we envision using some public process (like a smart contract) to choose it. All our protocol requires is that the MPC servers and auditors agree on which circuit was chosen. Also in our protocol the audit routine is public coin, so anyone could could re-run the auditing subroutine for themselves.

Integrity of outputs is required in both the $f \leq t$ and $f > t$ settings since the only outputs (lines 8 and 14) are from applying the given circuit **F** to the provided inputs. Confidentiality of inputs is required in the $f \leq t$ setting, since the only information leaked to the adversary is the identity of data-clients (line 4) and the final output of the function (line 10). In an auction application, this corresponds to corrupt servers not learning bids placed by data-clients within the round. If $f > t$ then new inputs are leaked to the adversary in line 16. Our definition only considers public outputs for simplicity, though the construction can be extended to support private outputs to designated parties as well.

Our ideal functionality implies a subtle security guarantee: input independence. Notice that even in the $f > t$ case, the inputs of honest parties are leaked only *after* the output is computed. In other words, corrupted parties must commit to and have knowledge of the inputs they provide, even before they learn anything about what honest parties input. In the context of our auction application, each time the auction update function is computed (one round of the auction), all of the bids collected during this bound must be independent of each other. Put another way, seeing commitments to honest bids does not help corrupt parties create related bids in the same round, and this holds regardless of the number of corruptions. Note however that this no longer holds across rounds. In the $f > t$ case, corrupt servers in later rounds of MPC can know the inputs of data-clients from earlier rounds and choose their inputs based on those past inputs. Consider the same auction example as before, where bidding servers maintain a state of the top $k$ bids. In every MPC round, a new party submits new bids and servers update the state to reflect the current top $k$ bids. When all servers are corrupted, data-clients can collude the servers to learn the top bids and bid accordingly. The use of optionally in our functionality (lines 9,12,15) express that our functionality does not guarantee availability. The adversary may prevent output from being provided, but if output is delivered it is guaranteed to be correct.

## Appendix I.
## UC Proof Sketch

The universal composability framework [35] is based on the real/ideal-world paradigm. In this setting, the real world consists of interactions between the environment $\mathcal{Z}$, real-world adversary $\mathcal{A}$, and parties $P_1, \ldots, P_n$ running a protocol $\pi$. The ideal world consists of interactions between the environment $\mathcal{Z}$, the ideal-world adversary (or simulator) $\mathcal{S}$, "dummy" parties $\mathcal{D}_1, \ldots, \mathcal{D}_n$, and an ideal functionality $\mathcal{F}$.

Security in this framework is defined by having $\mathcal{Z}$ output a bit after interacting in the real or ideal worlds. If a simulator can be defined so that, for

every possible $\mathcal{Z}$, the distribution on this output bit in the real world is indistinguishable from that in the ideal world, it follows that the the real and ideal worlds are indistinguishable. This is usually stating as saying that a protocol $\pi$ UC-realizes a functionality $\mathcal{F}$.

More formally, let $\mathsf{EXEC}^{\mathcal{F},\mathcal{S},\mathcal{Z}}_{\mathsf{IDEAL}}(\mathsf{z},\tilde{\mathsf{r}})$ be the output of $\mathcal{Z}$ in the ideal world, after interacting with the simulator $\mathcal{S}$ and the dummy parties $\mathcal{D}_1,\ldots,\mathcal{D}_n$ (where dummy parties act as passthrough parties between $\mathcal{Z}$ and $\mathcal{F}$). z here is the input $\mathcal{Z}$ is initialized with and $\tilde{\mathsf{r}}$ is the set consisting of $\mathcal{Z}$, $\mathcal{S}$, and $\mathcal{F}$'s random tapes. We define $\mathsf{EXEC}^{\mathcal{F},\mathcal{S},\mathcal{Z}}_{\mathsf{IDEAL}}(\mathsf{z})$ to be the random variable after choosing $\tilde{\mathsf{r}}$ uniformly at random.

We define $\mathsf{EXEC}^{\pi,\mathcal{A},\mathcal{Z}}_{\mathsf{REAL}}(\tilde{\mathsf{r}})$ in a similar manner in the real world, except now considering the execution consisting of the real-world adversary $\mathcal{A}$ and parties $P_1,\ldots,P_n$ running a protocol $\pi$.

A protocol $\pi$ is said to UC-realize a functionality $\mathcal{F}$ if, for all adversaries $\mathcal{A}$, there exists a simulator $\mathcal{S}$, such that for all environments $\mathcal{Z}$,

$$\{\mathsf{EXEC}^{\pi,\mathcal{A},\mathcal{Z}}_{\mathsf{REAL}}(\tilde{\mathsf{r}})\}_{\mathsf{z}\in\{0,1\}^*} \stackrel{c}{\equiv} \{\mathsf{EXEC}^{\mathcal{F},\mathcal{S},\mathcal{Z}}_{\mathsf{IDEAL}}(\mathsf{z},\tilde{\mathsf{r}})\}_{\mathsf{z}\in\{0,1\}^*}$$

One thing to note is that the every real-world adversary $\mathcal{A}$ can be split into two parts: (1) a logical adversary, which performs the actual computations and so on, and (2) a dummy adversary, which simply receives messages computed by the logical component and routes them to where it is instructed to send messages. As the UC security definition quantifies over all environments, Canetti [35] recognized that it is often simpler to work with the dummy adversary and proved the equivalence with the above definition. Accordingly, we will work with the following, simpler definition:

A protocol $\pi$ is said to UC-realize a functionality $\mathcal{F}$ if there exists a simulator $\mathcal{S}$, such that for all environments $\mathcal{Z}$ and the dummy adversary $\mathcal{D}$,

$$\{\mathsf{EXEC}^{\pi,\mathcal{D},\mathcal{Z}}_{\mathsf{REAL}}(\tilde{\mathsf{r}})\}_{\mathsf{z}\in\{0,1\}^*} \stackrel{c}{\equiv} \{\mathsf{EXEC}^{\mathcal{F},\mathcal{S},\mathcal{Z}}_{\mathsf{IDEAL}}(\mathsf{z},\tilde{\mathsf{r}})\}_{\mathsf{z}\in\{0,1\}^*}$$

We now note that our protocol is based in the algebraic group model. To our knowledge, the interaction between the universal composability (UC) framework and the algebraic group model has not previously been explored. We leave in-depth exploration of this relationship to future work. In this work, however, we recognize that we only consider algebraic adversaries and note that this corresponds to the logical component of real-world adversaries. As the UC-security definition in terms of the dummy adversary depends on "moving" the logical component of the adversary to the environment, we note that we therefore only quantify over "algebraic environments." That is, we only consider environments that provide a representation for group elements in their messages to the dummy adversary or simulator.

We now show that the protocol $\Pi_{\mathsf{AuditableMPC}}$ (Figure 12) UC-realizes the functionality $\mathcal{F}_{\mathsf{AuditableMPC}}$ (Figure 8). We show the indistinguishability between the real and ideal worlds by performing an exhaustive case analysis on the messages sent by $\mathcal{Z}$ in both worlds, and considering the resulting transcript. We consider the $f \leq t$ and $f > t$ cases separately.

**Case 1: f $\leq$ t:**

We argue that the simulator defined in Figure 14 causes indistinguishability between the real and ideal worlds when $f \leq t$. We show that the internal simulation always tracks what occurs in the real world and additionally, the messages that $\mathcal{Z}$ receives in both worlds are indistinguishable.

First, consider the messages that $\mathcal{Z}$ can send to honest parties:

- $\mathcal{Z}$ provides an input $x_i$ to an honest data-client $C_i$:
  **Real World:** By lines 2-6 of Figure 12, $C_i$ creates a commitment to $x_i$ and sends this to $\mathcal{F}_{\mathsf{BulletinBoard}}$.
  **Ideal World:** The dummy $C_i$ in the ideal world forwards $x_i$ to $\mathcal{F}_{\mathsf{AuditableMPC}}$. By line 4 of Figure 8, $\mathcal{F}_{\mathsf{AuditableMPC}}$ sends the message $C_i$ to $\mathcal{S}$. By lines 6-8 of Figure 14, $\mathcal{S}$ simulates an internal input of 0 by $C_i$ and all state transitions.
  In both cases, $\mathcal{Z}$ is activated with no incoming messages and $\mathcal{Z}$ will not be able to distinguish on its activation. Additionally, the internal simulation will be at the same point of the execution as the real world.

Next, we consider the messages $\mathcal{Z}$ sends to the auditor:

- $\mathcal{Z}$ provides a circuit **F** to the auditor:
  **Real World:** One of two things can happen here. Either (1) the auditor stores **F** or (2) the auditor forwards **F** to $\mathcal{F}_{\mathsf{ReactiveMPC}}$.
  In the case of (1), nothing happens in the real world.
  In the case of (2), by line 16 of Figure 12, $\mathcal{F}_{\mathsf{ReactiveMPC}}$ schedules an optional codeblock to send **F** to each $C_i$.
  **Ideal World:** By line 10 of Figure 8, $\mathcal{F}_{\mathsf{AuditableMPC}}$ forwards $(\mathbf{F}, o)$ to $\mathcal{S}$. By lines 20-21 of Figure 14, $\mathcal{S}$ emulates the same actions as in the real world internally. Since the simulated execution is at the same point as the real world execution, this means that the internal emulation does (1) and (2) exactly as above.
  In both worlds, $\mathcal{Z}$ is activated with no incoming message. Additionally, the internal simulation of the real world matches the actual real world after this step.

  In order to explain lines 10-19 of Figure 14, we note that, in line 34 of Figure 12, $\Pi_{\mathsf{Auditable}}$ runs a prover algorithm for an auditable zk-SNARK. As proven in Theorem 5.1, this prover algorithm is for an auditable zk-SNARK as defined in Section 5.1. As it satisfies this definition, there is a simulator $\mathcal{S}_{\mathsf{zk-SNARK}}$ for the zero-knowledge property. The adaptive zk-SNARK zero-knowledge definition makes $\mathcal{S}_{\mathsf{zk-SNARK}}$ sufficient for $\mathcal{S}$ to run internally: the adversary in the definition chooses strictly

more parameters than $\mathcal{Z}$ does in our setting. Additionally, as the definition quantifies over all adversaries, it quantifies over adversaries that run the prover algorithm themselves, for any set of parameters, a polynomial number of times, which is exactly what $\mathcal{Z}$ can do. This means that a simulated transcript and proof generated by $\mathcal{S}_{\mathsf{zk-SNARK}}$ will be indistinguishable from an actual transcript and proof, from the perspective of $\mathcal{Z}$.

On input **F**, using lines 10-19 of Figure 14, $\mathcal{S}$ generates a simulated copy of the Marlin transcript and proof using $\mathcal{S}_{\mathsf{zk-SNARK}}$. $\mathcal{S}$ also generates random honest shares of each poly-commit in the transcript and random honest shares of the proof, consistent with the views of corrupt parties. These shares are used in lines 40-43 of Figure 14, when they are sent to $\mathcal{Z}$. This is okay for the following reason: $\mathcal{Z}$ only knows enough information to reconstruct up to $f$ shares of corrupt parties - however, as $f \leq t$, $\mathcal{Z}$ does not learn enough shares to constrain honest shares of polycommits. We note that $\mathcal{Z}$ (unlike $\mathcal{S}$) knows inputs of honest parties in the protocol - however, as $\mathcal{Z}$ doesn't know the trapdoor or randomness used for polycommits, it does not learn enough information to verify the honest shares of polycommits created by $\mathcal{S}$. This means that the simulated honest shares will be indistinguishable from actual honest shares.

Finally, we analyze the messages that $\mathcal{Z}$ can send the adversary.

- $\mathcal{Z}$ instructs $\mathcal{D}$ to send a message to $\mathcal{F}_{\mathsf{RO}}$, $\mathcal{F}_{\mathsf{Setup}}$, or corrupt $P_i$:
  **Real World:** $\mathcal{D}$ forwards the message and forwards any incoming message to $\mathcal{Z}$.
  **Ideal World:** In lines 30-35 of Figure 14, $\mathcal{S}$ forwards the message to the internal emulation and forwards any generated message to $\mathcal{Z}$. $\mathcal{F}_{\mathsf{RO}}$ and $\mathcal{F}_{\mathsf{Setup}}$ will be computationally indistinguishable in both worlds, so any queries will have indistinguishable responses. Additionally, $\mathcal{Z}$ can also instruct a corrupt $P_i$ to send a message to $\mathcal{F}_{\mathsf{BulletinBoard}}$, but this will happen identically in both worlds. Thus, $\mathcal{Z}$ will not be able to distinguish based on these inputs.
- $\mathcal{Z}$ instructs $\mathcal{D}$ to send message to a corrupt data-client $C_i$:
  **Real World:** $\mathcal{D}$ forwards the message to $C_i$ and forwards any incoming message to $\mathcal{Z}$
  **Ideal World:** By lines 24-29 of Figure 14, if this is related to $C_i$ sending an input to $\mathcal{F}_{\mathsf{ReactiveMPC}}$, $\mathcal{S}$ instructs the dummy corrupt data-client $C_i$ to forward the same input to $\mathcal{F}_{\mathsf{AuditableMPC}}$. The reason for this is this is the mechanism by which corrupt inputs are created in the real world; they must be created in the ideal world as well. The thing to note is that $\mathcal{A}$ cannot modify an input after it sends $\phi_{x_i}$ to $\mathcal{F}_{\mathsf{ReactiveMPC}}$ - this means any computation will happen on this input in the real world, and the computation in the ideal world should also happen on this input. $\mathcal{S}$ then forwards the message to the internal simulation and forwards any generated message to $\mathcal{Z}$ - as the emulation and actual real worlds were indistinguishable before this point, the output message will be indistinguishable in both worlds.
- $\mathcal{Z}$ instructs $\mathcal{D}$ to send a message to $\mathcal{F}_{\mathsf{ReactiveMPC}}$: We consider two separate cases here: (1) message related to triggering the optionally on line 16 of $\mathcal{F}_{\mathsf{ReactiveMPC}}$ in Figure 13 and (2) all other messages.

  In the first case:
  **Real World:** $\mathcal{D}$ instructs $\mathcal{F}_{\mathsf{ReactiveMPC}}$ to deliver **F** to a data-client $C_i$.
  If $C_i$ is honest, by lines 7-12 of Figure 12, $C_i$ sends inputs (that previously committed to in line 5) to $\mathcal{F}_{\mathsf{AuditableMPC}}$. By line 6, if this results in all the inputs to **F** being sent to $\mathcal{F}_{\mathsf{ReactiveMPC}}$, the compute subroutine is run and, by line 29 of Figure 13, $(\mathbf{F}, o, \mathsf{inputs}', \mathsf{wires}', \mathsf{round})$ to $\mathcal{A}$ is sent to $\mathcal{D}$. Else, if it doesn't result in all inputs being sent, by line 7 of Figure 13, $\mathcal{F}_{\mathsf{ReactiveMPC}}$ sends $C_i$ to $\mathcal{D}$.
  Else, if $C_i$ is corrupt, $\mathcal{F}_{\mathsf{ReactiveMPC}}$ sends **F** to $C_i$, which is forwarded to $\mathcal{D}$. In each of these cases, the message to $\mathcal{D}$ is forwarded to $\mathcal{Z}$.
  **Ideal World:** By lines 33-35 of Figure 14, $\mathcal{S}$ executes its internal simulation of the real world and generates the same types of messages as above. Except for messages of type $(\mathbf{F}, o, \mathsf{inputs}', \mathsf{wires}', \mathsf{round})$, these will be the same in both worlds. In the case of $(\mathbf{F}, o, \mathsf{inputs}', \mathsf{wires}', \mathsf{round})$ being forwarded to $\mathcal{A}$, we note that only $f \leq t$ shares of each input are given to $\mathcal{Z}$. This means that, although honest inputs were simulated as inputs of 0, this is okay as $t+1$ shares are required to define a $t$-degree polynomial. From the perspective of $\mathcal{Z}$, shares in $\mathsf{inputs}'$ for actual inputs in the real world and simulated inputs from the emulated real world are therefore indistinguishable.

  In the second case:
  **Real World:** $\mathcal{D}$ forwards the message to $\mathcal{F}_{\mathsf{ReactiveMPC}}$. One of two things might happen: (1) a dummy corrupt party forwards a copy of the output and shares from line 26 of Figure 13 or (2) the auditor produces output. These would, in each case, be forwarded to $\mathcal{Z}$.
  **Ideal World:** By lines 33-35 of Figure 14, $\mathcal{S}$ forwards the message to the internal emulation. As a copy of the real world is run internally, one of two things can happen here: the simulated $\mathcal{D}$ forwards a copy of the output and shares from line 26 of Figure 13 to $\mathcal{S}$ or the auditor produces output (line 48-55 of Figure 12). We maintain indistinguishability in either situation:
  (1) In line 35 of Figure 14, $\mathcal{S}$ forwards copies of the simulated shares and output to $\mathcal{Z}$. As argued in the first case, these simulated shares are indistinguishable from actual honest shares.
  (2) By lines 45-47 of Figure 14, $\mathcal{S}$ instructs

$\mathcal{F}_{\text{AuditableMPC}}$ to deliver the output to the auditor, which then forwards it to $\mathcal{Z}$. As $\mathcal{F}_{\text{ReactiveMPC}}$ and $\mathcal{F}_{\text{AuditableMPC}}$ generate outputs by evaluating the same function, and the inputs in both worlds are the same, the output will be the same in both worlds. As these outputs by the auditors in both worlds will be the same, we have that $\mathcal{Z}$ cannot distinguish in this case. We note that the outputs in both worlds being identical demonstrates the correctness of this protocol.

- $\mathcal{Z}$ instructs $\mathcal{D}$ to send a message to $\mathcal{F}_{\text{BulletinBoard}}$:
  **Real World:** $\mathcal{D}$ forwards the message to $\mathcal{F}_{\text{BulletinBoard}}$. One of two things can happen: (1) a dummy corrupt party forwards a copy of the bulletin board to $\mathcal{D}$ (by line 7 of Figure 11) or (2) the auditor produces output (per lines 48-55 of Figure 12). These would, in each case, be forwarded to $\mathcal{Z}$.
  **Ideal World:** By lines 36-44 of Figure 14, $\mathcal{S}$ forwards the message to the internal emulation. As a copy of the real world is run internally, one of two things can happen here: the simulated $\mathcal{D}$ forwards a copy of the bulletin board to $\mathcal{S}$ (by line 7 of the simulated $\mathcal{F}_{\text{BulletinBoard}}$ in Figure 11) or the auditor produces output (lines 48-55 of the simulated $\Pi_{\text{AuditableMPC}}$ in Figure 8). We maintain indistinguishability in either case.
  (1) In lines 40-43 of Figure 14, shares of the Marlin transcript and proof in the bulletin board are replaced with the versions from lines 15-18 of $\mathcal{S}$ in Figure 14. As argued previously, these shares are indistinguishable in both worlds. We note that the indistinguishability here corresponds to the zero-knowledge property of this protocol.
  (2) By lines 45-47 of Figure 14, $\mathcal{S}$ instructs $\mathcal{F}_{\text{AuditableMPC}}$ to deliver the output to the auditor, which then forwards it to $\mathcal{Z}$. As $\mathcal{F}_{\text{ReactiveMPC}}$ and $\mathcal{F}_{\text{AuditableMPC}}$ generate outputs by evaluating the same function, and the inputs in both worlds are the same, the output will be the same in both worlds. Similar to before, this demonstrates the correctness of this protocol.

As indistinguishability is maintained despite any messages sent by $\mathcal{Z}$, $\mathcal{S}$ as defined in Figure 14 causes $\Pi_{\text{AuditableMPC}}$ to UC-realize $\mathcal{F}_{\text{AuditableMPC}}$ when $f \leq t$.

**Case 2: f > t:**

We argue that $\mathcal{S}$, as defined in Figure 15, creates indistinguishability between the real and ideal worlds. We show that the internal simulation always tracks what occurs in the real world and additionally, the messages that $\mathcal{Z}$ is activated with in both worlds are indistinguishable.

In this setting, zero-knowledge is no longer a concern, as MPC does not guarantee confidentiality when $f > t$. Instead, since when $f > t$, corrupt inputs in line 3 of $\mathcal{F}_{\text{ReactiveMPC}}$ (Figure 13) can be different from those sent to parties in line 34, $\mathcal{S}$ must extract corrupt inputs from corrupt commitments to send to $\mathcal{F}_{\text{AuditableMPC}}$. Additionally, per the ideal functionality, input independence still holds in this setting and our proof must demonstrate that.

Similar to the $f \leq t$ case, we analyze each possible input by $\mathcal{Z}$.

First, consider the messages that $\mathcal{Z}$ can send to honest parties:

- $\mathcal{Z}$ provides an input $x_i$ to an honest data-client $C_i$:
  **Real World:** By lines 2-6 of Figure-12, $C_i$ creates a commitment to $x_i$ and sends this to the bulletin board $\mathcal{F}_{\text{BulletinBoard}}$.
  **Ideal World:** The dummy $C_i$ in the ideal world forwards $x_i$ to $\mathcal{F}_{\text{AuditableMPC}}$. By line 4 of Figure 8, $\mathcal{F}_{\text{AuditableMPC}}$ sends $C_i$ to $\mathcal{S}$. By lines 6-8 of Figure 15, $\mathcal{S}$ simulates an internal input of 0 by $C_i$ and all state transitions.
  In both cases, $\mathcal{Z}$ is activated with no incoming messages and $\mathcal{Z}$ will not be able to distinguish on its activation. Additionally, the internal simulation will be at the same point of the execution as the real world.

Next, consider the messages that $\mathcal{Z}$ can send the auditor.

- $\mathcal{Z}$ provides a circuit **F** to the auditor:
  **Real World:** One of two things can happen here. Either (1) the auditor stores **F** or (2) the auditor forwards **F** to $\mathcal{F}_{\text{ReactiveMPC}}$.
  In the case of (1), nothing happens in the real world.
  In the case of (2), by line 16 of Figure 12, $\mathcal{F}_{\text{ReactiveMPC}}$ schedules an optional codeblock to send **F** to each $C_i$.
  **Ideal World:** The auditor sends **F** to $\mathcal{F}_{\text{AuditableMPC}}$. By line 17 of Figure 8, $\mathcal{F}_{\text{AuditableMPC}}$ sends **F** to $\mathcal{S}$. By lines 9-10 of Figure 15, $\mathcal{S}$ emulates the same actions as in the real world internally. Since the simulated execution is at the same point as the real world execution, this means that the internal emulation does (1) and (2) exactly as above. In the case of (2), the simulator additionally runs lines 27-41 of Figure 15. What this does is perform an extraction of corrupt inputs. Note that, as the auditor is unable to be corrupted, we have that the auditor only causes an output whenever the zk-SNARK verifies, per lines 52-54 of $\Pi_{\text{AuditableMPC}}$ (Figure 12). If we consider the extractability definition of zk-SNARKs in Section 5.1, we can see that $\mathcal{Z}$ satisfies the conditions for $\mathcal{A}$ in the definition. This means that, even though we consider the UC setting, we can apply Theorem 5.1 in this situation for the result that our zk-SNARK scheme is extractable. But as a result of this scheme being extractable, we know that the verification algorithm will pass only with negligible probability if input commitments are not to the inputs on which computation is performed. That means that, as $\mathcal{Z}$ is algebraic, we can use the representation provided by $\mathcal{Z}$ for corrupt commitments to extract inputs (by the binding property of commitments, this is the only representation that $\mathcal{Z}$, except with negligible probability).

This extraction is exactly what lines 28-39 of Figure 15 do; lines 40-41 then send these inputs to $\mathcal{F}_{\mathsf{AuditableMPC}}$, thereby providing corrupt inputs for this round of execution. $\mathcal{S}$ also executes the optionally statement on line 12 of Figure 8, which results in all honest inputs being leaked to $\mathcal{S}$ in line 16. Once honest inputs are leaked, $\mathcal{S}$ modifies its internal emulation in lines 11-18 of Figure 15 to make it consistent with these inputs - as lines 13 and 16 modify polynomials that have not been sent to $\mathcal{Z}$ and line 14-15 use knowledge of the trapdoor to open commitments to new values, $\mathcal{S}$ is able to ensure that its internal execution is the same as the actual real world execution.

At this point, the output to $\mathcal{Z}$ will be the same in both worlds, and the internal emulation will be at the same point as (or in case (2), identical to) the execution in the real world, maintaining indistinguishability.

Finally, we analyze the different inputs from $\mathcal{Z}$ to the adversary.

- $\mathcal{Z}$ instructs $\mathcal{D}$ to send message to $\mathcal{F}_{\mathsf{RO}}$, $\mathcal{F}_{\mathsf{Setup}}$, $\mathcal{F}_{\mathsf{ReactiveMPC}}$, $\mathcal{F}_{\mathsf{BulletinBoard}}$, or corrupt $P_i/C_i$:
  **Real World:** $\mathcal{D}$ forwards the message. There are 3 possibilities at this point: either (1) the auditor forwards **F** to $\mathcal{F}_{\mathsf{ReactiveMPC}}$, (2) the auditor generates an output, or (3) $\mathcal{D}$ receives some miscellaneous message.
  **Ideal World:** In lines 19-26 of Figure 15, $\mathcal{S}$ forwards the message to its internal emulation. We consider each possibility described above in turn:
  (1) If the auditor forwards **F** to $\mathcal{F}_{\mathsf{ReactiveMPC}}$ in the real world, this will have been because $\mathcal{Z}$ instructed $\mathcal{D}$ to deliver a copy of the bulletin board to the auditor. By lines 41-42 and 45-47 of Figure 12, the real world auditor will have received all commitments for inputs. This will also occur in the simulated real world and line 27 of Figure 15 will be triggered. Note that, in the ideal world, the auditor will already have sent $\mathcal{F}_{\mathsf{AuditableMPC}}$ the circuit **F**. Despite this, we note that at this point, the same situation as described in case (2) of $\mathcal{Z}$ providing the auditor the circuit will happen at this point. $\mathcal{S}$ extracts and provides inputs to $\mathcal{F}_{\mathsf{AuditableMPC}}$ via lines 27-41 of Figure 15 in the same manner, and security still holds for the same reasons.
  (2) If the auditor generates an output in the real world, this will be because the verify function on line 52 of Figure 12 passes. As argued previously, the extractability of the zk-SNARK scheme implies that this only passes if the inputs and output are actually as it should be. By lines 42-44 of Figure 15, $\mathcal{S}$ instructs $\mathcal{F}_{\mathsf{AuditableMPC}}$ to send this same output to the ideal world auditor, which will then forward it to $\mathcal{Z}$. As the outputs in the real and ideal worlds match, indistinguishability follows.
  (3) Finally, consider the case that $\mathcal{D}$ outputs any other message in the real world. There will be two types of messages: (a) those sent *before* the auditor sends **F** to $\mathcal{F}_{\mathsf{ReactiveMPC}}$ and (b) those sent after.

  In the case of (a), note that the only messages that $\mathcal{Z}$ can send $\mathcal{D}$ will be to either query $\mathcal{F}_{\mathsf{RO}}$ and $\mathcal{F}_{\mathsf{Setup}}$, instructing appending/delivery of the bulletin board in $\mathcal{F}_{\mathsf{BulletinBoard}}$, instructing corrupt parties to send a message to $\mathcal{F}_{\mathsf{BulletinBoard}}$, or instructing corrupt inputs to $\mathcal{F}_{\mathsf{ReactiveMPC}}$. None of these rely on any secret information, so $\mathcal{S}$ will simulate exactly what occurs in the real world.

  Now consider case (b). But by this point, $\mathcal{S}$ will have run lines 40-41 in Figure 15. This will have triggered line 16 in Figure 8 and by lines 11-18 of Figure 15, $\mathcal{S}$ will already have modified its internal emulation to account for actual honest inputs. This means that the emulation will be exactly like that of the real world, and so any outputs in the simulated real world will be exactly that which happens in the actual real world.

Thus, the simulator defined in Figure 14 for $f \leq t$ corruptions and defined in Figure 15 for $f > t$ corruptions causes the real and ideal worlds to be indistinguishable to $\mathcal{Z}$. So we have that $\Pi_{\mathsf{AuditableMPC}}$ UC-realizes $\mathcal{F}_{\mathsf{AuditableMPC}}$ in the ($\mathcal{F}_{\mathsf{Setup}}$, $\mathcal{F}_{\mathsf{RO}}$, $\mathcal{F}_{\mathsf{BulletinBoard}}$, $\mathcal{F}_{\mathsf{ReactiveMPC}}$) hybrid world.


Figure 9. Ideal functionality for trusted setup

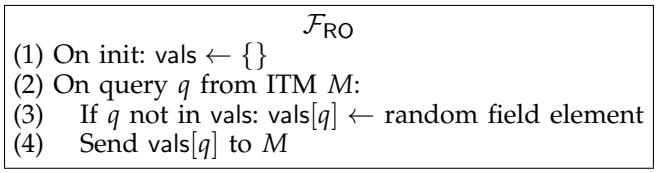


Figure 10. Ideal functionality for random oracle

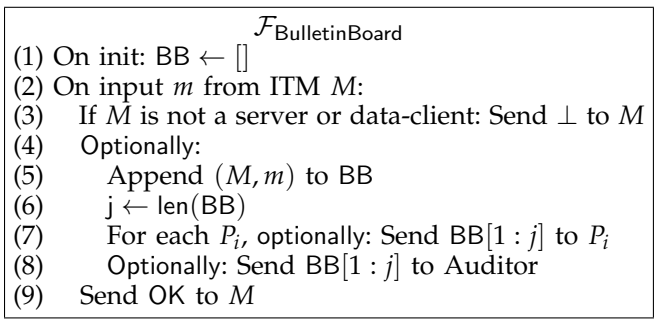


Figure 11. Ideal functionality for bulletin board

$\Pi_{\mathsf{AuditableMPC}}$

(1) **As data-client $C_i$:**
(2) On input $x_i$ from $\mathcal{Z}$:
(3) $r_i \xleftarrow{\$} \mathbb{Z}_q$
(4) $C_{x_i} \leftarrow \mathsf{Commit}_{\mathsf{eval}}(\mathsf{ck}_{\mathsf{e_i}}, x_i, r_i)$
(5) Send $C_{x_i}$ to $\mathcal{F}_{\mathsf{BulletinBoard}}$ and receive OK
(6) Store $(x_i, r_i)$
(7) On input $\mathbf{F}$ from $\mathcal{F}_{\mathsf{ReactiveMPC}}$:
(8) For each stored $(x_i, r_i)$:
(9) $\phi_{x_i}(\cdot) \leftarrow$ rand t-deg poly s.t. $\phi(0) = x_i$
(10) $\phi_{r_i}(\cdot) \leftarrow$ rand t-deg poly s.t. $\phi(0) = r_i$
(11) Unstore all $(x_i, r_i)$
(12) Send $\{(\phi_{x_i}, \phi_{r_i})\}_i$ to $\mathcal{F}_{\mathsf{ReactiveMPC}}$
(13) **As server $P_i$:**
(14) On init: BB $\leftarrow [\,]$
(15) On input BB′ from $\mathcal{F}_{\mathsf{BulletinBoard}}$:
(16) If len(BB)′ > len(BB):
(17) BB $\leftarrow$ BB′
(18) Run process_inputs
(19) On $(\mathbf{F}, o, \mathsf{inputs}, \mathsf{wires}, \mathsf{round})$ from $\mathcal{F}_{\mathsf{ReactiveMPC}}$:
(20) Send $(\mathsf{RANDS}, \mathsf{round})$ to $\mathcal{F}_{\mathsf{Reactive}}$ and get rands
(21) $[\![x_{i_1}]\!], \ldots, [\![x_{i_k}]\!] \leftarrow$ shares of new inputs
(22) $[\![r_{i_1}]\!], \ldots, [\![r_{i_k}]\!] \leftarrow$ shares in rands for new inputs
(23) $[\![\hat{x}]\!] \leftarrow$ interpolation of $[\![x_{i_1}]\!], \ldots, [\![x_{i_k}]\!]$
(24) $[\![\hat{r}]\!] \leftarrow$ interpolation of $[\![r_{i_1}]\!], \ldots, [\![r_{i_k}]\!]$
(25) $C_{[\![\hat{x}]\!]} \leftarrow \mathsf{PC.commit}(\mathsf{ck}, [\![\hat{\mathrm{x}}]\!], [\![\hat{\mathrm{r}}]\!])$
(26) Send $C_{[\![\hat{x}]\!]}$ to $\mathcal{F}_{\mathsf{BulletinBoard}}$
(27) Store $(\mathbf{F}, o, \mathsf{inputs}, \mathsf{wires}, \mathsf{round})$
(28) Run process_inputs
(29) Subroutine process_inputs:
(30) For each stored $(\mathbf{F}, o, \mathsf{inputs}, \mathsf{wires}, \mathsf{round})$:
(31) $\mathbf{C} \leftarrow$ shares of $C_{\hat{x}}$ for this round of inputs
(32) $b \leftarrow$ bit for if Step 4 of Appendix E.1 verifies
(33) If $b$:
(34) $[\![\pi]\!] \leftarrow$ prover protocol in Section 5.3
(35) Send $[\![\pi]\!]$ to $\mathcal{F}_{\mathsf{BulletinBoard}}$ and receive OK
(36) Unstore $(\mathbf{F}, o, \mathsf{inputs}, \mathsf{wires}, \mathsf{round})$
(37) If not $b$ and $\mathsf{len}(\mathbf{C}) == n$: abort
(38) **As Auditor:**
(39) On init: BB $\leftarrow [\,]$
(40) On input circuit $\mathbf{F}$ from $\mathcal{Z}$: Store $\mathbf{F}$ and run audit
(41) On input BB′ from $\mathcal{F}_{\mathsf{BulletinBoard}}$:
(42) If len(BB′) > len(BB): BB $\leftarrow$ BB′ and run audit
(43) On input $(\mathbf{F}, o, \mathsf{round})$ from $\mathcal{F}_{\mathsf{ReactiveMPC}}$:
(44) Store $(\mathbf{F}, o, \mathsf{round})$ and run audit
(45) Subroutine audit:
(46) For each stored $\mathbf{F}$:
(47) Send $\mathbf{F}$ to $\mathcal{F}_{\mathsf{ReactiveMPC}}$ if BB has $C_{x_i}$ for all $x_i$
(48) For each stored $(\mathbf{F}, o, \mathsf{round})$:
(49) If BB contains the entire Marlin proof:
(50) Run verify function described in Fig. 2 on the
(51) transcript in BB and $(\mathbf{F}, o, \mathsf{round})$
(52) If verify function passes:
(53) Unstore $(\mathbf{F}, o, \mathsf{round})$
(54) Send $(\mathbf{F}, o)$ to $\mathcal{Z}$
(55) Else: abort

Figure 12. UC protocol for auditable MPC

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

(1) On init: inputs $\leftarrow [\,]$, new_inputs $\leftarrow [\,]$, rands $\leftarrow \{\}$,
(2) round $\leftarrow 0$
(3) On input $(\phi_{x_i}, \phi_{r_i})$ from data-client $C_i$:
(4) Append $\phi_{x_i}$ to new_inputs
(5) Append $\phi_{r_i}$ to rands[round]
(6) If all inputs in $\mathbf{F}$ received: Execute compute
(7) Else: send $C_i$ to $\mathcal{A}$
(8) On input $(\mathsf{RANDS}, \mathsf{round}')$ from server $P_i$:
(9) If round′ > round: Send $[\,]$ to $P_i$
(10) If rand shares for proof for round′ not generated:
(11) $\phi_{r'_1}, \ldots, \phi_{r'_k} \leftarrow$ random t-deg polys for SNARK
(12) Append $\phi_{r'_1}, \ldots, \phi_{r'_k}$ to rands[round′]
(13) Send $[\phi(i)$ for each $\phi$ in rands[round′]] to $P_i$
(14) On input circuit $\mathbf{F}$ from Auditor:
(15) round $\leftarrow$ round $+ 1$
(16) Optionally: For each data-client $C_i$, send $\mathbf{F}$ to $C_i$
(17) Subroutine compute:
(18) If $f \leq t$:
(19) Add new_inputs to inputs and new_inputs $\leftarrow [\,]$
(20) $o \leftarrow \mathbf{F}(\mathsf{inputs})$
(21) wires $\leftarrow$ rand shares consistent with $\mathbf{F}$
(22) Optionally: Send $(\mathbf{F}, o, \mathsf{round})$ to Auditor
(23) For each $P_i$, optionally:
(24) inputs′ $\leftarrow P_i$'s shares of each input
(25) wires′ $\leftarrow P_i$'s shares of each wire
(26) Send $(\mathbf{F}, o, \mathsf{inputs}', \mathsf{wires}', \mathsf{round})$ to $P_i$
(27) inputs′ $\leftarrow$ corrupt shares of each input
(28) wires′ $\leftarrow$ corrupt shares of each wire
(29) Send $(\mathbf{F}, o, \mathsf{inputs}', \mathsf{wires}', \mathsf{round})$ to $\mathcal{A}$
(30) Else:
(31) Optionally ($\mathcal{A}$ providing o′):
(32) Send $(\mathbf{F}, o', \mathsf{round})$ to Auditor
(33) For each $P_i$:
(34) Optionally ($\mathcal{A}$ sending o′/new_inputs′/wires′):
(35) Append new_inputs′ to inputs[i]
(36) Send $(\mathbf{F}, o', \mathsf{inputs}[\mathrm{i}], \mathsf{wires}', \mathsf{round})$ to $P_i$
(37) Send $(\mathbf{F}, \mathsf{new\_inputs}, \mathsf{round})$ to $\mathcal{A}$

Figure 13. UC ideal functionality for reactive MPC

$\mathcal{S}_{\mathsf{AuditableMPC}}$

(1) **Case 1: f $\leq$ t:**
(2) On init: Initialize internal emulation of real world,
(3) including simulated parties and functionalities.
(4) Additionally, $\mathcal{F}_{\mathsf{Setup}}$ is initialized using $\mathcal{S}_{\mathsf{PEC}}.\mathsf{Setup}$
(5) and $\mathcal{S}_{\mathsf{zk-SNARK}}.\mathsf{Setup}$
(6) On input data-client $C_i$ from $\mathcal{F}_{\mathsf{AuditableMPC}}$:
(7) If $C_i$ is honest:
(8) Simulate $C_i$ being provided an input of 0
(9) On input $(\mathbf{F}, o)$ from $\mathcal{F}_{\mathsf{AuditableMPC}}$:
(10) $C_o, C_r \leftarrow$ polycommits to $o$ and extra degree of $\hat{x}$
(11) from $\Pi_{\mathsf{Marlin}}$, using internally simulated $r_{1,\ldots,k}(\cdot)$
(12) $C_{\hat{x}} \leftarrow$ [simulated/adv. commits to inputs, $C_o, C_r$]
(13) $(\mathsf{transcript}, \pi) \leftarrow \mathcal{S}_{\mathsf{SNARK}}.\mathsf{Prove}(\mathsf{td}_{\mathsf{srs}}, \mathsf{td}_{\mathsf{comm}}, \mathsf{C}_{\hat{x}}, \mathbb{I})$
(14) For each polycommit $C$ in transcript:
(15) Create share of $C$ for each honest $P_i$, consistent
(16) with the $f$ shares of corrupt parties and $C$
(1)
(17) Create share of $\pi$ for each honest $P_i$, consistent
(18) with the $f$ shares of corrupt parties and $\pi$
(2)
(19) Store $(\mathsf{transcript}, \mathsf{shares}, \pi, \mathbf{F}, o)$
(20) Pass $\mathbf{F}$ to the simulated Auditor, simulating
(21) $\mathcal{F}_{\mathsf{ReactiveMPC}}$ substituting $o$ as the output
(22) On input $(m, M')$ from $\mathcal{Z}$:
(23) If $M'$ is an honest $C_i$ or $P_i$: end activation
(24) If $M'$ is a corrupt data-client $C_i$:
(25) If $m$ is sending $(\phi_{x_i}, \phi_{r_i})$ to $\mathcal{F}_{\mathsf{ReactiveMPC}}$:
(26) Instruct $C_i$ to send $\phi_{x_i}(0)$ to $\mathcal{F}_{\mathsf{AuditableMPC}}$
(27) and wait for message $C_i$ from $\mathcal{F}_{\mathsf{AuditableMPC}}$
(28) Send $(m, M')$ to simulated $\mathcal{D}$.
(29) If $\mathcal{D}$ returns $m$, forward $m$ to $\mathcal{Z}$
(30) If $M'$ is a corrupt server $P_i$:
(31) Send $(m, M')$ to simulated $\mathcal{D}$.
(32) If $\mathcal{D}$ returns $m$, forward $m$ to $\mathcal{Z}$
(33) If $M'$ is $\mathcal{F}_{\mathsf{RO}}$, $\mathcal{F}_{\mathsf{Setup}}$, or $\mathcal{F}_{\mathsf{ReactiveMPC}}$:
(34) Send $(m, M')$ to simulated $\mathcal{D}$.
(35) If $\mathcal{D}$ returns $m$, forward $m$ to $\mathcal{Z}$
(36) If $M'$ is $\mathcal{F}_{\mathsf{BulletinBoard}}$:
(37) Send $m$ to simulated $\mathcal{F}_{\mathsf{BulletinBoard}}$
(38) If $\mathcal{D}$ returns $(P_j, \mathsf{BB})$ for corrupt $P_j$ and bulletin
(39) board BB:
(40) Replace honest shares of Marlin commits
(41) in BB with stored shares from (1)
(42) Replace honest shares of $\pi$ in BB with
(43) stored shares from (2)
(44) Send $(P_j, \mathsf{BB})$ to $\mathcal{Z}$
(45) If the simulated Auditor outputs $(\mathbf{F}, o)$:
(46) Instruct $\mathcal{F}_{\mathsf{AuditableMPC}}$ to execute the optionally
(47) statement to send $(\mathbf{F}, o)$ to the Auditor

Figure 14. Simulator for UC proof when $f \leq t$

$\mathcal{S}_{\mathsf{AuditableMPC}}$ **(continued)**

(1) **Case 2: f $>$ t:**
(2) On init: Initialize internal emulation of real world,
(3) including simulated parties and functionalities.
(4) Additionally, $\mathcal{F}_{\mathsf{Setup}}$ is initialized using $\mathcal{S}_{\mathsf{PEC}}.\mathsf{Setup}$
(5) and $\mathcal{S}_{\mathsf{zk-SNARK}}.\mathsf{Setup}$
(6) On input data-client $C_i$ from $\mathcal{F}_{\mathsf{AuditableMPC}}$:
(7) If $C_i$ is honest:
(8) Simulate $C_i$ being provided an input of 0
(9) On input $\mathbf{F}$ from $\mathcal{F}_{\mathsf{AuditableMPC}}$:
(10) Send $\mathbf{F}$ to simulated Auditor
(11) On input $(\mathbf{F}, \mathsf{new})$ from $\mathcal{F}_{\mathsf{AuditableMPC}}$:
(12) For each honest input $x_i$ in new:
(13) $\phi_{x_i} \leftarrow$ random $t$-degree consistent with $x_i$
(14) $C_{x_i} \leftarrow$ internally simulated commitment for $x_i$
(15) $r_i \leftarrow$ randomness to commit $x_i$ to $C_{x_i}$
(16) $\phi_{r_i} \leftarrow$ random $t$-degree consistent with $r_i$
(17) Modify internal simulation of $\mathcal{F}_{\mathsf{Reactive}}$ to use
(18) $\phi_{x_i}$ and $\phi_{r_i}$ for input $x_i$
(19) On input $(m, M')$ from $\mathcal{Z}$:
(20) If $M'$ is an honest $C_i$ or $P_i$: end activation
(21) If $M'$ is a corrupt $C_i$ or $P_i$:
(22) Send $(m, M')$ to simulated $\mathcal{D}$.
(23) If $M'$ is $\mathcal{F}_{\mathsf{RO}}$, $\mathcal{F}_{\mathsf{Setup}}$, $\mathcal{F}_{\mathsf{BulletinBoard}}$, or $\mathcal{F}_{\mathsf{Reactive}}$:
(24) Send $(m, M')$ to simulated $\mathcal{D}$.
(25) When $\mathcal{D}$ returns $m$, forward $m$ to $\mathcal{Z}$, unless the
(26) if statement below is triggered
(27) If the simulated Auditor provides $\mathbf{F}$ to $\mathcal{F}_{\mathsf{ReactiveMPC}}$:
(28) For each input commit $C_{x_i}$ on the bulletin board:
(29) If $x_i$ is not calculated for a corrupt party's $C_{x_i}$:
(30) $a_{1\ldots k} \leftarrow$ AGM repr. of $C_{x_i}$ provided by $Z$ in
(31) terms of SRS and group elems $\mathcal{S}$ sent $\mathcal{Z}$
(32) $a'_{1\ldots l} \leftarrow$ repr. of $C_{x_i}$ in terms of the SRS and
(33) elems created that $\mathcal{S}$ has no repr. for
(34) For each elem created with no representation:
(35) Create a representation w.r.t. the SRS using
(36) the trapdoor
(37) $[\overline{a_1}, \ldots, \overline{a_m}] \leftarrow$ repr. of $C_{x_i}$ in terms of the SRS
(38) $x_i, r_i \leftarrow [\overline{a_1}, \ldots, \overline{a_m}]$
(39) Store $x_i$ for input to $\mathcal{F}_{\mathsf{AuditableMPC}}$
(40) Send all $x_i$ to $\mathcal{F}_{\mathsf{AuditableMPC}}$ via dummy corrupt
(41) parties and instruct execution of inside optionally
(42) If the simulated Auditor outputs $(\mathbf{F}, o)$:
(43) Instruct $\mathcal{F}_{\mathsf{AuditableMPC}}$ to execute the optionally
(44) statement to send $(\mathbf{F}, o)$ to the Auditor

Figure 15. Simulator for UC proof when $f > t$