SUCCINCT PUBLICLY AUDITABLE MPC WITH UNIVERSAL SETUP

BY

SANKET MAKARAND KANJALKAR

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2020

Urbana, Illinois

Adviser:

Assistant Professor Andrew Miller

ABSTRACT

In recent years, Multiparty computation as a service (MPCSaaS) is gaining popularity as a promising approach for building privacy-preserving communication systems. Although there has been significant improvement in the efficiency and robustness of such protocols, in this thesis, we argue that these properties might not be enough for many MPC applications. Specifically, we need *unconditional public auditability*, which means that everyone can check that a given (secure) computation was performed correctly, even in the scenario where all the participants involved in the computation are corrupted. We also desire this auditability to be public, meaning any party that did not participate in the computation can audit it. We improve upon the previous state of the art adaptive Trinocchio by Veeningen [1] in three ways: 1) Our auditor has constant pairing cost 2) Our proof sizes are reduced by a factor of three, and 3) we do not rely on a circuit-specific setup.

In this work, we show the *first* construction of auditable robust MPC that supports fast verification, succinct proof size with one time universal, and updatable setup. Importantly, we provide auditability without significantly compromising the performance of the underlying Shamir secret shared MPC protocol, i.e, adding auditability only incurs a linear computation overhead and constant round communication overhead. We implement and evaluate our construction report various performance metrics.

To my parents, my brother, and my late aaji, ajoba.

ACKNOWLEDGMENTS

I thank my advisor Assistant Professor Andrew Miller for his guidance and advice, without whom all the three research papers that I am a part of would not have been possible. I would like to extend special thanks to Ye Zhang for his collaborations and the original idea for this project. I would also like to thank Pratyush Mishra for his help in understanding Marlin [2] paper. We also thank developers of zexe [3] and marlin [4] codebase which we build upon.

Lucky are the people who get to research full-time on things they like, and for that, I would like to extend a special thanks to Ruben, who got me started on my journey in bitcoin. I would like to thank Pieter, Andrew, Mario, and countless other unnamed people in bitcoin (Internet Relay Channels)IRC channels who helped me understand cryptocurrencies better.

Writing a thesis and doing academic research requires more than just technical advice. For that, I have countless people to thank for. I am especially grateful to my roommate Ankit, whose guidance, advice, and assistance have helped through everything at Univesity of Illinois at Urbana-Champaign(UIUC). I would like to specially acknowledge all my friend circles: Aurangabad Junta, Libidos Junta, Suwon Junta, Decentralized Systems Lab, Tamparature++, and the Core group for their support. The support from Anand, Ankit, Ashwin, Ayushi, Bai, Chirantan, Karan, Kiriti, Palash, Sahil, Samrat, Unnat was invaluable. Lastly, I would like to thank my parents, grandparents, and my brother, without whom this would not have been possible.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

In the past several years, there have been lots of innovations in increasing the performance of Multi Party Computation(MPC). Protocols like SPDZ [5], beaver's trick [6] have increased the performance by pushing out expensive computation to an offline phase to ensure a faster online phase. Furthermore, protocols like HoneybadgerMPC [7] and Blinder [8] provide robustness and tolerate malicious parties. MPC as service(MPCSaaS) is also gaining popularity because it allows clients to export the MPC computation to a set of cloud servers. MPCSaaS has been successfully deployed in practice for identifying the wage gap in gender differences [9] and cryptocurrency trusted parameter generation [10]. Applications like Callisto [11], MPC joins the dark side [12] and FuturesMex [13] also show promising applications for MPCSaaS. We note that all currently deployed MPC solutions offers security guarantees under the condition that a less certain threshold number of the servers are corrupted.

The need for auditability: Having *conditional* robust and efficient MPC protocol is not always enough: for many practical applications, we desire *unconditional public auditability*. For example, in a secure voting application, we would want to maintain the correctness of outcome even if all computing servers and participants are corrupted. Furthermore, we want any party, even those who did not even take part in the computation to verify the Integrity of computation.

We note that most MPC implementations only provide conditional security and if the threshold condition is not satisfied even the Integrity (or correctness) does not hold. Ideally, in the election application using secure MPC, we want to ensure Confidentiality: privacy of which voter voted for which candidate, Availability: the election should not be susceptible to DoS(Denial of service) attacks and Integrity: the candidate with the maximum votes should win the election. A loss of Confidentiality and Availability can lead to loss of privacy and denial of service attacks, but a compromise in Integrity can lead to the wrong election outcome, which might have far worse consequences. To summarize, we want to maintain Confidentiality and Availability when possible, but in the case that it is not possible, we would still like to preserve Integrity.

Table 1.1 shows the security guarantees provided by various MPC protocols. MPC toolkits like Viff [14], SPDZ [5], Mascot [15], Overdrive [16], EMP [17], SCALE-MAMBA [18], HYPERMPC [19] provide Confidentiality and Integrity if number of faults f is less than or equal to the threshold t. Toolkits like HoneyBadgerMPC [7] provide Confidentiality, Integrity and Availability if $f \leq t$. However, none of the above mentioned toolkits provide any security

guarantees if $f > t$. In this work, we show to how to add auditability to already existing protocols in order to provide Integrity if $f > t$.

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

Table 1.1: Security guarantees of MPC protocols: Conf, Int, Avail refer to Confidentiality, Integrity and availability. f, t represent the actual number of faults and the maximum faults tolerated by the MPC system.

Auditable MPC for generic protocols: Over the past two decades, lots of research by Schoenmakers et al. [20], Küsters et al. [21], Sako et al. [22] and Moran et al. [23] have considered a publicly auditable election protocols. Recently, Chen et al. [24] also propose a public auditable RSA modulus construction that can support thousands of parties and offers security against an arbitrary number of corrupted parties. Even among the few known useful applications of MPC already in deployment, such as RSA modulas construction , it's already important that they satisfy public auditability. When this has been provided, it's for a custom protocol. So far, none of the implementations of generic MPC frameworks have provided this feature.

Fast auditor with one-time universal setup: In MPCSaaS setting typically involves low powered clients sending outsourcing computation to the high-power servers. Thus, in order to audit the computation, the computation proof size must be small, and proof verification must be fast. Fast auditing is crucial in applications like secure election and cryptocurrency parameter generation, where the computation proofs are to be checked by a large number of low powered participants. Previous work by Baum et al. . [25] gave the first construction for auditable MPC; however, it involved the auditor reading the entire transcript, which would be inefficient for many practical applications.

Snarks have gained popularity as a for fast enable verifying NP statements. Over the years, numerous constructions [26] [2] [27] [28] [29] [30] of ZK-snarks have been proposed improving efficiency, proof size and verification times. In almost all of the constructions, proofs are verified with respect to plaintext inputs of the verifier; but for our reactive auditable(Refer section 2.2.2) MPC application, we want such proofs to able to be adaptive, meaning computations can be chosen after the input data has been committed.

Amongst the adaptive SNARKs, the hash-first approach suggested by Fiore et al. [31] for adaptive verifiable computations on outsourced data is not zero-knowledge. The construction by Lipmaa [32] for CaP SNARKs is zero knowledge but relies on a subset-sum

language instead of R1CS language. In more detail, even tough the Subset-Sum and R1CS languages are both NP-complete, most MPC computations are more naturally expressed as R1CS constraints whereas expressing the same in Subset Sum language would incur practical concrete overhead. Finally, another construction by Veeningen [1] shows how to adapt Trinocchio [33] for secure function evaluation. It, however, relies on a trusted third party to compute a circuit-specific setup for every new unknown computation. Generating a trusted setup server as a practical problem as to how to instantiate this party. The servers cannot be trusted to create this setup as our trust model involves all malicious servers. A one-setup can include the setups for some known circuits, but such a scenario one cannot change the computation after the setup. For deployment of auditable MPCSaaS, providers cannot know what computations the clients will be performing. Even if the client computations are known beforehand, clients can, at a later time, can request to update the computation because of implementation bugs or performance improvements. For a circuit-specific setup, it is not possible to update the computation without invoking a trusted third party.

Our construction uses Marlin [2] instead of Pinocchio [34] and thus does not require a trusted party for such a computation. The setup used marlin(and thus our construction) is also updatable [35] so that parameters for the setup can be updated and the soundness guarantees hold as long as there is at least one honest party in the update chain,

1.1 OUR CONTRIBUTIONS

- **First construction for fast auditable MPC with the one-time universal setup:** Our construction is the *first* to fast verification while relying on a one-time setup. We improve the previous state of the artwork by Veeningen [1] in three ways: 1) Our pairing costs in auditing is constant whereas the Veeningen's construction had to do pairings linear in the statement size and 2) Our proof sizes are 1/3 of that Veeningen's construction and 3) *importantly*, our construction uses a one-time universal setup instead of circuit-specific one.

- **Generation of Pre-processing arguments by MPC:** Marlin gave a new compiler design for constructing zk-SNARKs from Algebraic Holographic proofs and polynomial commitments. However, in Marlin [2], the prover knows the entire witness and thus easily construct such a marlin proof for R1CS(Rank-1 Constraint System). We show how to create Marlin proof for a relation where the witness and statements are secret shared across a set of parties. In order to facilitate the creation of Marlin proof via MPC, we formally suggest an MPC based polynomial commitment scheme where

MPC servers holding shares of polynomials can commit, evaluate, or create witness of the opened polynomial without opening the polynomial. This commitment scheme borrows core ideas of "interpolate in the exponent" from VSS literature, and we deem construction to be of independent interest.

- **Novel approach to adaptive pre-processing arguments:** In Marlin, the verifier has access to the statement and can verify the proof with respect to the statement. However, in an auditable MPC scenario, the auditor only has access to commitments of the statements and must make sure that proof is consistent with the commitment. Previous approaches to adaptive zk-SNARKs [1] divided the Pinocchio [34] proof into multiple blocks and embedded the statement in each block. Our new approach uses a commitment technique that allows building a polynomial commitment from Pedersen commitments to the evaluations that can be used for checking evaluations. To improve auditor efficiency, we also suggest a new commitment scheme based on Lipmaa's work [32] that can be of independent interest.

- We implement and evaluate our construction. In our experiments with 32 MPC servers, 10,000 constraints, and 8 statement size, our prover time is less than 3 seconds, auditing time less than 25ms, proof size is $\approx$ 1.7Kb, total MPC communication overhead is 700Kb with three additional rounds of communication.

CHAPTER 2: PRELIMINARIES

Secure multi-party computation (also known as MPC) helps parties to jointly compute a function over their inputs while keeping those inputs private. Informally, it is a function that has secret inputs and public output. The computation must preserve security properties, even is some(less than a threshold) of the parties collude and maliciously attack the protocol. Consider an auction example where parties want to compute the winning bid such that only the winning bid and nothing else is revealed.

We first informally define the security properties of interest for this application:

- Correctness/Integrity: The winning bid is correctly computed. A person with a lower bid cannot win
- Privacy: Participants only learn the winning bid and nothing else
- Independence of Inputs: Participants cannot bid 0.01$ more than the highest bid to win the auction
- Fairness: Participants cannot abort the auction if their bid is not the highest. If one party learns the outcome, all parties must learn the outcome.
- Guaranteed Output delivery: Parties cannot abort the protocol.

2.1 MULTI PARTY COMPUTATION

There are two ways of doing Multi-party computation: 1) Shamir secret sharing(SSS) [36] based and 2) Garbled circuits protocols [37] [38]. For this thesis, we focus on SSS based Multi prty computation.

2.1.1 Shamir Secret Sharing

Secret sharing is a technique for distributing a secret amongst a group of participants, each of whom is allocated a share of the secret. The secret can be reconstructed only when a sufficient number of shares of different types are combined. Individual shares do not reveal any information about the secret on their own. In general, given any-share, at least $t + 1$ parties are required to reconstruct the corresponding secret. We first recall some techniques from [39] [5] [40] which are used in MPC for practise.

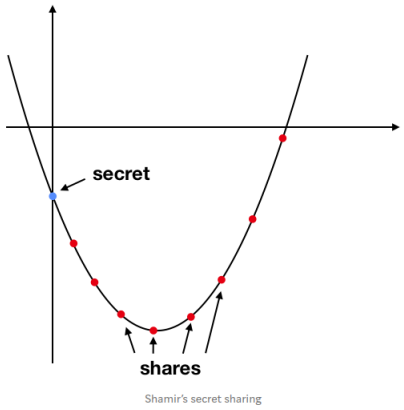

Figure 2.1: Shamir Secret Sharing illustration

For prime p and a secret $s \in \mathbb{F}_p$, $[\![s]\!]$ denotes Shamir secret sharing [36](SSS) in a (n, t) setting. In more detail, a degree-t polynomial $\phi : \mathbb{F}_p \to \mathbb{F}_p$ is sampled such that $\phi(0) = s$. The share $[\![s]\!]_t^i$ is same as evaluation ϕ at evaluation point $i \in \mathbb{F}_p$. We omit the superscript and/or subscript when it is clear from context. When constructing a secret we check whether $2t + 1$ correspond to the same polynomial and evaluate the secret at $\phi(0)$. (Refer Figure 2.1.1)

2.2 MPC MODES OF OPERATION

In our construction of auditable MPC, we make the distinction between data clients and input clients. Data clients only provide secret-shared input to MPC parties, while the input client chooses the computation f, which is to be performed upon the data. We say the protocol is adaptive if the function f can be chosen after the inputs are committed.

2.2.1 Single-shot MPC

In this style of MPC computation, the protocol executes in a single shot. Clients submit their inputs, servers carry out the respective computation, and send back the result to the clients. After the execution of the protocol, the auditor can check the validity of the results. As shown in [1], it is possible to guarantee input-independence in Single-shot MPC using aborts and non-malleable input commitments. Figure 2.2 shows the protocol in detail. Input independence is an important property in an auction application where the parties may not want their bids to input on other bids. For example, if a Pederson commitment scheme is committing inputs, client B can commit to value C_{x+1} using a commitment for

value C_x from client A. In the scenario where all servers are corrupted, we wish to compute the result, the servers would reveal x as an input of client A and use $x + 1$ as the input of client B making him win the auction with a small marginal price.

2.2.2 Reactive MPC

In a continuous style of computation, the MPC servers maintain a state of the application. The application interacts with 1) client inputs, 2) previous outputs of computation, and 3) public inputs to update its state. The application may also optionally reveal/open some part of the state to the result parties. This model allows us to capture the scenarios where an input to another completely different style of MPC might be an output of the previous computation. Let S_{old} and S_{new} denote the state of MPC computation after computing a circuit C .x_i denotes the client inputs, y_j denotes the output of the previous computations. Each server would have a share of state $[\![S]\!]$, the share of previous computations output y_j, a share of a secret client of input x_i, and public inputs x_{pub}. The MPC state update rule would then be shown by:

$$[\![S_{new}]\!] = \mathsf{MPC}(\mathsf{C}, [\![S_{old}]\!], [\![x_i]\!], [\![y_j]\!], x_{pub}) \tag{2.1}$$

The auditor can, at any time, check whether the outputs and the latest state of the MPC are consistent with the operations carried out MPC.

Note that input independence is not possible in reactive MPC across multiple rounds. In the scenario where all servers are corrupted, the servers will directly know the input and finish the computation for a particular round of MPC. The parties in the next round of MPC can know the inputs of servers from the first round and choose their inputs based on the inputs from the previous round. In the same auction example, where the bidding servers maintain a state of top k bids. In every MPC round, a new party submits the new bids; the server updates the state to reflect the correct top k bids. When all servers are corrupted, the next input client can collude the servers to know the current max bid and bid accordingly.

2.3 CRYPTOGRAPHIC COMMITMENTS

A non-interactive commitment scheme consists of a pair of probabilistic polynomial time algorithms $\mathsf{Setup}, \mathsf{Comm}$. The setup algorithm $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{pp}$ generates public parameters pp for the scheme, and for security parameter λ. The commitment algorithm Comp defines a function $\mathsf{Comm}_{\mathsf{pp}}(m, r)$ outputs a commitment to the message m with randomness r.

Client input processing for ensuring input independence

Data-clients $I_1, I_2, \ldots, I_m$ have input $\mathbf{x_i} \in \mathbb{F}$ respectively. Input Clients I_{inp} also knows the function f which they want to compute their data upon.

1. A trusted third party performs Keygen for a keyed trapdoor commitment family and distributes the keys ck_i to each data client.
2. Each data-client computes commitment to the inputs $(C_{ck_i}(x_i))$ and posts it on the bulletin board
3. The input-client then choose a function f to compute upon and submits it to the MPC servers for desired computation.
4. Each data-client provides the input to the MPC servers. The MPC servers obtain $[\![x_i]\!]$ and for each client check whether the inputs are consistent with the commitment $C_{ck_i}(x_i)$ posted on the bulletin board.
5. If the commitments do not match or a party fails to open the commitment, then abort. Otherwise proceed with MPC computation.

Figure 2.2: MPC input independent processing

Informally, Commitments satisfy hiding and binding properties.

1. Commitment does not leak any information about m
2. Commitment cannot be opened in two different ways.

2.3.1 Pedersen Commitments

Pedersen [41] showed how to information theoretically hide a secret popularly known as Pedersen commitments. The commitment is scheme is as follow: Let $\mathbb{G}$ be a group in which Discrete log is considered hard.

- $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{g}, \mathsf{h}$: Sample random generators g and h such that the discrete log between them in unknown.
- $\mathsf{Comm}_{\mathsf{pp}}(m, r) \rightarrow g^m h^r$: The commitment to a message m is simply a product of group elements g^m and h^r where r is sampled randomly.

Pedersen commitments also follow the linear homomorphirsm property:

$$\mathsf{Comm}(x_1; r_1) + \mathsf{Comm}(x_2; r_2) = \mathsf{Comm}(x_1 + x_2; r_1 + r_2) \tag{2.2}$$

2.3.2 Polynomial Commitments

Polynomial commitments [42] help a prover commit to a polynomial, so to prove that evaluations are correct without revealing the full polynomial. Polynomial commitments have been implicit in all cryptographic protocols.

A PolyCommit scheme consists of the following algorithms.

- $\mathsf{Setup}(1, t)$ generates system parameters SP to commit to a polynomial over of degree bound t. Setup is run by a trusted or distributed authority. SP can also be standardized for repeated use.
- $\mathsf{PolyCommit}(\mathsf{SP}, \phi(\cdot))$ outputs a commitment C to a polynomial $\phi(\cdot)$ for the system parameters SP, and some associated decommitment information aux.
- $\mathsf{CreateWitness}(\mathsf{SP}, \phi(\cdot), i, \mathsf{aux})$ outputs $\langle i, \phi(i), w_i \rangle$, where w_i is a witness for the decommitment information for the evaluation $\phi(i)$ of $\phi(\cdot)$ at the index i.
- $\mathsf{VerifyEval}(\mathsf{SP}, C, i, \phi(i), w_i)$ verifies that $\phi(i)$ is indeed the evaluation at the index i of the polynomial committed in C. If so, the algorithm outputs *accept*, otherwise it outputs *reject*.

A PolyCommit scheme must satisfy the following properties:

- **Correctness**: If $C, \mathsf{aux} \leftarrow \mathsf{Commit}(\mathsf{SP}, \phi(\cdot))$ and $w_i, \mathsf{aux}_i \leftarrow \mathsf{CreateWitness}(\mathsf{SP}, \phi(\cdot), i, \mathsf{aux})$, then the correct evaluation of $\phi(i)$ is successfully verified by $\mathsf{VerifyEval}(\mathsf{SP}, C, i, \phi(i), w_i, \mathsf{aux}_i)$.
- **Polynomial Binding**: If $C, \mathsf{aux} \leftarrow \mathsf{Commit}(\mathsf{SP}, \phi(\cdot))$, then except with negligible probability, an adversary can not create a polynomial $\phi'(\cdot)$ such that $\mathsf{VerifyPoly}(\mathsf{SP}, C, \phi(\cdot)', \mathsf{aux}) = 1$ if $\phi(\cdot) \neq \phi'(\cdot)$.
- **Evaluation Binding**: If $C, \mathsf{aux} \leftarrow \mathsf{Commit}(\mathsf{SP}, \phi(\cdot))$ and $w_i, \mathsf{aux}_i \leftarrow \mathsf{CreateWitness}(\mathsf{SP}, \phi(\cdot), i, \mathsf{aux})$ then except with negligible probability, an adversary can not create an evaluation $\phi(j)$, witness w_j, and decommitment information aux_j such that $\mathsf{VerifyEval}(\mathsf{SP}, C, i, \phi(j), w_j, \mathsf{aux}_j) = 1$ if $i \neq j$.
- **Hiding**: Given C and w_i for any i, an adversary either
  - Can only determine $\phi(\cdot)$ or $\phi(i)$ with negligible probability given bounded computation *(Computational Hiding)*
  - Can not determine any information about $\phi(\cdot)$ or $\phi(i)$, even given unbounded computation *(Unconditional Hiding)*

2.4 ZERO KNOWLEDGE PROOFS

Proofs of knowledge allow any prover to demonstrate knowledge of a satisfying witness to some NP statement. We are primarily interested in three properties of these proof systems, namely Correctness, Soundness, and Zero-Knowledge.

2.4.1 Bird's Eye View of Zero Knowledge Proofs

We informally state the three properties for brevity:

- Correctness/Completeness: If the statement is true, the honest verifier will be convinced of this fact by an honest prover after a successful protocol execution.
- Soundness: If the statement is false, no cheating prover can convince the honest verifier that it is true, except with some negligible probability.
- Zero-knowledge: If the statement is true, no verifier learns anything other than the fact that the statement is true.

2.4.2 Reference String Models

- The common reference string, also known as `crs` model, captures the setup in which all involved parties get access to the same string. This string is often called the `crs`.
- The structured reference string, also is known as `srs` model, captures the setup in which all involved parties get access to the same string, which has some structure and is generated by trapdoors. This string is often called the `srs`. No parties must have the knowledge of trapdoors used in constructing these structured reference strings.
- A universal structured reference string (`u-srs`) allows a single setup to support all circuits of some bounded size. Some `u-srs` constructions are updatable, meaning an open and dynamic set of participants can contribute secret randomness to it indefinitely.

Throughout this thesis, we refer to `u-srs` as `srs` as the universality is clear from the context.

Finally, the gold standard for setups is the "no trusted setup" model in which the `crs` is sampled randomly. Therefore, there is no possibility of backdoor and vastly increases the confidence in the system.

2.4.3 Metrics for Comparison of Proof Systems

We describe some metrics to compare different proof systems.

- **Trusted setup, and it is type:** There can be three broad categories for trusted setup. 1) No trusted setup, 2) circuit-specific trusted setup, and 3) Universal Trusted setup.
- **Prover time:** The amount of time required by the prover to generate a proof. The common proving time for circuits are $O(n), O(n \log n)$
- **Verification time:** The amount of time required by the verifier to validate a proof. The common proving time for circuits are $O(1), O(\log n), O(n)$. In standard literature, we say that verification is fast if the at-most $O(\log n)$
- **Proof size:** The size of non-interactive proof generated by the prover. The common proof sizes for circuits are $O(n), O(\sqrt{n}), O(1), O(\log n)$

2.5 SNARKS FOR R1CS

2.5.1 Indexed Relations

Marlin defines Snarks for 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 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.

Indexer $\mathbb{I}$ is a ppt machine that is responsible for creating polynomials from the circuit index $\mathbb{i}$. Given two interactive algorithms A and B, we denote by $\langle A(x), B(y)\rangle(z)$ the output of $B(y, z)$ when interacting with $A(x, z)$.

2.5.2 Marlin: Preprocessing zkSNARKs with Universal SRS

Marlin considers argument systems for indexed relations with the two following features:

- Security is proved under the SRS model.
- Anyone can publicly and deterministically preprocess a given circuit in an offline phase, in order to avoid recurring online costs for reusing the same circuit.

A preprocessing argument with universal SRS is a tuple of four algorithms $\mathsf{ARG} = (\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V})$. A ppt G samples an SRS srs that supports circuits up to a fixed number of constraints. The indexer I is a deterministic polynomial-time algorithm that uses srs and circuit index $\mathtt{i}$ satisfying the srs constraint bound, outputs an index proving and verification key $\mathsf{ipk}, \mathsf{ivk}$. Prover P uses ipk instead of $\mathtt{i}$ to provide a proof π for a indexed relation $\mathcal{R}$ which is then verified by the verifier V using ivk.

CHAPTER 3: RELATED WORK

Zk-SNARKs(Zero-knowledge Succinct Non-Interactive arguments of knowledge) have risen as a popular solution for fast verification of arithmetic relations. Over the past decade, several SNARKs like Pinocchio [34], Groth's constructions [43] [44] [45] and many others have made the pairing based Zk-SNARKs practical. Most of these pairing approaches required circuit-specific common reference string (srs), which requires a trusted third party to generate *toxic waste* (setup parameters which must be known be known to anybody). On the other hand, approaches like bulletproofs [27], STARKs [46] [47] [48] [49] [50] [51], hyrax [28] and ZKboo [52] [53] rely on uniform random string(urs) do not have such toxic waste but suffer from practical performance drawbacks in verification time. As a compromise between these approaches, a one-time universal setup [35] [26] [54] [55] [2] has emerged as a popular alternative to achieve the efficiency of pairing-based approaches with minimal only one-time toxic waste that is not circuit specific. We base our construction on a state of the art universal updatable Zk-SNARK Marlin [2] and show how to adapt the Zk-SNARK proof generation via MPC.

The idea of auditing a protocol has been around for a while. For example, auditable protocols have been studied extensively in electronic voting applications and secret sharing applications. The term public verification was introduced by Cohen and Fischer in 1985 [56]. In a popular work called Helios [57], the authors proposed the first web-based open audit voting system. A majority of research in auditing protocols has focused on auditing specific applications like elections [20] [21] [22] [23], auctions [58] [59] or secret shared dealing [20] [60] [61]. In a recent work Diogenes, by Chen et.al [24] propose a strong notion of security with identifiable-abort along with public-auditability for an RSA modulus

The notion of public auditable MPC for general arbitrary computations was first introduced by Baum et al. [25]. The authors base their work on celebrated SPDZ [62] [5] and suggest the first theoretical auditable SPDZ protocol. However, the protocol suggested by Baum et al. involves the auditor looking at the entire protocol transcript, and hence the audit phase has verification time and proof size linear in the size of the computation. In a client-server MPSaaS model, where lightweight clients want to audit the protocol execution, a linear audition is prohibitively expensive. Constructions from LegoSnark [63] show how to compose different "Commit and Prove"(CaP) SNARKs from different gadgets. Trinocchio [33] protocol offers a multi-party based construction for Pinocchio proof generation.

Veeningen showed [1] how to make the Trinocchio protocol adaptive(computations are chosen after the input data is committed), which closely resembles our work. The construc-

tion from Veeningen [1] relies on a circuit-specific setup and invoking a trusted third party for new circuit computation, whereas our construction has no such drawback. Secondly, Veeningen's construction only considers a single-shot execution of MPC because of which it faces no communication cost for MPC computation. In order to avoid the extra communication, they outsource the combination of proofs to the auditor, which makes the total proof size proportional to the number of servers. Our construction supports reactive MPC style computations (using outputs of previous rounds as inputs to new MPC rounds), and our proof size does not increase with the number of MPC servers but instead incurs three rounds of communication overhead. Third, in order to achieve input independence (parties cannot choose their input on inputs of other parties), Veeningen's construction introduced a keyed-commitment scheme even when all MPC servers are corrupt. We show that it is not possible to achieve input independence when all servers are corrupt in a reactive MPC and therefore base our construction on a simpler commitment scheme. For single-shot MPC, we also show how to extend our construction to achieve input independence for single-shot MPC using a new commitment scheme. Finally, because of statement embedding the proof checking, the auditor in Veeningen's construction has pairing cost linear in the number of commitments while we only incur a penalty in group operations(far cheaper than pairings) and keep constant pairing costs.

CHAPTER 4: AUDITABLE MPC OVERVIEW

We first describe the primary operations of our system at a high level. We have two main phases in our computation: The offline phase and the online phase. The offline phase carries out the expensive part of the computation. The online phase includes interaction between MPC servers 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 4 shows the high-level overview describing the components in our protocol.

4.1 OFFLINE PHASE

The offline phase of auditable MPC includes the following components.

- The one-time setup creation for universal Snarks: This setup should only be carried out once.
- Generating the MPC pre-processing elements for random shares and beaver triples: In order to facilitate fast online multiplication of MPC servers, it is necessary to consume offline generated beaver triple shares and random element shares. Thus, it is critical to ensure a steady supply rate of pre-processed elements.
- Marlin Indexed circuit generated indexer prover and verification keys: This phase should be run every time there is a request for a new computation indexing or an update to an existing computation.

The one-time setup is the source of toxic waste in our scheme and must be carried out by a trusted party or a collection of trusted parties [64]. Pre-processed elements are used in MPC to speed up online computation. It is possible to have an auditable MPC proof for robust

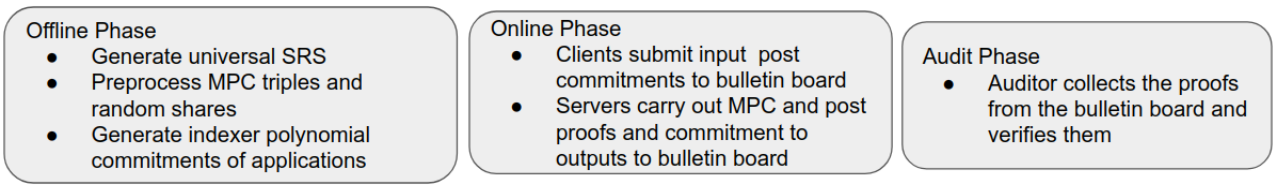

Figure 4.1: High Level overview of of auditable MPC protocol

offline phase as done in [25], but we majorly focus on online phase for this work. Finally, Marlin relies on pre-processing some circuit index in a deterministic fashion. Anyone can pre-process an index and check whether the corresponding proving and verification keys are correct. In practice, one can assume that proving and verification keys for standard circuits are readily available and would not require recomputing the keys again. In chapter 6, we will describe the details of the construction.

4.2 ONLINE PHASE

In the online phase, the clients provide the inputs and servers to engage in an MPC protocol to compute the desired function. We divide our Online Phase into further three parts:

- In the **input phase**, the input parties I_i provide commitments of their input on the bulletin board. These commitments are later used as part of the proof to verify the computation results. In a single-shot case, these commitments can also ensure that input value independence across parties. First, the input parties' secret shares their input values to the MPC servers. The servers then use robust interpolation in the exponent to check whether the shares are consistent with the ones provided by the input party.

- In the **Compute phase**, the MPC servers carry our the MPC protocol to obtain the secret shares of the output. After computing the output shares at each server, the servers engage in another MPC protocol to compute the Marlin proof for that given circuit. This computation involves servers combining the input and output(only in reactive MPC) share commitments to create a statement commitment. Servers interact amongst each other to produce polynomial evaluation proofs of secret shared polynomials required in Marlin protocol and post a single combined proof to the bulletin board while maintaining the soundness and zero-knowledge property of Marlin. The single proof can either be a simple Multi-signature or a threshold signature of 2t+1 parties.

- In the **audit phase**, the auditor checks that the marlin proof correctly verifies. Normally in zero-knowledge proofs, a component of verification equation based on the statement is computed locally by the verifier. Instead, in an auditable MPC scenario, where the auditor does not have access to the statement, the proof supplies the statement component of the verification equation(still not revealing anything about the

statement) to the auditor. The auditor checks 1) the claimed statement component is consistent with input and output commitments of computation, and 2) that the marlin proof verifies correctly.

CHAPTER 5: SECURITY DEFINITIONS AND IDEAL FUNCTIONALITIES

In this section, we will formally define our SNARK definitions along with security guarantees. We first define a new scheme for committing the evaluations of a polynomial and state the security proofs. We then define our construction for adaptive snark and finally give an informal outline for the ideal functionality of auditable MPC.

5.1 POLYNOMIAL EVALUATION COMMITMENT(PEC) SCHEME

We now define a new polynomial evaluation commitment scheme that is required for our application. The primary operations that we wish to support are the creation of a polynomial commitment from commitments to evaluations and checking the evaluation proofs for that polynomial commitment.

Our polynomial evaluation commitment scheme over a field $\mathbb{F}$ is defined by the following set of algorithms $\mathsf{PEC} = (\mathsf{Setup}, \mathsf{Commit}_{\mathsf{eval}}, \mathsf{Interpolate}, \mathsf{Open}, \mathsf{Check})$.

- $\mathsf{Setup}(1^\lambda, D) \rightarrow \mathsf{ck}_{\mathsf{eval}}, \mathsf{ck}_{\mathsf{poly}}, \mathsf{rk}_{\mathsf{poly}}, \mathsf{td}$: On input a security parameter λ (in unary), and a maximum degree bound $D \in \mathbb{N}$, Setup samples and outputs some public parameters $ck_{eval}, ck_{poly}, rk_{poly}$ and trapdoor td.

- $\mathsf{Commit}_{\mathsf{eval}}(\mathsf{ck}_{\mathsf{eval}}, \mathsf{evals}, \mathsf{points}, \omega) \rightarrow \mathsf{c}_{\mathsf{eval}}$ Given input evaluation committer key $\mathsf{ck}_{\mathsf{eval}}$, univariate polynomial evaluations $\mathsf{evals} = [eval_i]_{i=1}^n$ at evaluation points $\mathsf{points} = [point_i]$ over a field $\mathbb{F}$, $\mathsf{Commit}_{\mathsf{eval}}$ outputs commitments $\mathsf{c}_{\mathsf{eval}} = [c_{eval_i}]_{i=1}^n$ to the evaluations $\mathsf{evals} = [eval_i]_{i=1}^n$. The randomness $\omega = [\omega_i]_{i=0}^n$ is used if the commitments are required to be hiding.

- $\mathsf{interpolate}(\mathsf{ck}_{\mathsf{poly}}, \mathsf{c}_{\mathsf{eval}}) \rightarrow c$: On input $\mathsf{ck}_{\mathsf{eval}}$ vector, $\mathsf{c}_{\mathsf{eval}} = [c_{eval_i}]_{i=1}^n$, $\mathsf{interpolate}$ outputs a commitment to the interpolated polynomial corresponding the evaluations of the committed in $\mathsf{c}_{\mathsf{eval}}$.

- $\mathsf{Open}(\mathsf{ck}_{\mathsf{poly}}, p, q; \omega) \rightarrow v, \pi$ On inputs $\mathsf{ck}_{\mathsf{poly}}$, uni-variate polynomial p over a field $\mathbb{F}$, a query point q consisting of tuples $\in F$, Open outputs an evaluation v and evaluation proof π. If the commitment is hiding, the ω used must be consistent with the one used in $\mathsf{commit}_{\mathsf{eval}}$.

- $\mathsf{Check}(\mathsf{rk}_{\mathsf{poly}}, c, q, v, \pi) \in \{0, 1\}$: On input rk, commitments $c = [c_i]_{i=1}^n$, a query $q \in F$, claimed evaluations v for q and evaluation proof π, check outputs 1 if π attests that all the claimed evaluations corresponding the committed polynomial is correct.

Further, we want our definitions to satisfy the following properties: For simplicity, we only state our definitions for a single query though it can be extended to multiple queries too.

- Completeness: We say that PEC is complete if for all adversaries the following holds

$$\Pr\left[\begin{array}{c|c} & \mathsf{ck}_{\mathsf{eval}}, \mathsf{ck}_{\mathsf{poly}}, \mathsf{rk}_{\mathsf{poly}}, \mathsf{td} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{D}) \\ & (\mathsf{evals}, \mathsf{points}, q, \omega) \leftarrow \mathcal{A}(\mathsf{ck}_{\mathsf{eval}}, \mathsf{ck}_{\mathsf{poly}}, \mathsf{rk}_{\mathsf{poly}}; \mathsf{z}) \\ deg(p) \leq D & \mathsf{c}_{\mathsf{eval}} \leftarrow \mathsf{Commit}_{\mathsf{eval}}(\mathsf{ck}_{\mathsf{eval}}, \mathsf{evals}, \mathsf{points}; \omega) \\ \Downarrow & c \leftarrow \mathsf{interpolate}(\mathsf{ck}_{\mathsf{eval}}, \mathsf{c}_{\mathsf{eval}}) \\ \mathsf{Check}(\mathsf{rk}_{\mathsf{poly}}, \mathsf{c}, q, v, \pi) = 1 & p \leftarrow \mathsf{Lagrange\_interpolate}(\mathsf{evals}, \mathsf{points}) \\ & v \leftarrow p(q) \\ & \pi \leftarrow \mathsf{Open}(ck_{poly}, p, q; \omega) \end{array}\right] = 1 \tag{5.1}$$

- Extractable: We say that our Snark is extractable if for every size bound $\mathsf{D} \in \mathbb{D}$, every efficient adversary $\mathcal{A}_1$ there exists an efficient extractor $\mathcal{E}$ such that for every $\mathcal{A}_2$ the following holds:

$$\Pr\left[\begin{array}{c|c} & \mathsf{ck}_{\mathsf{eval}}, \mathsf{ck}_{\mathsf{poly}}, \mathsf{rk}_{\mathsf{poly}}, \mathsf{td} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{D}) \\ \mathsf{Check}(\mathsf{rk}_{\mathsf{poly}}, \mathsf{c}, q, v, \pi) = 1 & (\mathsf{C}_{\mathsf{eval}}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{ck}_{\mathsf{eval}}, \mathsf{ck}_{\mathsf{poly}}, \mathsf{rk}_{\mathsf{poly}}; \mathsf{z}) \\ \Downarrow & c \leftarrow \mathsf{interpolate}(\mathsf{ck}_{\mathsf{eval}}, \mathsf{c}_{\mathsf{eval}}) \\ deg(p) \leq D \;\; and \;\; v = \mathbf{p}(q) & \mathbf{p} \leftarrow \mathcal{E}^{A_1}(\mathsf{ck}_{\mathsf{eval}}, \mathsf{ck}_{\mathsf{poly}}, \mathsf{rk}_{\mathsf{poly}}; \mathsf{z}) \\ & v, \pi, q \leftarrow \mathcal{A}_2(\mathsf{ck}_{\mathsf{eval}}, \mathsf{ck}_{\mathsf{poly}}, \mathsf{rk}_{\mathsf{poly}}) \end{array}\right] = \mathsf{negl}(\lambda) \tag{5.2}$$

- Zero knowledege: There exists a polynomial-time simulator $\mathsf{S} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ such that, for every maximum degree bound $D \in \mathbb{N}$, and efficient adversary $\mathcal{A} = (\mathcal{A}_1)$, both of the worlds are perfectly indistinguishable.

Real World	Ideal World
$\mathsf{ck}_{\mathsf{eval}}, \mathsf{ck}_{\mathsf{poly}}, \mathsf{rk}_{\mathsf{poly}} \leftarrow \mathsf{PEC.Setup}(1^\lambda, \mathsf{D})$	$\mathsf{ck}_{\mathsf{eval}}, \mathsf{ck}_{\mathsf{poly}}, \mathsf{rk}_{\mathsf{poly}}, \mathsf{td} \leftarrow \mathsf{S.Setup}(1^\lambda, \mathsf{D})$
$\mathsf{evals}, \mathsf{points}, \mathsf{q}; \omega \leftarrow \mathcal{A}_1(\mathsf{ck}_{\mathsf{eval}}, \mathsf{ck}_{\mathsf{poly}}, \mathsf{rk}_{\mathsf{poly}})$	$\mathsf{evals}, \mathsf{points}, \mathsf{q}; \omega \leftarrow \mathcal{A}_1(\mathsf{ck}_{\mathsf{eval}}, \mathsf{ck}_{\mathsf{poly}}, \mathsf{rk}_{\mathsf{poly}})$
$\mathsf{c}_{\mathsf{eval}} \leftarrow \mathsf{Commit}_{\mathsf{eval}}(\mathsf{ck}_{\mathsf{eval}}, \mathsf{evals}, \mathsf{points}; \omega)$	$\mathsf{c}_{\mathsf{eval}} \leftarrow \mathsf{Commit}_{\mathsf{eval}}(\mathsf{ck}_{\mathsf{eval}}, \mathsf{evals}, \mathsf{points}; \omega)$
$c \leftarrow \mathsf{PEC.interpolate}(\mathsf{ck}_{\mathsf{eval}}, \mathsf{c}_{\mathsf{eval}})$	$c \leftarrow \mathsf{PEC.interpolate}(\mathsf{ck}_{\mathsf{eval}}, \mathsf{c}_{\mathsf{eval}})$
$\pi \leftarrow \mathsf{Open}(ck_{poly}, p, q; \omega)$	$\pi \leftarrow \mathsf{S.Open}(ck_{poly}, 0_{poly}, v, q; \omega)$

(5.3)

5.2 INDEXED RELATIONS WITH COMMITMENTS

We extend the indexed relations defined in Section 2.5.1 to the following indexed commitment relations. Given indexed relation $\mathrm{x}\mathcal{R}$ and $\mathsf{ck}_{\mathsf{eval}} = [c_{eval_i}]_{i=1}^{n}$ from the above described polynomial commitment scheme, we define:

$$\mathcal{R}_{\mathsf{ck}_{eval}} := \{(\mathbf{C_x}, \mathbb{i}, \mathbb{x}, \mathbf{r}, \mathbb{w})) : \forall i \; C_{x_i} = \mathsf{PEC.Commit}_{\mathsf{eval}}(\mathsf{ck}_{\mathsf{eval}_i}, x_i, i, r_i) \wedge (\mathbb{x}, \mathbb{i}, \mathbb{w}) \in \mathcal{R}\} \quad (5.4)$$

Put simply, an adaptive zk-SNARK is a zk-SNARK for indexed relation with the constraint that inputs must satisfy the commitments. We next define our adaptive ZK-SNARK definition:

5.2.1 Adaptive Preprocessing Arguments with Universal SRS

Let PEC(Polynomial Evaluation commitment scheme) be a scheme for committing polynomial evaluations generated for a suitable degree bound d sufficient to capture the largest index of interest. Further, let $\mathsf{PEC.Setup}(1^\lambda, D) \rightarrow \mathsf{ck}_{\mathsf{eval}}, \mathsf{ck}_{\mathsf{poly}}, \mathsf{rk}_{\mathsf{poly}}, \mathsf{td}$.

We define an adaptive Preprocessing arguments with universal SRS for extractable trapdoor commitment scheme PEC and relation $\mathsf{R}_{\mathsf{ck}_{\mathsf{eval}}}$ is a tuple of four algorithms $(\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V})$.

- $\mathsf{G}(\mathsf{N}) \rightarrow \mathsf{srs}$: is ppt generator which when given a size bound $\mathsf{N} \in \mathbb{N}$, samples an SRS srs that supports indices of size up to N.
- $\mathsf{I}^{\mathsf{srs}}(\mathbb{i}) \rightarrow \mathsf{ipk}, \mathsf{ivk}$: The indexer algorithm takes in a circuit index $\mathbb{i} < \mathsf{N}$ outputs indexer proving key and indexer verification key.
- $\mathsf{P}(\mathsf{ipk}, \mathsf{ck}_{\mathsf{eval}}, \mathbf{C_x}, \mathbf{x}, \mathbf{r}, \mathbf{w}) \rightarrow \pi$: The prover is a ppt which on input ipk, evalution committer keys $\mathsf{ck}_{\mathsf{eval}}$, statement $\mathbf{x}$, commitment randomness $\mathbf{r}$ and witness $\mathbf{w}$ outputs a proof π. Note that ipk also contains $\mathsf{ck}_{\mathsf{poly}}$.
- $\mathsf{V}(\mathsf{ivk}, \mathsf{rk}_{\mathsf{poly}}, \mathbf{C_x}, \pi) \rightarrow \{0, 1\}$: V is a ppt which upon input index verification key ivk, polycommit receiver key $\mathsf{rk}_{\mathsf{poly}}$, polynomial commitment $\mathbf{C_x}$ and a proof π outputs either 0 or 1. In the subsequent definitions, we model $\mathsf{V}(\mathsf{ivk}, \mathbf{C_x}, \pi)$ assuming r_{poly} is a part of ivk.

We next list the properties which we want our definition to satisfy:

- Perfect Completeness:

$$\Pr\left[\begin{array}{c}(\mathbf{C_x},\mathbb{i},\mathbb{x},\mathbb{r},\mathbb{w})\notin\mathcal{R}_{\mathsf{ck}_{eval}}\\ \vee\\ \langle\mathsf{P}(\mathsf{ipk},\mathsf{ck_{eval}},\mathbf{C_x},\mathbf{x},\mathbf{r},\mathbf{w}),\mathsf{V}(\mathsf{ivk},\mathbf{C_x})\rangle=1\end{array}\middle|\begin{array}{c}\mathsf{srs}\leftarrow\mathsf{G}(\mathsf{N})\\ \mathsf{ck_{eval}}\leftarrow\mathsf{PEC.Setup}\\ (\mathbf{C_x},\mathbb{i},\mathbb{x},\mathbb{r},\mathbb{w})\leftarrow\mathcal{A}(\mathsf{srs})\\ \mathsf{ipk},\mathsf{ivk}\leftarrow\mathsf{I}^{\mathsf{srs}}(\mathbb{i})\end{array}\right]=1 \quad (5.5)$$

- Extractable: We say that our Snark 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

$$\Pr\left[\begin{array}{c}\\ (\mathbf{C_x},\mathbb{i},\mathbb{x},\mathbb{r},\mathbb{w})\notin\mathcal{R}_{\mathsf{ck}_{eval}}\\ \wedge\\ \langle\mathsf{A}_2(\mathsf{st}),\mathsf{V}(\mathsf{ivk},\mathbf{C_x})\rangle=1\\ \\ \end{array}\middle|\begin{array}{c}\mathsf{srs}\leftarrow\mathsf{G}(1^\lambda,\mathsf{N})\\ \mathsf{ck_{eval}}\leftarrow\mathsf{PEC.Setup}\\ (\mathbf{C_x},\mathbb{i},\mathsf{st})\leftarrow\mathcal{A}_1(\mathsf{srs};\mathsf{z})\\ (\mathbb{x},\mathbb{r},\mathbb{w})\leftarrow\mathcal{E}^{\mathcal{A}_1}(\mathsf{srs};\mathsf{z})\\ \mathsf{ipk},\mathsf{ivk}\leftarrow\mathsf{I}^{\mathsf{srs}}(\mathbb{i})\end{array}\right]=\mathsf{negl}(\lambda) \quad (5.6)$$

- Zero Knowledge: We say that our Snark is zero knowledge if there exists a simulator $\mathsf{S}=(Setup,Prove)$ if for every efficient adversary $\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)$ it holds that

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

$$\Pr\left[\begin{array}{c}(\mathbf{C_x},\mathbb{i},\mathbb{x},\mathbb{r},\mathbb{w})\in\mathcal{R}_{\mathsf{ck}_{eval}}\\ \wedge\\ \langle\mathsf{S.Prove}(\mathsf{td_{srs}},\mathsf{td_{comm}},\mathbf{C_x},\mathbb{i}),\mathcal{A}_2(\mathsf{st})\rangle=1\end{array}\middle|\begin{array}{c}\mathsf{srs},\mathsf{td_{comm}},\mathsf{td_{srs}}\leftarrow\mathsf{S.Setup_1}(\mathsf{N})\\ (\mathbf{C_x},\mathbb{i},\mathbb{x},\mathbb{r},\mathbb{w}),\mathsf{st})\leftarrow\mathcal{A}_1(\mathsf{srs})\end{array}\right] \quad (5.7)$$

5.3 SECURE FUNCTION EVALUATION FROM ADAPTIVE SNARKS

Next, we show how to securely evaluate a function for auditable MPC using the above ZKSnark. We consider two scenarios:

- The first scenario lets us capture the case where all function is securely evaluated without any of the parties learning any input.
- The second scenario lets us capture the case where the correct output is delivered even when all the servers are corrupted.

As an extended version of this, we plan to include proof in the Universal Composable(UC)[65] setting, but for the scope of this thesis, we only write a simple explanation.

5.3.1 Conditional Privacy: $f \leq t$ case

First, we show an informal description of the functionality for the private function execution that is satisfied when $f \leq t$. Figure 5.1 shows the description for an ideal functionality

Ideal functionality for secure private MPC

Data-clients $I_1, I_2, \ldots, I_m$ have input $\mathbf{x_i} \in \mathbb{F}$ respectively. Input Clients I_{inp} also knows the function f which they want to compute their data upon. Adversary can corrupt upto $f \leq t$ servers and any number of corrupt input parties.

- Honest Input parties send *bid* to the Ideal functionality
- Adversary sends inputs *bid* of the corrupted parties to the ideal functionality. The adversary can modify the bids or not send the bids at all. (denoted by $\perp$)
- Adversary chooses a function f and sends it to ideal functionality.
- Ideal functionality computes the function $y = f(x_1, \ldots, x_m)$. $y = \perp$ if any $x_i = \perp$.
- Ideal functionality sends y to all the parties and the adversary.

Figure 5.1: Conditional private function evaluation

The adversary learns all the inputs of the corrupt parties x_i, the function f and the output of function y.

5.3.2 Unconditional Correctness

Next we show an informal description of the functionality for the unconditional correct case. Figure 5.2 shows the description for an ideal functionality

Ideal functionality for Unconditionally Correct function execution

Data-clients $I_1, I_2, \ldots, I_m$ have input $\mathbf{x_i} \in \mathbb{F}$ respectively. Input Clients I_{inp} also knows the function f which they want to compute their data upon. Adversary can corrupt upto $t \leq n$ (possibly all) servers and any number of corrupt input parties.

- Honest Input parties send *bid* to the Ideal functionality
- Adversary sends inputs *bid* of the corrupted parties to the ideal functionality.
- The Ideal functionality sends inputs of **all** honest parties to the adversary.
- Input client chooses a function f and sends it to ideal functionality.
- Ideal functionality computes the function $y = f(x_1, \ldots, x_m)$. $y = \perp$ if any $x_i = \perp$.
- Ideal functionality then sends y to the adversary and gets value r. Modeling aborts
- If $r = \perp$, send $\perp$ to the input parties or otherwise send y to all the parties(honest and malicious).

Figure 5.2: Unconditionally Correct function execution

The adversary learns all the inputs of the corrupt parties x_i, the function f and the output of function y. This functionality guarantees input privacy, correctness but not the independence of inputs, fairness, and guaranteed output delivery.

CHAPTER 6: OUR CONSTRUCTION

6.1 CONSTRUCTIONS FOR POLYNOMIAL COMMITMENT SCHEME

We now show two constructions for PEC schemes. The first construction based on Pedersen commitments PEC.Peddoes not require any trusted setup, while the second scheme based on Lipmaa's construction [32] PEC.Poly uses polynomial commitments that require a trusted setup. On the other hand, the verification in PEC.Ped requires n exponentiation and n group multiplications whereas PEC.Poly requires only n group operations. Our implementation uses PEC.Ped scheme for faster prototyping.

6.1.1 Construction using Pedersen Commitments

Our polynomial evaluation commitment scheme PEC.Ped over a cyclic group $\mathbb{G}$ is constructed as follows: $(\mathsf{Setup}, \mathsf{Commit}_{\mathsf{eval}}, \mathsf{Interpolate}, \mathsf{Open}, \mathsf{Check})$.

- $\mathsf{Setup}(1^\lambda, D) \rightarrow \mathsf{ck}_{\mathsf{eval}}, \mathsf{ck}_{\mathsf{poly}}, \mathsf{rk}_{\mathsf{poly}}, \mathsf{td}$: Sample random generators g and $h = g^{\mathsf{td}}$ and return $\mathsf{ck}_{\mathsf{eval}} = (g, h), \mathsf{ck}_{\mathsf{poly}} = (g, h), \mathsf{rk}_{\mathsf{poly}} = (g, h), \mathsf{td}$.

- $\mathsf{Commit}_{\mathsf{eval}}(\mathsf{ck}_{\mathsf{eval}}, \mathsf{evals}, \mathsf{points}; \omega) \rightarrow \mathsf{c}_{\mathsf{eval}}$ Parse $\mathsf{ck}_{\mathsf{eval}} = (g, h)$. Then the commitment to a evaluations $\mathsf{evals} = [eval_i]_{i=1}^n$ returns $\mathsf{c}_{\mathsf{eval}} = [g^{eval_i} h^{r_i}]_{i=1}^n$ where r_i is sampled randomly according to randomness ω.

- $\mathsf{interpolate}(\mathsf{ck}_{\mathsf{poly}}, \mathsf{c}_{\mathsf{eval}}) \rightarrow c$: $\mathbf{c} = \mathsf{ck}_{\mathsf{eval}}$

- $\mathsf{Open}(\mathsf{ck}_{\mathsf{poly}}, \mathbf{p}, q; \omega) \rightarrow v, \pi$: Obtain the interpolated randomness r_i from ω(must the be same as the one used for $\mathsf{Commit}_{\mathsf{eval}}$). Interpolate $\mathbf{r}$ polynomial with evaluations r_i and return $v, \pi = (\mathbf{p}(q), \mathbf{r}(q))$.

- $\mathsf{Check}(\mathsf{rk}_{\mathsf{poly}}, \mathsf{c}, \mathsf{q}, \mathsf{v}, \pi) \in \{0, 1\}$: Parse $\mathsf{rk}_{\mathsf{poly}} = (g, h)$, $\mathsf{c} = [g^{eval_i} h^{r_i}]_{i=1}^n$, Check

$$\Pi_{i=1}^n [(g^{\mathsf{eval_i}} h^{r_i})^{\ell_i(q)}] \stackrel{?}{=} g^v h^\pi \tag{6.1}$$

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

6.1.2 Construction using Polynomial Commitments

Our construction for Auditable MPC on Marlin relies on creating a Polynomial commitment to a polynomial from commitments off the shares of evaluations. Our key idea is that 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. For our construction, it is beneficial to use this scheme instead of the previous one based on Pedersen 6.1.1 commitments because 1) we are already using polynomial commitments in an underlying protocol, so the setup step is already performed and 2) this commitment scheme is compatible with polynomial commitment batching techniques that we can use for more efficiency.

We first list some preliminaries that we use in this scheme. Let $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$ be a evaluation vector of length n which we wish to commit. Further, let For simplicity, we assume n is a power of two, and let

- ω be the n-th primitive root of unity in a field $\mathbb{F}_p$.
- $v_H(X)$ denote the vanishing polynomial over the multiplicative subgroup set H defined by $\Pi_{i=0}^{i=n-1}(X - \omega^i) = X^n - 1$. Note that $v_H(\omega^i) = 0 \forall i \in \mathbb{N}$.
- $\ell_i(X)$ be $\Pi_{j \neq i} \frac{X - \omega^j}{\omega i - \omega^j}$ be ith Lagarange polynomial that 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. another popular choice for independent polynomials in polycommit scheme is $1, X, X^2, \ldots, X^{n-1}$.

$$L_{\mathbf{a}}(X) = \Sigma_{i=1}^{n} a_i \ell_i(X) \tag{6.2}$$

We define our scheme base don Lip'16 [32]'s construction with three major adaptations.

- Lip'16 schemes uses evaluations of $\ell_i(\beta)$ for a trapdoor secret β. We instead use the standard $1, X, X^2$, and create a polynomial commitment using Kate style polynomial commitments. Since our ZKsnark for Marlin uses the same CRS, it allows us to use standard polycommit proof batching techniques for efficiency.
- Lip'16 scheme was based on PKE assumptions and hence required 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 highlight how to construct a polynomial commitment for giving evaluation proofs of polynomial, which is not shown in Lip'16. (Lip'16 was designed to commit and prove snark and not as polycommit scheme, so it was not required for them to satisfy the evaluation proof properties).

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. Let $\mathsf{PC} = \mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check}$ be the polynomial commitment scheme as mentioned in 2.3.2. We define PEC.Poly as follows:

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

$$\Sigma := \begin{pmatrix} g & g^{\alpha} & g^{\alpha^2} & \dots & g^{\alpha^D} \\ g^{\gamma} & g^{\alpha\gamma} & g^{\gamma\alpha^2} & \dots & g^{\gamma\alpha^D} \end{pmatrix}. \tag{6.3}$$

$$\mathsf{ck}_{\mathsf{poly}} := (\Sigma, D), \mathsf{ck}_{\mathsf{eval}} := ([g^{\ell_i(X)}]_{i=1}^n, g^{v_H(X)}), \mathsf{rk}_{\mathsf{poly}} := (D, g^{\gamma}, h^{\alpha}), \mathsf{td} := (\alpha, \gamma) \tag{6.4}$$

- $\mathsf{Commit}_{\mathsf{eval}}(\mathsf{ck}_{\mathsf{eval}}, \mathsf{evals}, \mathsf{points}; \omega) \to \mathsf{c}_{\mathsf{eval}}$: Parse $([g^{\ell_i(X)}]_{i=1}^n, g^{v_H(X)}) = \mathsf{ck}_{\mathsf{eval}}$ and parse points as $\mathsf{points} = [\omega^j]_{i=1}^n$ for some j. Compute the commitment to evaluations $\mathsf{evals} = [\mathsf{eval_i}]_{i=1}^n$ as follows: return $\mathsf{c}_{\mathsf{eval}} = [(g^{\ell_j(X)})^{\mathsf{eval_j}} g^{r_i \mathsf{v_H}(x)}]_{i=1}^n$ where r_i is sampled randomly according to randomness ω. Note that all the terms can be computed directly from the $\mathsf{ck}_{\mathsf{eval}}$.

- $\mathsf{interpolate}(\mathsf{ck}_{\mathsf{poly}}, \mathsf{c}_{\mathsf{eval}}) \to c$: return $c = \Pi_{i=1}^n \mathsf{c}_{\mathsf{eval,i}}$. Note that this returns a polynomial commitment to the polynomial whose shares are committed by $\mathsf{c}_{\mathsf{eval}}$.

- $\mathsf{Open}(\mathsf{ck}_{\mathsf{poly}}, \mathbf{p}, q; \omega) \to v, \pi$: Same as regular polycommit open operation as described in Sec 2.3.2

- $\mathsf{Check}(\mathsf{rk}_{\mathsf{poly}}, \mathsf{c}, \mathsf{q}, \mathsf{v}, \pi) \in \{0, 1\}$: Same as regular polynomial commitment operation as described in Sec 2.3.2

6.2 CONSTRUCTION OF ADAPTIVE ZK-SNARK

Our construction for an adaptive Preprocessing arguments$((\mathsf{G}, \mathsf{I}, \mathsf{P}, \mathsf{V}))$ with universal SRS for extractable trapdoor commitment scheme PEC and relation $\mathsf{R}_{\mathsf{ck}_{\mathsf{eval}}}$ is shown in 6.1. It is possible to instatiate PEC with any of the above PEC.Ped or PEC.Poly

Next, describe how to implement the Marlin prover algorithm by using Multi-party computation where the servers only know shares to witness and statement and wish to compute

Adaptive Zk-Snark using Marlin

Let $\mathsf{PEC} = (\mathsf{Setup}, \mathsf{Commit}_{\mathsf{eval}}, \mathsf{Interpolate}, \mathsf{Open}, \mathsf{Check})$ be a polynomial evaluation commitment scheme. We can instantiate our scheme with any of the above constructions PEC.Ped or PEC.Poly. Further let $\mathsf{G_m}, \mathsf{I_m}, \mathsf{P_m}, \mathsf{V_m}$ be a preprocessing argument from Marlin. Let $\mathbf{C_x}$ be the commitments to the statement $\mathbb{x}$ with the associated randomness $\mathbf{r}$. Prover knows $(\mathbb{i}, \mathbb{x}, \mathbb{r}, \mathbb{w}, \mathbf{C_x})$. Verifier only knows $\mathbf{C_x}$.

- $\mathsf{G}(\mathsf{N}) \rightarrow \mathsf{srs}$: Use the same Σ as defined in PEC.Setup and obtain $\mathsf{ck}, \mathsf{rk} := \mathsf{ck_{poly}}, \mathsf{rk_{poly}}$
- $\mathsf{I}^{\mathsf{srs}}(\mathbb{i}) \rightarrow \mathsf{ipk}, \mathsf{ivk}$: Our indexer would be same as the indexer in Marlin $\mathsf{I_m}$. Same as in Marlin, our indexer would output commitments to polynomials indexed by R1CS matrices A, B and C.
- $\mathsf{P}(\mathsf{ipk}, \mathsf{ck_{eval}}, \mathbf{C_x}, \mathbf{x}, \mathbf{r}, \mathbf{w}) \rightarrow \pi$:
  1. Sample a random statement element $x^{\mathbb{b}}$ from the domain of statement elements and create a evaluation commitment to it using $\mathbf{C}_{\mathbf{x}}^{\mathbb{b}} \leftarrow \mathsf{PEC.Commit_{eval}}(\mathsf{PEC.ck_{eval}}, \mathsf{x}^{\mathbb{b}}, .; \omega)$
  2. Use augmented statement commitment $\mathbf{C}'_{\mathbf{x}} = (\mathbf{C_x}, \mathbf{C}_{\mathbf{x}}^{\mathbb{b}})$ as the starting transcript for Fiat Shamir and execute the marlin prover with updated statement as $x||x^{\mathbb{b}}$, i.e execute $\mathsf{P_m}(\mathsf{ipk}, x||x^{\mathbb{b}}, w)$ to obtain π_m.
  3. Let $\hat{x}(X)$, β_1 be the interpolated polynomial used the statement polynomial by the prover and prover second challenge respectively. Invoke $\mathsf{PEC.open}(\mathsf{ck_{poly}}, \hat{x}(X), \beta_1; \omega)$ to get evaluation and it's corresponding proof as $\hat{(}x)(\beta_1), \pi_c$.
  4. return $\pi = (\pi_m, \pi_c, \hat{x}(\beta_1), \mathbf{C}_{\mathbf{x}}^{\mathbb{b}})$.
- $\mathsf{V}(\mathsf{ivk}, \mathsf{rk_{poly}}, \mathbf{C_x}, \pi) \rightarrow \{0, 1\}$:
  1. Parse proof π as $(\pi_m, \pi_c, \hat{x}(\beta_1), \mathbf{C}_{\mathbf{x}}^{\mathbb{b}})$ and compute augmented statement commitment as $\mathbf{C}'_{\mathbf{x}} = (\mathbf{C_x}, \mathbf{C}_{\mathbf{x}}^{\mathbb{b}})$
  2. Invoke the marlin verifier routine $b_m \leftarrow \mathsf{V'_m}(\mathsf{ipk}, , \hat{x}(\beta_1), \pi)$. which is same as $\mathsf{V_m}(\mathsf{ipk}, x, \pi)$ but 1) uses $\mathbf{C_x}$ as Fiat Shamir transcript and 2) directly uses the value $\hat{x}(\beta_1)$ instead of computing it from the x statement. Let b_m be the returned bit by the verifier.
  3. Obtain a polynomial commitment $C_x\hat{x}$ from $\mathsf{PEC.Interpolate}(\mathbf{C}'_{\mathbf{x}})$
  4. Invoke $\mathsf{PEC.check}(\mathsf{vk_{poly}}, C_{\hat{x}}, \beta_1, \hat{x}(\beta_1))$ to get result bit b_c.
  5. return 1 iff both $b_m = 1$ $b_c = 1$.

Figure 6.1: MPC input independent processing

the MPC proof. Let us first recap about the Offline operations required for our construction. We then next suggest our protocol for auditable MPC, which uses this MPC prover.

6.3 AUDITABLE MPC: OFFLINE PHASE

6.3.1 Marlin Universal Setup

Similar to Marlin [2] and Sonic [26], we prove security under the universal SRS model where parties have access to long structured universal reference string. Since our srs is the same as of Marlin, our construction also follows the updatable srs property. Informally, SRS is updatable [35] if there exists an update algorithm that can be run at any time by anyone to update the SRS, with the guarantee that security soundness, completeness, and zero-knowledge holds as long as there is at least one honest updater since the beginning of time. This property has practical significance as it helps in organizing MPC ceremonies for creating this reference strings [64] [10]. We refer the reader to Groth's first work [35] on the formal definitions of updatable srs and ignore the details in favor of simplicity.

Depending on the model of the adversary in ZK proof games, we need to sample different crs. Similar to Sonic [26] and Marlin [2], we consider Algebraic Group Model(AGM) by Fuchsbauer et al. [66] which requires a simpler CRS. Note that we could also use the plain model to prove Marlin under knowledge assumptions, but comes at the cost of double the proof elements. Appendix B in Marlin [2] shows how to modify the construction with a different crs to add proof under standard knowledge assumptions in the plain model.

Universal srs construction: On input a security parameter λ, and a maximum degree bound $\mathbf{D} \in N$, PC.Setup samples public parameters committer key and receiver key. $\langle group \rangle = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g, h, e)$ 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.

$$\Sigma := \begin{pmatrix} g & g^{\alpha} & g^{\alpha^2} & \dots & g^{\alpha^D} \\ g^{\gamma} & g^{\alpha\gamma} & g^{\gamma\alpha^2} & \dots & g^{\gamma\alpha^D} \end{pmatrix}. \tag{6.5}$$

$$ck := (\mathbf{\Sigma}, D), rk := (D, g^{\gamma}, h^{\alpha})$$

6.3.2 Generating the MPC pre-processing elements

Pre-processed elements are used in MPC to speed the online computation. Beaver triples [6] are used for the evaluation of multiplications in an MPC setting with t-shares. Each beaver

triple is a triplet of t-shares $[\![x]\!]_t$, $[\![y]\!]_t$ and $[\![z]\!]_t$ such that $z = x * y$. We can use these triples to evaluate the multiplication of shares $[a]_t$ and $[b]_t$ by making use of the following equations.

$$\begin{aligned} M &= \mathsf{Open}([\![a]\!]_t - [\![x]\!]_t) \\ N &= \mathsf{Open}\left([\![b]\!]_t - [\![y]\!]_t\right) \\ [\![ab]\!]_t &= M * N + M * [\![y]\!]_t + N * [\![x]\!]_t + [\![z]\!]_t \end{aligned} \tag{6.6}$$

The shares of random numbers are generated by sampling a random polynomial of degree t and distributing the shares to all parties. We assume a robust offline phase to generate these triples.

6.3.3 Marlin Circuit Indexing

Informally, the indexer $\mathbf{I}$ receives as input the index $\mathbb{i}$ to be pre-processed and outputs one or more univariate polynomials over $\mathbb{F}$ encoding $\mathbb{i}$. The uni-variate polynomials are then committed to using a extractable polynomial commitment scheme. For our purpose, it is essential that the indexer process for generating polynomial commitments to the indexer polynomials, as well as the creation of index polynomials, is entirely deterministic for a given circuit and srs. Anyone can publicly pre-process a given index (e.g., a circuit) in an offline phase, in order to avoid incurring costs related to the index in (any number of) subsequent online phases that check different instances. For our purpose, we assume that the commitments to the indexed polynomials for circuits of interest are already published onto the bulletin board. We refer the reader to section 4 of Marlin [2] for details of the indexer algorithm.

6.4 ONLINE PHASE: CLIENT INPUT PROCESSING

Every Data Input client I_i would carry out to following protocol to submit its input x_i to the MPC servers. We describe the protocol in two modes, the first mode does not provide input Independence, but the second mode provides input Independence. Note that in the reactive MPC scenario, the only first mode is applicable. The steps in blue show the steps required for operating in the second mode.

At a high level, the above simply provides the servers with shares of client inputs and the randomness used with an additional commitment check to ensure against malicious clients. We present two different schemes for evaluating the Comm operation in the above scheme. The difference between the two schemes does not show up in the client input phase but in

Client input processing for ensuring input independence

Data-clients $I_1, I_2, \ldots, I_m$ have input $\mathbf{x_i} \in \mathbb{F}$ respectively. Input Clients I_{inp} also knows the function f which they want to compute their data upon.

1. Every I_i get inputs shares of two random pre-processed value $[\![r_{1,i}]\!]$ and $[\![r_{2,i}]\!]$ from the MPC servers.
2. Client opens $r_{1,i} = \mathsf{open}([\![r_{1,i}]\!])$ and $r_{2,i} = \mathsf{open}([\![r_{2,i}]\!])$, samples another random value r_{x_i} sends $y_i = r_{1,i} + x_i$ and $r_i = r_{2,i} + r_{x_i}$.
3. The client posts the commitment $C_{x_i} = \mathsf{Comm}(\mathsf{x_i}, \mathsf{r_{x_i}})$ to the bulletin board.
4. In the single-shot scenario carry out the protocol listed in 2.2
5. Finally servers compute $[\![x_i]\!] = y_i - [\![r_i]\!]$, $[\![r_{x_i}]\!] = y_i' - [\![r_i']\!]$ and save $[\![x_i]\!]$ as the party input share. The servers check the commitment to $C_{x_i} = \mathsf{Comm}([\![x_i]\!], [\![r_{x_i}]\!])$. The servers abort the protocol if the check fails.

Figure 6.2: MPC input processing

auditor cost, which we will describe later.

- Clients make a Pederson commitment of the form $C_{x_i} = g^{x_i} h^{r_{x_i}}$ and servers use protocol mentioned in Figure 6.4 to check whether the commitment is correct
- Clients our new protocol as mentioned in section 6.1.2 to commit to the inputs. Servers can similarly use a variant of figure 6.4 to check the commitment.

6.5 ONLINE PHASE: COMPUTING QAP AND WITNESS

It is worth noting that the MPC computation described and the SNARK circuit might not be the same. For SSS based MPC, if the circuit is described as an arithmetic circuit, the values for witnesses are exactly the values for the ZKSNARK. If a custom MPC protocol is used, one has to handcraft an R1CS corresponding to that protocol to generate a proof for it. It remains an interesting research question to create efficient SNARK circuits from the MPC program specification. What remains is how to compute the solution of the QAP using multi-party computation. Trinocchio [33] suggests a protocol for handling split gate and inversion MPC gates. For simplicity, we restrict ourselves to the case where the input is provided as an arithmetic circuit.

After receiving the Input circuit, the servers carry out the MPC protocol to evaluate it and store the witness for proof computation.

6.6 ONLINE PHASE: GENERATION OF MARLIN PROOF BY MPC

We now show how to adapt the existing marlin protocol to make it adaptive and suitable for auditable MPC application. Before we go into technical details of marlin, we first revise some of the notations used in this section.

Notation: Informally, R1CS is a relation given by the tuple, $(i, x, w) = ((A, B, C), x, w)$ where i is index of the relation, x is the statement and w is the witness and $z = (x, w)$ such that $Az \circ Bz = Cz$. Let n_A, m_A denote the number of variables of number of constraints in matrix A and $|A|$ denote the number of non-zero elements in A. All $\hat{g}$ denote a low degree extension of the function g. $\mathbb{F}$ denotes the field on primes, $g(X) \in \mathbb{F}^d[X]$ denotes a polynomial of degree at most d over a field $\mathbb{F}$. b denotes the bound on number of queries to the polynomial. H, K, X denote the multiplicative subgroups of $\mathbb{F}$ such that $|H|$ is minimal size such that $|H| \geq max(n_A, m_A, n_B, m_B, n_C, m_C)$, $|K| \geq max(|A|, |B|, |C|)$ and $|X| \geq |x|$. $v_H(X)$ denotes the vanishing polynomial on multiplicative group H. PC be a polycommit scheme which is extractable and hiding with standard definitions for $\mathsf{PC.setup}$, $\mathsf{PC.commit}$, $\mathsf{PC.open}$, $\mathsf{PC.createwitness}$, $\mathsf{PC.verifyeval}$. $r(X, Y)$ denotes the derivative polynomial over H defined by: $r(X, Y) = \frac{X^{|H|} - Y^{|H|}}{X - Y}$. Let M denote the set of matrices A, B, C and $r_M(X, Y) = \sum_{\kappa \in H} r(X, \kappa)\hat{M}(\kappa, Y)$. Let $C_{p(X)}$ denote a polynomial commitment of the poly $p(X)$ which is extractable and hiding.

In our setting, we are running n instances of marlin protocol, one at each server. Each server computes the R1CS circuit in the marlin and calculates the share of the output. In our application, let the MPC circuit have k number of outputs. In case of reactive MPC, the statement $\mathbf{x}$ would be $(x_1, x_2, \ldots, x_m, o_1, o_2, \ldots, o_k)$ where $(x_1, x_2, \ldots, x_m,$ are inputs to the MPC statement and $o_1, o_2, \ldots, o_k$ are outputs of the MPC computation. Whereas in the scenario of one-shot MPC, where the output is revealed, the statement $\mathbf{x}$ only consists of $(x_1, x_2, \ldots, x_m$. All other wire values which are not input or output would be the witness. Values in $[\![.]\!]$ denote the secret shared elements while the values in blue denote the public values.

6.7 HIGH-LEVEL OVERVIEW OF PROOF GENERATION

In the online phase, the clients and servers engage in an MPC protocol. We are not instantiating N MPC provers one at each server; rather, the MPC servers combined together act as a single prover with each having individual share elements. While engaging in the MPC protocol, clients post commitments to their inputs onto the bulletin board, whereas all MPC servers together act of the prover and post the protocol messages to the bulletin

board. In each round of the marlin protocol, the servers use MPC to generate commitments to certain polynomials from shares of evaluations of the polynomials. In the last round, based on the challenges from the verifier, the servers again engage in an MPC protocol to provide evaluation proofs of the polynomials at the verifier chosen challenge points. The MPC servers post these polycommits, evaluation proofs, and another marlin interactive proof message onto the bulletin board.

After a round MPC protocol has been executed, the auditor looks at the proof messages from the bulletin board and verifies the proof messages and outputs True with overwhelming probability if and only if the protocol was executed correctly. Next, show the adaptations we had to make for each round.

6.7.1 Prover Initialize

1. MPC servers engage in protocol Figure 6.4 to create commitments 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 $\mathtt{b} = 1$ 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.

6.7.2 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

$$\hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) = h_0(X)v_H(X) \tag{6.7}$$

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 $\mathtt{b}$ is an additional degree added for zero-knowledge.

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

$$[\![h_0(X)]\!] = \frac{[\![\hat{z}_A(X)]\!][\![\hat{z}_B(X)]\!] - [\![\hat{z}_C(X)]\!]}{v_H(X)} \tag{6.8}$$

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 σ_1 round.

The servers use protocol from fig 6.5 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 α, $\eta_A, \eta_B, \eta_C \in \mathbb{F}$. 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.

6.7.3 Marlin Second Round

Prover computes the polynomial

$$q_1(X) = s(X) + r(\alpha, X)\left(\sum_M \eta_M \hat{z}_M(X)\right) - \left(\sum_M \eta_M r_m(\alpha, X)\right)\hat{z}(X)) \tag{6.9}$$

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

$$q_1(X) = h_1(X)v_H(X) + g_1(X)X \tag{6.10}$$

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

$$[\![q_1(X)]\!] = [\![s(X)]\!] + r(\alpha, X)\left(\sum_M \eta_M [\![\hat{z}_M(X)]\!]\right) - \left(\sum_M \eta_M r_M(\alpha, X)\right)[\![\hat{z}(X)]\!] \tag{6.11}$$

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 protocol listed in fig 6.8 to get $[\![h_1(X)]\!]$ and $[\![g_1(X)]\!]$ such that:

$$[\![q_1(X)]\!] = [\![h_1(X)]\!]v_H(X) + [\![g_1(X)]\!]X \tag{6.12}$$

and $deg([\![g_1(X)]\!]X) < deg(v_H(X))$. Recall that σ_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 fig 6.5 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 fig 6.5 to create $C_{h_1(X)}, C_{g_1(X)}$ from local polynomial shares.
3. Server challenge β_1 is sampled from $\mathbb{F}\backslash H$ based on the transcript of first transcript plus the commitments $C_{h_1(X)}, C_{g_1(X)}$.

6.7.4 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

$$q_2(X) = r(\alpha, X)\left(\sum_M \eta_M \hat{M}(X, \beta_1)\right) \tag{6.13}$$

and the sum-check result

$$\sigma_2 = \sum_{\kappa \in H} r(\alpha, \kappa)\left(\sum_{M \in A,B,C} \eta_M \hat{M}(\kappa, \beta_1)\right) \tag{6.14}$$

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

$$q_2(X) = h_2(X)v_H(X) + g_2(X)X + \sigma_2/|H| \tag{6.15}$$

and $deg(g_2(X)X) < deg(v_H(X))$. Such a division is similar to protocol 6.8 except that it is carried out in the open instead of secret shared form. Servers can then use standard $\mathsf{PC.commit()}$ to create commitments $C_{h_2(X)}, C_{g_2(X)}$ which are extractable and hiding. MPC servers sample β_2 from $\mathbb{F}\backslash H$ using Fiat Shamir using transcript upto the current round.

6.7.5 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:

$$\sum_{M \in \{A,B,C\}} \eta_M \frac{v_H(\beta_2)v_H(\beta_1)\hat{val}_M(X)}{(\beta_2 - \hat{row}_M(X))(\beta_1 - \hat{col}_M(X))} \tag{6.16}$$

$$\sigma_3 = \sum_{\kappa \in K} \sum_{M \in \{A,B,C\}} \eta_M \frac{v_H(\beta_2)v_H(\beta_1)\hat{val}_M(\kappa)}{(\beta_2 - \hat{row}_M(\kappa))(\beta_1 - \hat{col}_M(\kappa))} \tag{6.17}$$

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

$$h_3(X)v_K(X) = a(X) - b(X)(Xg_3(X) + \sigma_3/|K|) \tag{6.18}$$

. Finally, compute $C_{h_3(X)}, C_{g_3(X)}$ using PC.commit() since all polynomials are public. The prover samples sends a challenge β_3 sampled from $\mathbb{F}$ according to fiat sharmir.

6.7.6 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 β_j using `PC.createwitness`(β_j, $C_{p_i(X)}$, $p_i(X)$). To create a proof for public polynomials $p(X)$, we would standard `PC.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 fig 6.6.

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. Depending on the commitment scheme used by the clients, the proof will consist of $\hat{r}(\beta_1)$ of Pederson commitment or a PC.createwitness proof for the new scheme we mention.

6.8 AUDITOR VERIFICATION

The auditor check comprises of two main components: a) Marlin proof check and b) Input consistency check. The first check verifies that marlin proof was correctly produced and prover indeed knows the corresponding witness corresponding to the claimed evaluation of x polynomial. The second check comprises checking whether the claimed evaluation for x polynomial is consistent with the statement commitments. All the round checks are the same as they are in marlin, which we refer the reader to the Marlin paper [2].

6.8.1 Input Consistency Check

With the above checks, the auditor has verified that the MPC system computed the circuit correctly, however, the auditor still does not know whether the inputs used by the MPC system was indeed the same inputs provided by the prover. That is, it needs to check

whether the polycommit to $x(\hat{X})$ is consistent with the input commitments from the input parties.

There are two main ways of checking the consistency based on which commitment schemes the client used.

- If Pedersen Commitments are used as the choice of commitment, the proof consists of claimed evaluations $x_c(\beta_1)$ and $r_c(\beta_1)$. The verification then consists of interpolating the commitments to the statement polynomials in the exponent using 6.4 to get a value $g^{x(\beta_1)}h^{r(\beta_1)}$. The auditor can then himself compute whether $g^{x(\beta_1)}h^{r(\beta_1)} \stackrel{?}{=} g^{x_c(\beta_1)}h^{r_c(\beta_1)}$

- If our new commitment scheme is used, the client provides a proof that the commitment was evaluated correctly using standard PC.open for $x(X)$ at β_1. Note that this proof can be batched with other proofs at β_1 for efficiency. The client uses PC.interpolate() as detailed in section 6.1.2 to the commitment for verifying the proofs.

We leave evaluating the trade-offs between the commitment scheme as a part of continued work. Our initial results, we suspect that the second scheme might be more efficient than the first one because of support for batching proofs. When reporting evaluations, we only consider the implementation based on Pedersen schemes.

6.9 AUDITABLE MPC USING MARLIN

Figure 6.3 shows the auditable MPC protocol using the constructions for adaptive zk-SNARKS, marlin based MPC prover and Polynomial Evaluation commitment schemes. This construction follows the functionalities for secure private function evaluation 5.1 and unconditional correct execution 5.2. We leave the UC proof as future work.

Auditable MPC using universal SRS

Data-clients $I_1, I_2, \ldots, I_m$ have input $\mathbf{x_i} \in \mathbb{F}$ respectively. Input Clients I_{inp} also knows the function f which they want to compute their data upon. Let PEC be Polynomial Evaluation Commitment scheme as defined in section 5.1

1. Perform the Setup for the PEC scheme and universal CRS for Marlin and send the evlaution commitment keys $\mathsf{ck_{eval}}$ to clients, verifation key $\mathsf{vk_{poly}}$ to the auditor and the crs, $\mathsf{ck_{eval}}$, $\mathsf{ck_{poly}}$ to the servers.

2. For the MPC computations, robustly pre-process the MPC random values and beaver triples. For the snark computation, pre-process the indexes for known computations. That is, the servers deterministically pre-processes the indexes for set of known functions ($F_s = f_1, f_2 .... f_k$).

3. Each data-client computes commitment to the inputs (C_{x_i}) using $\mathsf{PEC.Commit_{eval}}(\mathsf{ck_{eval}}, x_i)$

4. The input-client then choose a function f to compute upon and submits it to the MPC servers for desired computation.

5. Each data-client provides the input using protocol Fig 6.2 to the MPC servers. The MPC servers obtain $[\![x_i]\!]$ and for each client check whether the inputs are consistent with the commitment C_{x_i} posted on the bulletin board by using protocol 6.5. If the commitments do not match, then the servers abort.

6. If the function f is already pre-processed (i.e $f \in F_s$), then send already pre-processed $\mathsf{index_{pk}}$ to the MPC servers and $\mathsf{index_{vk}}$ to the auditor. If the circuit is not already pre-processed, then the servers run the indexer on circuit obtain corresponding $\mathsf{index_pk}, \mathsf{index_{vk}}$ and send them to respective parties and post the $\mathsf{index_{pk}}, \mathsf{index_{vk}}$ on bulletin board.

7. Generate a adaptive zk-SNARK proof π by using the MPC prover construction in 6.6 for the statement commitments with n inputs and k server outputs $C_{stmt} = ([C_{x_i}]_{i=1}^n, [C_o]_{i=1}^k)$ using ipk obtained in the previous step.

8. The auditor audits that the function was correctly executed based on input commitments C_{x_i} from the clients, output commitments C_{o_k} from the servers, and proof π by invoking verification routine of adaptive zk-snark as desribed in 6.1

Figure 6.3: Auditable MPC with universal SRS

6.10 SUBPROTOCOLS

In this section, we list the sub-protocols which we use for different operations in our construction

Robust Interpolation in the exponent

Input: Secret shared values $[\![x_i]\!]$ from each server P_i.
Output: Pederson Commitment to x of the form $g^x h^r$.
Guarantees: If $t < n/3$, then servers don't learn anything from x.
For each MPC server P_i:

- Use pre-processed random shares $[\![r_i]\!]$ to compute share commitment $C_{[\![x_i]\!]} = g^{[\![x_i]\!]} h^{[\![r_i]\!]}$.
- Send commitment $C_{[\![x_i]\!]}$ to all other servers P_j
- Use robust interpolation in the exponent to compute the commitment $g^x h^r$. In more detail, we note that all the Lagrange polynomials are fixed for the given set of n parties. After receiving all the shares, we check easily the evaluate a candidate interpolation result polynomial $g^{p(x)} h^{p'(x)}$ at any point 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 and return $g^{f(0)} h^{r(0)}$ as $g^x h^r$

Figure 6.4: Robust Interpolation in the exponent

MPC Polynomial Commitments from shares of evaluations

Input: Secret shared evaluations of a polynomial $[\![x_{i_1}]\!], \ldots, [\![x_{i_k}]\!]$ from each server P_i.
Output: Hiding polynomial commitment $g^{p(X)} h^{r(X)}$ to polynomial $p(x)$ at degree at most $k-1$ which respects the evaluations $x_1, \ldots, x_k$.
Guarantees: If $t < n/3$, then servers don't learn anything from $p(X)$.
For each MPC server P_i:

- Obtain a sharing of the polynomial $[\![p(X)]\!]$ by locally interpolating at shares of evaluations $[\![x_{i_1}]\!], \ldots, [\![x_{i_k}]\!]$. Use pre-processed random shares $[\![r_{i_1}]\!], \ldots, [\![r_{i_k}]\!]$ to construct shares of hiding polynomial $[\![r(X)]\!]$
- Compute $C_{[\![p(X)]\!]} = \texttt{PC.commit(ck,}\ \ [\![p(X)]\!]\ \ \texttt{,}\ \ [\![r(X)]\!]\ \ \texttt{)}$ and send the commitment $C_{[\![p(X)]\!]}$ to all other parties.
- Use robust interpolation in the exponent to compute the commitment $g^{p(X)} h^{r(X)}$. The interpolation is carried out similarly to the method described in Figure 6.4

Figure 6.5: MPC Polynomial Commitments from shares of evaluations

MPC Secret shared polynomials Create Witness

Input: Secret shared polynomials: Secret shares of evaluations of a polynomial $[\![x_{i_1}]\!], \ldots, [\![x_{i_k}]\!]$ from each server P_i and a evaluation point β.

Output: Let $p(X)$ be the interpolated polynomial interpolated from the by combining the shares of the evaluations. The output consists of $(\pi, p(\beta))$ where $\pi =$ `PC.createwitness(p(X), β)`.

Guarantees: If $t < n/3$, then servers don't learn anything from $p(X)$ apart from $p(\beta)$

For each MPC server P_i:

- Obtain a sharing of the polynomial $[\![p(X)]\!]$ by locally interpolating at shares of evaluations $[\![x_{i_1}]\!], \ldots, [\![x_{i_k}]\!]$. Use same hiding polynomial $[\![r(X)]\!]$ which was used to creating the polycommit.
- Compute $[\![\phi(X)]\!] = \frac{[\![p(X)]\!] - p(\beta)}{X-\beta}$, $[\![\hat{\phi}(X)]\!] = \frac{[\![r(X)]\!] - r(\beta)}{X-\beta}$ and send the commitment $C_{[\![\phi(X)]\!]} = g^{[\![\phi(X)]\!]} h^{[\![\hat{\phi}(X)]\!]}$ along with $[\![r(\beta)]\!]$ to all other parties.
- Use robust interpolation in the exponent to compute the commitment $g^{\phi(X)} h^{r(X)}$ and $r(\beta)$. The interpolation is carried out similarly to the method described in 6.4

Figure 6.6: MPC secret shared polynomials Create witness

MPC Secret shared polynomials Evaluation

Input: Secret shared polynomials: Secret shares of evaluations of a polynomial $[\![x_{i_1}]\!], \ldots, [\![x_{i_k}]\!]$ from each server P_i and a evaluation point β.

Output: Let $p(X)$ be the interpolated polynomial interpolated from the by combining the shares of the evaluations. The output consists of $p(\beta)$.

Guarantees: If $t < n/3$, then servers don't learn anything from $p(X)$ apart from $p(\beta)$

For each MPC server P_i:

- Obtain a sharing of the polynomial $[\![p(X)]\!]$ by locally interpolating at shares of evaluations $[\![x_{i_1}]\!], \ldots, [\![x_{i_k}]\!]$.
- Evaluate $[\![p(\beta)]\!]$ and send it all other parties..
- Use regular Lagrange interpolation to obtain the value $p(\beta)$.

Figure 6.7: MPC secret shared polynomials evaluate

Divmod operations on secret shared polynomials.

Input: Secret shared polynomials: Secret shares of evaluations of a polynomial $a(X)$ as $([\![x_{i_1}]\!], \ldots, [\![x_{i_k}]\!])$, and another publicly known polynomial $b(X)$.

Output: Secret shared evaluations of resultant polynomial $r(X)$ after the corresponding arithmetic operation $[\![q(X)]\!], [\![r(X)]\!] = [\![a(X)]\!]/b(X)$ such that $q(X)b(X) + r(X) = a(X)$ and degree$(r(X))$ ¡ degree$(b(X))$.

Guarantees: If $t < n/3$, then servers don't learn anything from $a(X), b(X), r(X)$

For each MPC server P_i:

- Obtain a sharing of the polynomial $[\![a(X)]\!]$ by locally interpolating at shares of evaluations $[\![x_{i_1}]\!], \ldots, [\![x_{i_k}]\!]$.
- Use FFT based divmod on local shares of polynomials $[\![a(X)]\!]$ to compute the shares of $[\![q(X)]\!], [\![r(X)]\!]$. Appendix A shows the details of the polynomial division operation that can be our in $n \log n$

Figure 6.8: Secret shared polynomial division with remainder by a publicly known polynomial

CHAPTER 7: ANALYSIS AND SECURITY PROOFS

7.1 DETAILED PROTOCOL ANALYSIS

		size/cost(bytes)		Time Complexity		
		Comm	$\|\pi\|$	Prover	Auditor	Setup
Vee'17	$\mathbb{G}_1$	-	$3\|X\| + 8$	$\mathcal{O}(n)$	$6\|X\| + 12$ pair.	circ
	$\mathbb{G}_2$	-	-	-	-	
	$\mathbb{F}_q$	-	-	$\mathcal{O}(m + n \log n)$	-	
This work	$\mathbb{G}_1$	$N(N-1)(\|X\|+12)$	$13 + \|X\|$	21 v-MSM($3m$) + 7 v-MSM(N)	1 v-MSM($\|X\|$) + 2	univ
	$\mathbb{G}_2$	-	-	-	3(2*) pairings	
	$\mathbb{F}_q$	$14N(N-1)$	23	$\mathcal{O}(N + m \log m)$	$\mathcal{O}(\log m)$	

Table 7.1: M, N, m, n represents the maximum size of circuit, number of MPC parties, total number of gates and number of multiplication gates respectively. Comm, π represent the total communication cost across all MPC servers and proof size.v-MSM(m) denote variable-base multi-scalar multiplications (MSM) each of size m. It is possible to reduce the pairings to 2 pairings, however our implementation currently does not support it.

Next, we show a detailed analysis of our protocol in terms of the exact number of group elements in ($\mathbb{G}_1$ and $\mathbb{G}_2$) and field elements($\mathbb{F}_q$). Note that Vee'17 does not report a detailed breakdown of proof elements, so we use the $\mathcal{O}()$ asymptotic, as shown in their paper. We note that despite our setup being universal, our proof size and auditor size are smaller than those of Vee'17 construction.

The reason for our efficiency is because of different approaches to construction where the Vee'17 construction incurs penalty because of having to check pairings for each statement element. Although Vee'17 prover does not show the exact prover computation numbers, we believe that the prover they have will be faster than ours because of access to circuit-specific encoding in the exponent. This is to be expected as Marlin Prover [2] is about 3-4 times slower than Groth16 [45], which is the state of the art circuit-specific snark. We believe that if one can apply our ideas to Groth'16 SNARK, it possible to overcome these limitations. We leave exploring that as a future work as we were interested in diverse applicability Universal Setup.

Table 7.1 shows the detailed comparison of our work with the previous work by Veeningen [1]. Adaptive Trinochio construction in Vee'17 had an auditor do pairings of the order of statement size $|X|$ whereas our construction only uses three pairings. Note that it is possible to use an optimization detailed in Section 9 of Marlin [2] to reduce this cost to 2 pairings.

However, our implementation does not currently support this, and hence we report our cost as three pairings instead of two.

-	Message Type	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{F}_q$
Initial Round	commitments	$\|X\|(N-1)$	-	-
First Round	commitments	$4(N-1)$	-	-
Second Round	commitments	$4(N-1)$	-	-
	PolyEval proofs	$6(N-1)$	-	$6(N-1)$
	evaluations	-	-	$8(N-1)$

Table 7.2: Communication Cost breakdown per server per round. N represents the number of MPC servers and $|X|$ is the statement size.

Table 7.2 shows the detailed round wise communication cost analysis for each round. In the initial round, the servers exchange the commitments to shares of statements; hence the per-server cost is $|X|$. In the first round, the servers commit to the secret shared polynomials $w, mask, z_a, z_b$. The second round of communication consists of 3 separate messages sent together: 1) Commitments to g_1 and h_1, 2) evaluation proofs for $g_1, h_1, w, mask, z_a, z_b$ at β_1 and 3) evaluations of x and r polynomials.

7.2 SECURITY PROOFS

In this section, we will show that completeness, extraction, and zero-knowledge properties of our adaptive zk-snarks described in section 6.1. We show the following proofs assuming an Extractable PEC scheme.

- Perfect Completeness:

  Proof: By inspection

- Extractable: We say that our Snark 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

$$\Pr\left[\begin{array}{c|c} & \mathsf{srs} \leftarrow \mathsf{G}(1^\lambda, \mathsf{N}) \\ (\mathbf{C_x}, \mathbb{i}, \mathbb{x}, \mathbb{r}, \mathbb{w}) \notin \mathcal{R}_{\mathsf{ck}_{eval}} & \mathsf{ck_{eval}} \leftarrow \mathsf{PEC.Setup} \\ \wedge & (\mathbf{C_x}, \mathbb{i}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{srs}; \mathsf{z}) \\ \langle \mathsf{A}_2(\mathsf{st}), \mathsf{V}(\mathsf{ivk}, \mathbf{C_x}) \rangle = 1 & (\mathbb{x}, \mathbb{r}, \mathbb{w}) \leftarrow \mathcal{E}^{A_1}(\mathsf{srs}; \mathsf{z}) \\ & \mathsf{ipk}, \mathsf{ivk} \leftarrow \mathsf{I}^{\mathsf{srs}}(\mathbb{i}) \end{array}\right] = \mathsf{negl}(\lambda) \tag{7.1}$$

The output proof from $\mathcal{A}_2$ would be of the form $(\mathsf{prf}_{\mathsf{mar}}, \mathsf{v}, \pi_{\mathsf{comm}}, \mathbf{C}^{\mathbb{b}}_{\mathbf{x}})$. 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 $\mathbf{C}_{\mathbf{x}}, \mathbf{C}^{\mathbb{b}}_{\mathbf{x}}$.
- Use the marlin extractor $\mathcal{E}_{Mar}$ with above extracted statement $\mathbb{x}$ and $\mathsf{prf}_{\mathsf{mar}}$ to get witness $\mathbb{w}$. The constructions of this extractor is described in Section 8.3 of Marlin [2] 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 ϵ, 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 [2] 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 $\mathcal{R}$. Recall that:

$$\mathcal{R}_{\mathsf{ck}_{eval}} := \{(\mathbf{C}_{\mathbf{x}}, \mathbb{i}, \mathbb{x}, \mathbf{r}, \mathbb{w})) : \forall i \; C_{x_i} = \mathsf{PEC.Commit}_{\mathsf{eval}}(\mathsf{ck}_{\mathsf{eval}_i}, x_i, i, r_i) \wedge (\mathbb{x}, \mathbb{i}, \mathbb{w}) \in \mathcal{R}\} \quad (7.2)$$

By properties of the extractor $\mathcal{E}_{comm}$, we know that openings to the commitments $\mathbf{C}_{\mathbf{x}}$ 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 v is computed the verifier himself based on the statement x. We need show that with high probability that the 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 completeness of underlying AHP Marlin protocol, we know that

$$\hat{z}(X) := \hat{w}(X) v_H(X) + \hat{x}(X) \quad (7.3)$$

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 β_1 from the verifier. $|X|$ is the statement length and also the degree of the polynomial $\hat{x}$.

- Zero Knowledge: We say that our Snark is zero knowledge if there exists a simulator $\mathsf{S} = (\mathsf{S.Setup}, \mathsf{S.Prove})$ if for every efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ it holds that

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

$$\Pr\left[\begin{array}{c} (\mathbf{C_x}, \mathbb{i}, \mathbb{x}, \mathbb{r}, \mathbb{w}) \in \mathcal{R}_{\mathsf{ck}_{eval}} \\ \wedge \\ \langle \mathsf{S.Prove}(\mathsf{td_{srs}}, \mathsf{td_{comm}}, \mathbf{C_x}, \mathbb{i}), \mathcal{A}_2(\mathsf{st}) \rangle = 1 \end{array} \middle| \begin{array}{c} \mathsf{srs}, \mathsf{td_{comm}}, \mathsf{td_{srs}} \leftarrow \mathsf{S.Setup}(\mathsf{N}) \\ (\mathbf{C_x}, \mathbb{i}, \mathbb{x}, \mathbb{r}, \mathbb{w}), \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{srs}) \end{array} \right] \quad (7.4)$$

We need to the simulator with access to the trapdoor can generate proofs that can indistinguishable from real proofs. In other words, our simulator 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:

Simulator for Adaptive ZK Construction in 6.1

- Recall that our proof consists of four elements $(\mathsf{prf_{mar}}, \mathsf{v}, \pi_{\mathsf{comm}}, \mathbf{C}_{\mathbf{x}}^{\mathbb{b}})$. First, the simulator samples a random polynomial $\hat{x}$ of $\mathsf{deg}(|\mathsf{C_x}|) + \mathsf{deg}(|C_x^{\mathbb{b}}|)$. 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 $\mathsf{prf'_{mar}}$ and obtain the evaluation $\hat{x}(\beta_1)$.
- Interpolate the commitments $\mathbf{C_x}$ using $\mathsf{PEC.interpolate}$ to obtain a polynomial commitment c_p for the $\hat{x}$ and provide a evaluation proof π'_{comm} for the correct value $\hat{x}(\beta_1)$.
- return the proof $(\mathsf{prf'_{mar}}, \hat{x}(\beta_1)\ \pi'_{\mathsf{comm}}, (|C_x'^{\mathbb{b}}|))$.

Figure 7.1: Simulator construction for adaptive zk-snark

The figure 7.1 shows the construction for simulator satisfying the above definition for our construction in 6.1. 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 $\mathsf{deg}(\hat{\mathsf{x}}) = \mathsf{deg}(\mathsf{x}) + \mathbb{b}$ queries less than the query bound $\mathbb{b}$ information theoretically does not reveal any information about x.

Similar to the construction in [51], we would construct a simulator 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 [2]. At the high level, all the subsequent messages are hidden by adding the additional s polynomial and hence do not reveal any information.

From the marlin simulator with the trapdoors to srs which uses AHP' instead of AHP described in marlin, we know that $\mathsf{prf}'_{\mathsf{mar}}$ and $\mathsf{prf}_{\mathsf{mar}}$ 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 $\mathsf{td}_{\mathsf{comm}}$ for PEC, we can simulated proofs for $(\pi'_{comm}, \hat{x}(\beta_1))$ are indistinguishable from $(\pi_{\mathsf{comm}}, \mathsf{v})$.

Although, we have argued about the individual distributions of $\mathsf{prf}'_{\mathsf{mar}}$ and $\mathsf{prf}_{\mathsf{mar}}$ are same and distributions for $(\pi_{\mathsf{comm}}, \mathsf{v},$, 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 PEC scheme ensures that the simulator by using the trapdoor $\mathsf{trap}_{\mathsf{comm}}$ 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.

CHAPTER 8: EVALUATION

In this section, we describe our implementation and evaluation for auditable MPC protocol. We build auditable MPC as a rust application built on top of Marlin codebase [4]. Our code is available at https://github.com/sanket1729/auditable_mpc as a rust application. We strive to make our benchmarks completely reproducible by dockerizing our setup and one-line command to generate all the graphs in this section. The reader can refer to the README for details.

We simulate the MPC behavior by using artificial delays in communication latency of 200ms and an uplink speed of 200Mbits/per second. We use these as worst-case communication latency and uplink speed as reported in [7] Appendix C. We report our performance numbers on a single thread machine with a modern CPU processor with 1200 MHz. We use BLS381 curve [67] for pairing friendly and fft optimizations. For the fast computation of Lagrange interpolations and Lagrange interpolations in the exponent, we use preprocessed VanderMonde matrices. We use Pedersen commitment as a choice a commitment for fast prototyping of our construction. We will be conducting more benchmarks on the other commitment scheme as a continued work.

Unlike regular SNARKS, where the proving/auditing and proof size are majorly defined by the number of constraints, our system, because of MPC nature, is affected by other factors detailed below. In the subsequent subsections, we show how the prover time, auditor time, proof size, and communication cost are affected statement size, the number of constraints, and the number of MPC servers involved. We vary our statement size from $[2^2, 2^6]$, our witness size from $[2^{10}, 2^{20}]$ and number of MPC servers from $[2^2, 2^6]$. Our experiment parameter ranges cover a wide range of circuits of interest, SHA256 circuit, for example, has statement size 1, witness size $\approx 2^{14}$. Practical MPC implementations range from 3 servers to 50 servers [7]. For the generation of random circuits, we sample random R1CS instances with only multiplication constraints as the addition in circuits is free in R1CS.

8.1 PROVER COST

As mentioned in the previous section, each server computation consists of three main components: 1) Computing the underlying MPC circuit, 2) calculating the witness and output wire values at each server, and 3) computing the Marlin proof by doing another MPC amongst the servers. When we report prover time benchmarks, we only consider the time for the third component in the proof. As our prover works in a round-wise synchronous

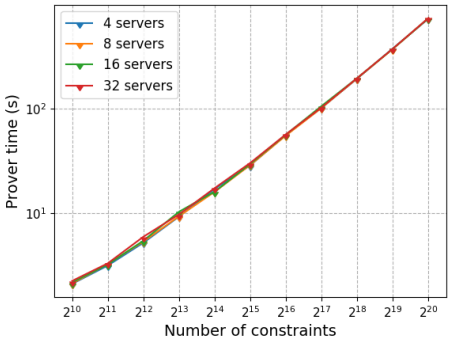

Figure 8.1: Prover cost as a function of number of constraints. The different lines indicate number of MPC servers

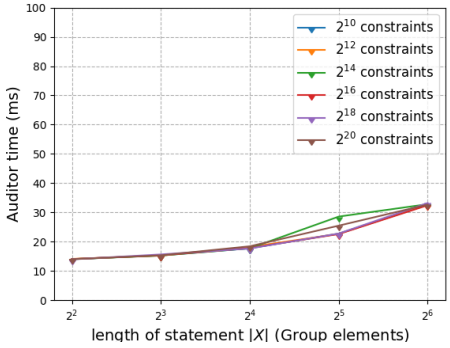

Figure 8.2: Auditor cost as a function of statement length. The different lines indicate the number of constraints

fashion, we consider a round-time for MPC as the worst time amongst all provers in that round. As shown in 8.1 prover cost increases almost linearly with the number of constraints. The different lines show the prover time with different number of servers. As expected, the prover time overhead due to the extra communication cost is marginal compared to the Marlin prover cost, and hence all the lines are almost overlapping. The effect of statement size on prover time is insignificant because it is overshadowed by the polynomial commitment operations on witness elements. The reason for that prover only has made a single group exponentiation per statement value and that the statement length is typically far less than the number of constraints.

8.2 AUDITOR COST

The auditor computation mainly consists of two 1) Verifying the marlin proof and 2) Carrying out the input consistency check. The first computation involves pairings to verify that the claimed evaluations are consistent with the commitments, while the second operation involves interpolating a polynomial in the exponent for the input consistency check. The auditor cost is typically less than 30 ms and hence the exact measurement is susceptible to CPU scheduling errors and other processes running along with it. Therefore, we report the timings as the average over 100 auditor verifications for the same proof. Recall that our auditor is oblivious to the number of servers used in the computation; therefore, the auditor time does not depend on the number of MPC servers used. The first component of auditor cost stays constant with the increasing number of constraints and statement size, while the second component of auditor time increases linearly with statement size. Figure 8.2 shows the auditor cost as a function of statement length. As expected, with the increase state-

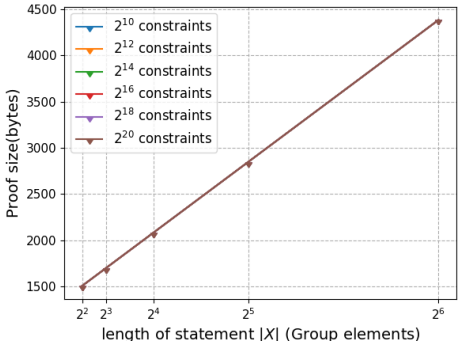

Figure 8.3: Proof size as a function of output statement size. Multiple lines show the number of constraints

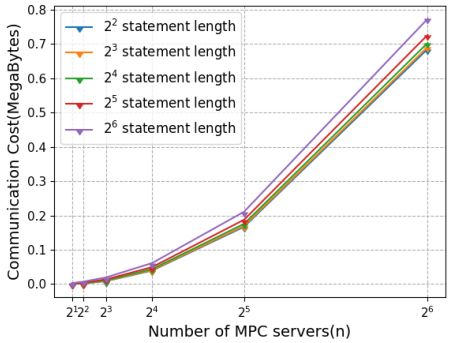

Figure 8.4: Communication Cost as a function of number of MPC servers

ment size, the auditor cost increases. We note that after statement size ≈ 64, the second component starts dominating the first component. [1]

8.3 PROOF SIZE

Our auditable MPC proof is just a Marlin proof combined with statement commitments. Typically in SNARKS, the statement is not considered a part of proof because the verifier is assumed to have access to it. While it is reasonable to assume that the auditor already has client input commitments, but the output commitments are computed by the servers at proving time. Therefore, we explicitly model output statement commitments in proof size. In particular, the commitment to the output to servers should also be a part of the proof. As expected, the marlin proof is constant is size while our input checking component adds a linear overhead in statement size. As shown in 8.3, proof size is a straight line against the output statement size.

8.4 COMMUNICATION COST

Figure 8.4 shows the additional communication overhead incurred in proving due to the three extra communication rounds we introduced. Our first round involves broadcasting the commitments to secret shared statement values. In contrast, the second and third rounds involve broadcasting commitments to secret polynomials and evaluations of secret shared polynomials. Note that our communication cost does not depend on the number

[1]The x-label on the graph shows statement size of 2^k, it is 2^k - 2 where one statement is value 1, and one auxiliary value for hiding the commitment

of constraints in the circuit, so we only show the total communication cost and statement size in Fig 8.4. As expected, our communication cost is linear per server, and the total communication cost is quadratic in the number of servers.

CHAPTER 9: FINAL REMARKS

9.1 CONCLUSION

Most deployed MPC solutions do not provide *unconditional* correctness in the case that all servers are corrupted. In this work, we extend the current construction of auditable MPC to reflect the

In this work, we show the *first* construction of auditable robust MPC, which supports fast verification, succinct proof size with one time universal, and updatable setup. Importantly, we provide auditability without significantly compromising the performance of the underlying Shamir secret shared MPC protocol, i.e., adding auditability only incurs a linear computation overhead and constant round communication overhead. We implement and evaluate our construction report various performance metrics.

9.2 FUTURE WORK

- The current scheme for auditable MPC uses an auditor that reads the entire statement and then grows linearly in the size of the statement. It is possible to use Zk-rollup techniques to reduce the size of these proofs.
- A Universal Composable(UC) model the auditable MPC functionality. The current version is compatible with UC functionality, but reasoning about the proofs would require a formal UC model to capture the auction application.
- It would be interesting to explore whether other snarks based on constructions of snarks based on bulletproofs, Groth'16, and others can be modeled for auditable MPC.

APPENDIX A: FFT POLYNOMIAL OPERATIONS

This appendix lists the FFT operations used by MPC servers for addition, subtraction, multiplication and division. This code is are based from class notes from CS577 class 2003 [68] from Iowa state university and has been taken from this stackoverflow answer [69]. This is not our contribution, we merely state this for completeness and self-containment of this work.

```
def poly_divmod(u, v):
    """Fast polynomial division ``u(x)`` / ``v(x)`` of polynomials
    with degrees m and n. """
    if not u or not v:
        return []
    m = poly_deg(u)
    n = poly_deg(v)

    # ensure deg(v) is one less than some power of 2
    # by extending v -> ve, u -> ue (mult by x^nd)
    nd = int(2**ceil(log(n+1, 2))) - 1 - n
    ue = poly_scale(u, nd)
    ve = poly_scale(v, nd)
    me = m + nd
    ne = n + nd

    s = poly_recip(ve)
    q = poly_scale(poly_mul(ue, s), -2*ne)

    # handle the case when m>2n
    if me > 2*ne:
        # t = x^2n - s*v
        t = poly_sub(poly_scale([1], 2*ne), poly_mul(s, ve))
        q2, r2 = poly_divmod(poly_scale(poly_mul(ue, t), -2*ne), ve)
        q = poly_add(q, q2)

    # remainder, r = u - v*q
    r = poly_sub(u, poly_mul(v, q))

    return q, r
```

REFERENCES

[1] 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.

[2] 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.

[3] "Zexe (zero knowledge execution) rust library," 2019. [Online]. Available: https://github.com/scipr-lab/zexe

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

[5] 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.

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

[7] 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.

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

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] M. Keller, E. Orsini, and P. Scholl, "Mascot: faster malicious arithmetic secure computation with oblivious transfer," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 830–842.

[16] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: making spdz great again," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 158–189.

[17] 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.

[18] A. Aly, M. Keller, E. Orsini, D. Rotaru, P. Scholl, N. P. Smart, and T. Wood, "Scale–mamba v1. 3: Documentation," Technical Report, Tech. Rep., 2019.

[19] 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.

[20] B. Schoenmakers, "A simple publicly verifiable secret sharing scheme and its application to electronic voting," in *Annual International Cryptology Conference*. Springer, 1999, pp. 148–164.

[21] R. Küsters, T. Truderung, and A. Vogt, "Accountability: definition and relationship to verifiability," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 526–535.

[22] K. Sako and J. Kilian, "Receipt-free mix-type voting scheme," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1995, pp. 393–403.

[23] T. Moran and M. Naor, "Receipt-free universally-verifiable voting with everlasting privacy," in *Annual International Cryptology Conference*. Springer, 2006, pp. 373–392.

[24] M. Chen, U. Northeastern, C. Hazay, U. Bar-Ilan, Y. Ishai, Y. Kashnikov, D. Micciancio, T. Riviere, M. Venkitasubramaniam, and R. Wang, "Diogenes: Lightweight scalable rsa modulus generation with a dishonest majority."

[25] 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.

[26] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn, "Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2111–2128.

[27] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 315–334.

[28] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish, "Doubly-efficient zksnarks without trusted setup," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 926–943.

[29] J. Zhang, T. Xie, Y. Zhang, and D. Song, "Transparent polynomial delegation and its applications to zero knowledge proof."

[30] B. Bünz, B. Fisch, and A. Szepieniec, "Transparent snarks from dark compilers," Cryptology ePrint Archive, Report 2019/1229, 2019, https://eprint. iacr. org . . . , Tech. Rep., 2019.

[31] D. Fiore, C. Fournet, E. Ghosh, M. Kohlweiss, O. Ohrimenko, and B. Parno, "Hash first, argue later: Adaptive verifiable computations on outsourced data," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1304–1316.

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

[33] B. Schoenmakers, M. Veeningen, and N. de Vreede, "Trinocchio: privacy-preserving outsourcing by distributed verifiable computation," in *International Conference on Applied Cryptography and Network Security*. Springer, 2016, pp. 346–366.

[34] 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.

[35] 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.

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

[37] P. Mohassel, M. Rosulek, and Y. Zhang, "Fast and secure three-party computation: The garbled circuit approach," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 591–602.

[38] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 35–52.

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

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

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

[42] 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.

[43] J. Groth, "Short pairing-based non-interactive zero-knowledge arguments," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2010, pp. 321–340.

[44] J. Groth and A. Sahai, "Efficient non-interactive proof systems for bilinear groups," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2008, pp. 415–432.

[45] J. Groth, "On the size of pairing-based non-interactive arguments," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2016, pp. 305–326.

[46] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 781–796.

[47] E. Ben-Sasson, A. Chiesa, A. Gabizon, and M. Virza, "Quasi-linear size zero knowledge from linear-algebraic pcps," in *Theory of Cryptography Conference*. Springer, 2016, pp. 33–64.

[48] E. Ben-Sasson, A. Chiesa, A. Gabizon, M. Riabzev, and N. Spooner, "Interactive oracle proofs with constant rate and query complexity," in *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[49] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Scalable zero knowledge via cycles of elliptic curves," *Algorithmica*, vol. 79, no. 4, pp. 1102–1160, 2017.

[50] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable zero knowledge with no trusted setup," in *Annual International Cryptology Conference*. Springer, 2019, pp. 701–732.

[51] 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.

[52] I. Giacomelli, J. Madsen, and C. Orlandi, "Zkboo: Faster zero-knowledge for boolean circuits," in *25th {usenix} security symposium ({usenix} security 16)*, 2016, pp. 1069–1083.

[53] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha, "Post-quantum zero-knowledge and signatures from symmetric-key primitives," in *Proceedings of the 2017 acm sigsac conference on computer and communications security*, 2017, pp. 1825–1842.

[54] S. Setty, "Spartan: Efficient and general-purpose zksnarks without trusted setup," Cryptology ePrint Archive, Report 2019/550, Tech. Rep., 2019.

[55] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," Cryptology ePrint Archive, Report 2019/953, Tech. Rep., 2019.

[56] J. D. Cohen and M. J. Fischer, *A robust and verifiable cryptographically secure election scheme*. Yale University. Department of Computer Science, 1985.

[57] B. Adida, "Helios: Web-based open-audit voting." in *USENIX security symposium*, vol. 17, 2008, pp. 335–348.

[58] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," in *Proceedings of the 1st ACM conference on Electronic commerce*, 1999, pp. 129–139.

[59] K. Sako, "An auction protocol which hides bids of losers," in *International Workshop on Public Key Cryptography*. Springer, 2000, pp. 422–432.

[60] E. Fujisaki and T. Okamoto, "A practical and provably secure scheme for publicly verifiable secret sharing and its applications," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1998, pp. 32–46.

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

[62] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Annual Cryptology Conference*. Springer, 2012, pp. 643–662.

[63] 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.

[64] 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.

[65] 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.

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

[67] S. Bowe, "Faster subgroup checks for bls12-381."

[68] 2003. [Online]. Available: http://web.cs.iastate.edu/~cs577/handouts/polydivide.pdf

[69] 2003. [Online]. Available: https://stackoverflow.com/questions/44770632/fft-division-for-fast-polynomial-division