

(Im)possibility of Asynchronous MPC with Honest Majority over Blockchains

Ashish Choudhury ¹, Sannidhi V Hebbar ², Aniket Kate ³, Pabitra Mandal ², and Arpita Patra ²

¹ IIT Bangalore, Email: ashish.choudhury@iiitb.ac.in

² Indian Institute of Science, Emails: {sannidhiv,pabitr,arpita}@iisc.ac.in

³ Mysten Labs / Purdue University, Email: aniket@purdue.edu

Abstract. This work studies asynchronous verifiable secret sharing (AVSS) and asynchronous multi-party computation (AMPC) in the blockchain-hybrid model, where parties have black-box access to an ideal (asynchronous) blockchain functionality providing only persistence and eventual liveness. Motivated by the practical deployment of MPC in blockchain applications such as privacy-preserving payments and threshold wallets, we investigate whether blockchain access can improve the classical resilience bound of $n > 3t$, where n is the total number of parties, and t is the number of parties that can be compromised by an adversary. In particular, in the blockchain-hybrid model, we provide a comprehensive set of lower and upper bounds across three cryptographic settings: (i) no trusted setup, (ii) trusted setup with Minicrypt assumptions, (iii) trusted setup with public-key assumptions.

1. We show that without a trusted setup, or under Minicrypt assumptions, even with a setup, the classical resilience bound for AMPC is inherent: AMPC is impossible for $n \leq 3t$, even against weaker fail-stop or omission adversaries.
2. We establish separations between AVSS and AMPC in the intermediate regime $2t < n \leq 3t$: against a fail-stop adversary, unlike AMPC, AVSS is possible for $n > 2t$ without any setup. Moreover, against a Byzantine adversary, again unlike AMPC, AVSS is possible for $n > 2t$ under Minicrypt assumptions with a setup.
3. In contrast, under public-key assumptions with trusted setup, we construct an AMPC protocol tolerating Byzantine adversaries whenever $n > 2t$. Our protocol leverages threshold homomorphic encryption, threshold signatures, commitments, and zero-knowledge proofs to minimize on-chain communication, achieving blockchain communication complexity independent of the circuit size. In the process, we define an efficient agreement on a common subset primitive for large messages in the blockchain-hybrid model, which can be of independent interest for secure distributed computing systems.

Table of Contents

1	Introduction	1
1.1	Our Contributions	2
1.2	Related Work	3
2	Technical Overview	4
2.1	AVSS against Omission Adversary with No Setup (Section 4.1)	4
2.2	Perfectly Secure AVSS against Fail-stop Adversary (Section 4.2)	4
2.3	Minicrypt Regime: Impossibility of MPC (Section 5.1)	5
2.4	AVSS in Minicrypt with Honest Majority (Section 5.2)	5
2.5	MPC in the Public-key Setting with Honest Majority (Section 6)	5
3	Preliminaries	8
3.1	Notation	8
3.2	The Security Model	8
3.3	The Ideal Blockchain Functionality	10
3.4	The Ideal Asynchronous Reliable Broadcast Functionality	11
3.5	Ideal Functionality for Asynchronous Verifiable Secret Sharing	12
4	Setting with no Setup Assumption	12
4.1	Impossibility of AVSS Against Omission Adversary	12
4.2	Perfectly Secure VSS Against fail-stop Adversary with $n > 2t$	15
5	Setting with Setup and Minicrypt Assumptions	17
5.1	Impossibility of MPC with Setup and Minicrypt Assumptions	17
5.2	AVSS with Honest Majority and Minicrypt in $\mathcal{F}_{\text{Acast}}$ -hybrid	18
6	Setting with Setup and Public-key Cryptography	23
6.1	Cryptographic Primitives Deployed	23
6.2	Encrypted-and-Committed Values	28
6.3	The Preprocessing Phase Protocol	29
6.3.1	Agreement on Common Subset (ACS)	29
6.3.2	Ideal Functionality for One-Time Setup.	31
6.3.3	Protocol for Securely Realizing $\mathcal{F}_{\text{Pre}}$	31
6.4	Asynchronous Protocol for Public Decryption with a Linear Overhead	37
6.5	Beaver's Multiplication Protocol	38
6.6	The Asynchronous MPC Protocol	39

1 Introduction

Multi-Party Computation (MPC) [12, 23, 44, 70, 82] allows a set of n parties to securely compute an agreed function on their private input. A secure MPC protocol ensures that the function output is available to receivers, but nothing more about the private input is revealed, even when an adversary compromises a subset of t parties. With applications ranging from privacy-preserving machine learning to anonymous broadcast, MPC has been studied for more than three decades, and a few practical implementations have emerged in the last decade.

Cryptographic literature has primarily focused on MPC in the (bounded) synchronous communication setting, which assumes that all participants have access to a synchronized global clock and all messages are delivered within a pre-determined timeout. These protocols may lose liveness and, at times, safety and privacy when those communication bounds are broken. Indeed, when MPC solutions are deployed in the real-world setting, such as over the Internet, it is more suitable to design MPC protocols to tolerate network asynchrony; i.e., the MPC protocols should withstand arbitrary message delays to maintain safety and privacy, and eventually terminate as messages reach their destinations. In the last ten years, several asynchronous MPC (AMPC) protocols have been proposed [1, 8, 24–26, 33, 46, 62, 68], and a few have been implemented [7, 61, 75].

Weakening the communication assumption, however, is not without a tradeoff: in the asynchronous setting, when an honest party does not receive an expected message, it cannot distinguish between an honest-but-slow sender (whose message got delayed due to network asynchrony) and a corrupt sender that did not send anything in the first place. As a result, asynchronous MPC protocols are typically less resilient than their bounded-synchronous MPC counterparts; e.g., against a Byzantine adversary, computationally secure synchronous MPC with guaranteed output delivery requires a strict honest majority ($n > 2t$), while AMPC requires a super honest majority ($n > 3t$).

In the literature, with oracle accesses to some functionalities, resilience of AMPC has been improved: Beerliová-Trubíniová, Hirt, and Nielsen [11] observed that AMPC with ($n > 2t$) is possible given access to an oracle that allows each party to broadcast one single message synchronously. Backes et al. [5] achieve the same using an (asynchronous) non-equivocation functionality, and Cohen [27] in the asynchronous Byzantine agreement (ABA) hybrid model. Our work considers the problem of AMPC and the relevant primitives of asynchronous verifiable secret sharing (AVSS) with oracle access to a blockchain, a widely available and easily accessible oracle in practice today.

With the growing adoption of blockchains, several on-(block)chain applications demand access to MPC. For example, on-chain smart contracts often require access to MPC for privacy-preserving payment [77, 78], structured and unstructured randomness [40, 60], and threshold signatures [50]. To support these applications, it is essential to extend the MPC primitive by defining *MPC for use over blockchains*, which mandates that the final MPC output generated through the MPC process be recorded on the blockchain ledger in a verifiable manner. Here, it is important for the MPC protocol not to impose any additional network assumptions on the (destination) blockchain that uses it. As most MPC-relevant blockchains [2, 74, 76] are non-synchronous, the MPC design must be asynchronous in terms of the network assumption.

A straightforward approach to solve this problem is to first perform a standalone state-of-the-art AMPC [7, 75] over a system with n parties and subsequently publish the MPC output (signed by $t + 1$ parties) on-chain. While being conceptually simple, this approach misses the opportunity to use the blockchain-based agreement itself in the MPC process. As a result, this incurs significant communication and computation overhead, and more importantly, restricts the protocol’s fault tolerance to $t < n/3$. In this work, we study the problem of performing AMPC (for use) over a blockchain using the blockchain in a systematic manner.

The cryptographic literature has considered a blockchain-hybrid model [6] for secure computation before [45, 47, 48, 71]. Motivated by the earlier Bitcoin analyses, the employed functionality here makes strong communication assumptions: the speed of the ledger can be neither too slow nor too fast. However, for modern (second and third generation) blockchains, in the presence of fairness attacks such as front-running [31] and censorship, network congestion crises [59], and liveness bugs [65], any bounds on the time it takes for honest users’ messages to appear on the blockchain cannot be guaranteed. As a result, as a preliminary step, we define an asynchronous blockchain functionality that only assumes persistence and eventual liveness. In particular, our functionality only offers two properties matching the state machine replication [63] from the literature: (a) liveness: every honest party’s message is eventually added to the

Table 1: **Our Results.** Lower-bound results indicate impossibilities, while the upper-bound results are backed by protocol constructions. To keep the results in the table succinct, we employ the following three strict hierarchies: (i) (A)VSS is a weaker primitive than (A)MPC, as VSS is a simple special case of MPC. Thus, a lower bound result on AVSS also applies to AMPC, and an upper bound result for AMPC also applies to AVSS, but not vice versa. (ii) Similarly, the Byzantine adversary is a weaker assumption than the omission adversary, which itself is a weaker assumption than the fail-stop adversary. (iii) Finally, (setup + Public key) is more demanding than (setup + Minicrypt), which is more demanding than (no trusted setup). In the table, the lower and upper bound results for the adversary setting and setup should be treated similarly to AVSS and AMPC.

Setting	Lower Bound	Upper Bound
No trusted setup	AVSS (omission): $n \leq 3t$	AVSS (fail-stop): $n > 2t$
Setup + Minicrypt	AMPC (fail-stop): $n \leq 3t$	AVSS (Byzantine): $n > 2t$
Setup + Public Key	AVSS (fail-stop): $n \leq 2t^*$	AMPC (Byzantine): $n > 2t$

* the lower bound of $n \leq 2t$ for (A)VSS is trivial even against the fail-stop adversary, as otherwise, sharing reconstruction cannot be guaranteed.

ledger, (b) persistence: different receivers can observe messages posted on the ledger at different points in time, but they all observe messages in the same total order.

This inherent asynchrony, motivated by the experiences in practice, sets our analysis apart from the state of the art. We perform an extensive analysis on lower-bound and upper-bound results and present novel results.

1.1 Our Contributions

We study *three* settings to determine if AVSS and AMPC can leverage the presence of an ideal (asynchronous) blockchain functionality (i.e., our blockchain-hybrid model) and improve the existing resiliency bound of $n > 3t$. The settings are: (a) computational security with *no* trusted setup, (b) computational security from Minicrypt [53] style assumptions and with an arbitrary trusted setup, and (c) computational security from public-key assumptions and with a trusted setup. We show that no trusted setup (with no restriction on cryptographic assumptions) and cryptography from Minicrypt with a trusted setup, leave very little to be done in the resiliency regime of $2t < n \leq 3t$, even for *weaker* form of corruptions such as omission [15] and fail-stop [15].⁴ However, in the last/third setting, we construct an AMPC protocol with $n > 2t$ in the blockchain-hybrid model. Specifically, we show the following results and summarize them in Table 1.

No Trusted Setup. Computationally-secure AVSS/AMPC tolerating a Byzantine adversary, even in the presence of a trusted setup (such as PKI), requires $n > 3t$. This resiliency bound stems from a much weaker primitive of asynchronous reliable broadcast (Acast) [14] for the same setting. In our blockchain-hybrid model, we ask if the resiliency bound for computational AVSS/AMPC *without* any setup assumption improves. Our results are primarily negative.

- On the *negative* side, we show that, even against a *weaker* omission adversary, the task of AVSS *cannot* be realized in blockchain-hybrid model with *computational* security if $n \leq 3t$. As Acast against the omission adversary is possible for $n > t$, our impossibility shows that in the *absence* of any setup, AVSS requires a strictly *higher* bound than Acast. The result also implies that AMPC is impossible as well in this setting with $n \leq 3t$.

The above results also imply that, without setup, computationally secure AVSS/AMPC is impossible for $n \leq 3t$ against the Byzantine adversary in the blockchain-hybrid model.

- On the *positive* side, we present an AVSS with *perfect* security against *fail-stop* adversaries with $n > 2t$. This interestingly *separates* out the possibility of AMPC and AVSS in the $2t < n \leq 3t$ regime against fail-stop adversaries.

⁴ Omission and fail-stop are both honest-but-curious adversaries, but an omission adversary can omit any message sent or received by a compromised party, while a fail-stop adversary can crash and stop the functioning of a compromised party at any time. Here, a crashed party remains inactive for the rest of the protocol execution.

Minicrypt with Setup Assumptions. Here, we consider computational security from Minicrypt assumptions *with* a trusted setup in the blockchain-hybrid model. Again, our results are primarily negative.

- On the *negative* side, we show that if $n \leq 3t$, then AMPC is *impossible* in this setting, even against the *weakest* fail-stop adversary. Thus, the classical $n \leq 3t$ resilience bound is *inherent* to AMPC and *cannot* be bypassed in the Minicrypt world, even with access to an ideal blockchain functionality.
- On the *positive* side, in the blockchain-hybrid model, we show that AVSS *can* be realized against a Byzantine adversary with $n > 2t$ with a setup along with Minicrypt. Moreover, as AVSS is not restricted by the famous FLP impossibility [37] on agreement, we only need a *weaker* Acast functionality, instead of the stronger blockchain functionality. This again demonstrates a sharp separation between AMPC and AVSS in this regime.

Public-key Assumptions with a Trusted Setup. Assuming access to an ideal blockchain functionality, we construct an AMPC protocol against Byzantine adversaries with $n > 2t$ using a threshold homomorphic public key encryption (THE), threshold signature (TSIG), homomorphic commitments, polynomial commitments, and zero-knowledge proofs. We employ all these primitives to reduce information posted on the blockchain (as it is typically an expensive resource). Our protocol assumes setup for the above primitives, and follows the standard preprocessing (offline-online) MPC paradigm. For securely computing a function with c_M multiplication gates and for a computational security parameter κ , the preprocessing phase incurs a communication of $\mathcal{O}(n^2 c_M \kappa)$ bits over the point-to-point channels and publishes $\mathcal{O}(n\kappa)$ bits over the blockchain. The online phase incurs a communication of $\mathcal{O}((nc_M + n^3)\kappa)$ bits over the point-to-point channels and publishes $\mathcal{O}(n\kappa)$ bits over the blockchain. Hence, the amount of communication over the blockchain is *independent* of c_M .

While building this AMPC, we define an efficient agreement on a common subset protocol for large messages in the blockchain-hybrid model, which can be of independent interest for secure distributed computing systems.

1.2 Related Work

Realizing Distributed Computing Protocols over Blockchains. A diverse set of asynchronous secure distributed computing protocols for use over a blockchain has been designed and put in production in the last five years. Here, we offer a summary of relevant results and their relation to our work.

The distributed storage problem involves storing an information blob in a fault-tolerant manner so that its retrievability is guaranteed. A trivial way to realize it on a blockchain is to post the entire blob on-chain, but on-chain storage may not scale with stored information, or at least is expensive. Walrus [80] (built over the Sui blockchain [76]) and Shelby [73] (built over the Aptos blockchain [2]) are real-world asynchronous distributed fault-tolerant storage solutions that realize blockchain-assisted distributed storage while ensuring the blockchain footprint is independent of the size of the stored information. At the conceptual level, the corresponding problem can be considered as asynchronous verifiable information dispersal (AVID) over a blockchain [64]. While built in the blockchain-hybrid model, both Walrus and Shelby employ $n > 3t$ similar to the stand-alone AVID primitive [16]. While those works do not prove the lower bound $n \leq 3t$, we believe our proof for the impossibility of AVSS against the omission adversary can be easily extended to the AVID problem in the blockchain-hybrid model.

Recently, Kate et al. [55] designed an asynchronous distributed key generation (ADKG) [58] protocol for threshold BLS signatures for users over a blockchain. Standalone ADKG requires $n > 3t$, while in the blockchain-hybrid model, their protocol employs a PKI setup and manages ADKG with a strict honest majority. However, they do not offer any proof for the necessity of PKI setup. Given the close relationship between VSS and DKG, our lower-bound analysis for AVSS should be applicable there as well. In fact, it can be interesting to build an ADKG protocol in the blockchain-hybrid model in our (Minicrypt + setup) setting.

Blockchain Oracles [20, 21] are also instances of a blockchain-hybrid distributed computing protocol, where the primitive realized over a blockchain is convex agreement [42] with strong validity. Convex agreement in the blockchain-hybrid model also needs $n > 3t$. While the literature does not offer formal proof, it should follow the standard asynchronous Byzantine agreement lower-bound proofs, where the agreement and strong validity properties contradict each other.

Existing Blockchain Functionalities. Towards building secure computations on top of blockchains, Bentov and Kumaresan [13] offered an early formalization of blockchain (the Bitcoin network in their case). However, the definition only works in synchronous rounds. Utilizing a Bitcoin formalization by Badertscher et al. [6], Rai Choudhuri, Goyal, and Jain [71] offered a detailed ideal functionality towards building cryptographic primitives, which has been employed in a series of work afterward [45, 47, 48]. While this functionality is not strictly bounded-synchronous, it expects that blocks are not generated too slowly or too fast. As we discussed earlier, this cannot be guaranteed for modern high-throughput and low-latency blockchain systems. David et al. [35] offered a simple ideal functionality for sharded blockchains in the partial synchronous model. We adopt their functionality to define our ideal blockchain functionality $\mathcal{F}_{\text{BC}}$. Recently, Damgård et al. [33] defined asynchronous total-order broadcast functionality that depicts an asynchronous blockchain. Our results should also hold given access to this functionality. Finally, Graf et al. [49] offer an elaborate functionality for smart-contract-enabled blockchains. We find this functionality to be too detailed for our purpose, as we do not employ smart-contract features of blockchains.

AMPC with a Strict Honest Majority. In any AMPC protocol, the parties need to agree on the “state” of the computation. Since ABA inherently requires $n > 3t$ (with or without cryptography) [36], it follows that the bound holds even for AMPC. A few alternatives have been considered to overcome this bound and get a protocol with $n > 2t$. In [11], Beerliová-Trubíniová et al. consider a “limited” usage of *synchronous* Byzantine agreement. In [5], Backes et al. employ a non-equivocation oracle, which prevents a corrupt party from sending “conflicting” messages to parties. In [27], Cohen assumes that the parties have access to an ideal ABA functionality; additionally, the protocol uses threshold fully-homomorphic encryption (TFHE), as the focus was to get a constant round MPC with complexity independent of the circuit size. As these approaches are different from ours, we cannot compare them. Nevertheless, as asynchronous blockchain can be realized using ABA, our analysis is applicable to [27] as well.

It should be noted that, in the spirit of [27], one can also obtain a conceptually simpler 2-round AMPC protocol in the blockchain hybrid model using TFHE and non-interactive zero-knowledge proofs (NIZKs). In such a solution, parties can encrypt their inputs and publish them on the blockchain together with a non-interactive proof of the correct formation of the encrypted inputs. Once a sufficient number of valid encrypted inputs appear in the blockchain, parties can locally evaluate the target circuit over the encrypted inputs and subsequently publish the corresponding partial decryption of the final output on the blockchain together with a non-interactive proof for correct decryption.

However, the objective of our work is not merely to establish the feasibility of AMPC with an honest majority in the blockchain-hybrid model. Rather, we aim to develop a more comprehensive understanding of the feasibility and infeasibility of asynchronous primitives in this model through a sequence of positive and negative results. Furthermore, our feasibility results are designed under comparatively minimal assumptions. In particular, our AMPC construction relies on THE, which is a significantly weaker assumption compared to TFHE.

2 Technical Overview

We overview our negative and positive results across various regimes.

2.1 AVSS against Omission Adversary with No Setup (Section 4.1)

To show that AVSS against omission adversaries is impossible in $\mathcal{F}_{\text{BC}}$ -hybrid model with $n \leq 3t$ in the absence of any setup, we show the impossibility first for 3 parties and one corruption and then extend it for any $n \leq 3t$ using the standard player partitioning approach. The impossibility for 3 parties is established through a contradiction. We assume there exists a protocol, and then, through a series of various possible executions, we show a breach of the privacy property of VSS.

2.2 Perfectly Secure AVSS against Fail-stop Adversary (Section 4.2)

We present a perfectly secure AVSS against fail-stop adversaries with $n > 2t$. One might use the standard Shamir’s secret-sharing [72] against a fail-stop adversary. However, a subtle issue is that a *corrupt* dealer may crash after giving shares *only* to a subset of honest parties. To address this, we introduce a simple

acknowledgment mechanism. Namely, *after* distributing shares to all parties, the dealer broadcasts an acknowledgment. Parties consider the sharing complete and output their shares, only *after* receiving this acknowledgment. The idea here is that if any *honest* party receives dealer’s acknowledgment, it knows that the dealer *did not* crash before distributing shares to all, and so the party can wait for its own share if it has not yet received it. To reconstruct the secret, the parties exchange their shares and wait for any $t + 1$ shares (allowed because $n - t \geq t + 1$). A similar protocol, but with $n > 3t$, is presented in [19].

2.3 Minicrypt Regime: Impossibility of MPC (Section 5.1)

We show the impossibility of general AMPC with $n \leq 3t$ and Minicrypt against fail-stop adversaries, even with a setup, as follows. From the seminal work of [53], it is well-known that the 2-party *oblivious transfer* (OT) functionality cannot be securely computed with Minicrypt in the *synchronous* communication setting, against *semi-honest* adversaries. Using the standard player partitioning argument, we extend this result and show that there exist n -party functionalities, which *cannot* be securely computed in the *synchronous* setting against *semi-honest* adversaries with $n \leq 2t$ and Minicrypt, even with access to an ideal blockchain functionality. Then, using the classical reduction of *asynchronous* MPC with $n \leq 3t$ against *fail-stop* adversaries to *synchronous* MPC with $n \leq 2t$ against *semi-honest* adversaries, due to Canetti [19], we arrive at our impossibility result.

2.4 AVSS in Minicrypt with Honest Majority (Section 5.2)

We present an AVSS in the Minicrypt regime against Byzantine adversaries with $n > 2t$, based on commitments and assuming a setup where for every pair of parties P_i, P_j , there exists a common key k_{ij} for a symmetric key encryption scheme.⁵ Moreover, we assume that the parties have access to an ideal asynchronous reliable broadcast functionality. For simplicity, let P_1 be the dealer. It computes shares of its secret s as per Shamir’s scheme. It then computes the commitment of each share s_i , and also an encryption of s_i , and the opening information used for computing the commitment, where the encryption happens under the key k_{1i} held between P_1 and P_i . The dealer then makes public all the encrypted and committed shares via a reliable broadcast. Moreover, it proves the “consistency” of the encrypted and committed shares using a generic zero-knowledge (ZK) proof. The ZK protocol can be constructed using the MPC-in-the-Head paradigm combined with the Fiat–Shamir transform in the random oracle model (ROM), following the techniques of [22]. Each party P_i , upon verifying the ZK proof, decrypts the i^{th} encrypted share and opening information using the key k_{1i} . The privacy of s is maintained, as the adversary learns up to t shares. On the other hand, even if the dealer is corrupt, if any honest party P_i computes its share, then every other honest party will be able to eventually compute its share, since the encrypted shares of *all* the parties are made public via reliable broadcast. Moreover, P_i would have verified the “consistency” of *all* encrypted and committed shares through the ZK proof, before computing its share.

The reconstruction of a shared secret is straightforward, where the parties exchange their respective shares and the opening information for the commitment of the shares. Each received share (and opening information) is then verified with respect to the commitments. Once a party has $t + 1$ verified shares (which it will eventually have since $n > 2t$), it interpolates them to reconstruct the secret.

2.5 MPC in the Public-key Setting with Honest Majority (Section 6)

Assuming an ideal blockchain functionality, we design an MPC protocol with $n = 2t + 1$ against Byzantine adversaries. Let $y = f(x_1, \dots, x_n)$ be a function over a finite field $\mathbb{F}$, expressed as an arithmetic circuit consisting of c_M multiplication gates. Our protocol is a variant of the AMPC protocol of [51] with $n > 3t$. We evaluate the entire circuit over what we call *encrypted-and-committed* values. An element $s \in \mathbb{F}$ is said to be encrypted-and-committed, if every party has a commitment $\mathbf{com}_s$ of s , as well as an encryption $\mathbf{c}_s$ of s and an encryption $\mathbf{c}_{r_s}$ of the randomness r_s (namely, the opening information) used to compute $\mathbf{com}_s$. We denote by $\langle s \rangle$ the triplet $(\mathbf{com}_s, \mathbf{c}_s, \mathbf{c}_{r_s})$.

Each party P_i , on having input x_i for f , computes $\langle x_i \rangle$, publishes it over the blockchain, and proves using zero-knowledge (ZK) the “correctness” of $\langle x_i \rangle$ to all. Due to asynchronous communication, the

⁵ Note that the commitment scheme *need not* be homomorphic.

parties can wait for $\langle x_i \rangle$ of only $n - t$ parties. To identify a *common* subset $\mathcal{CR}$ of $n - t$ such input providers, we depend on the “*persistence*” property of the blockchain.

Namely, each P_i , after proving the correctness of $\langle x_i \rangle$ to a verifier, collects a signature-share from the verifier and, upon having $t + 1$ signature-shares, computes a threshold-signature on $\langle x_i \rangle$. Then along with $\langle x_i \rangle$, party P_i also publishes a threshold-signature on $\langle x_i \rangle$ on the blockchain. To identify a common $\mathcal{CR}$, the parties simply consider the *first* $n - t$ parties P_i , who publish $\langle x_i \rangle$ along with a valid signature. Such a $\mathcal{CR}$ will eventually exist, as there are at least $n - t$ honest parties, who always constitute a candidate $\mathcal{CR}$. Moreover, $\mathcal{CR}$ will be common for all, due to the persistence property of the blockchain. Furthermore, the presence of a valid signature on $\langle x_i \rangle$ confirms that P_i has proved the correctness of $\langle x_i \rangle$ to at least one honest verifier. We stress that identification of a *common* $\mathcal{CR}$ is *not* possible, if the parties have access to a reliable broadcast functionality. This is because different honest parties may get output from the reliable broadcast instances of various sender parties in *different* order. And so the set of *first* $n - t$ sender parties who publish $\langle x_i \rangle$ along with a valid signature through reliable broadcast, might be *different* for different honest parties.

Once a common $\mathcal{CR}$ is identified, the parties substitute a default $\langle 0 \rangle$ as the input for the parties outside $\mathcal{CR}$ and start evaluating the circuit in a topological ordering, where for each gate, the parties maintain the following invariant: given the gate input(s) in an $\langle \cdot \rangle$ fashion, the parties securely compute the gate output in an $\langle \cdot \rangle$ fashion. Due to the linearity of THE and commitments, the variant can be maintained non-interactively for the linear gates. For multiplication gates, the parties use the Beaver’s method [9], by deploying random *multiplication-triples* in $\langle \cdot \rangle$ fashion, generated in the preprocessing phase (the details available in the sequel). Finally, once the parties have $\langle y \rangle$, the parties publicly decrypt y . Intuitively, the security holds as throughout the circuit evaluation, each value remains encrypted and committed, and only the function output is made public.

Need for Public Commitments as Part of $\langle \cdot \rangle$ Representation. As part of the Beaver’s multiplication protocol, parties need to do public decryption of random masked values, available in an encrypted-and-committed fashion. Let parties hold $\langle u^{(0)} \rangle, \dots, \langle u^{(t)} \rangle$ and the goal is to publicly reconstruct $u^{(0)}, \dots, u^{(t)}$. The standard approach would be to perform distributed decryption of $\mathbf{c}_{u^{(0)}}, \dots, \mathbf{c}_{u^{(t)}}$, available as part of $\langle u^{(0)} \rangle, \dots, \langle u^{(t)} \rangle$ respectively. Each distributed decryption would incur a communication of $\mathcal{O}(n^2\kappa)$ bits, as each party needs to send a decryption share to every other party (and prove its correctness), and so the total cost will be $\mathcal{O}(n^3\kappa)$ bits. Instead, we design a protocol Π_{PubDec} , which incurs a communication of $\mathcal{O}(n^2\kappa)$ bits and hence involves a linear “overhead”.

The protocol is based on an idea used in [10]. Let $f(X)$ be the t -degree polynomial with coefficients $u^{(0)}, \dots, u^{(t)}$ and let $v^{(i)} = f(i)$, where $v^{(i)}$ is a publicly-known linear function of $u^{(0)}, \dots, u^{(t)}$. Hence the parties can locally compute $\langle v^{(1)} \rangle, \dots, \langle v^{(n)} \rangle$ from $\langle u^{(0)} \rangle, \dots, \langle u^{(t)} \rangle$. Next, each $v^{(i)}$ is reconstructed *only* towards party P_i by using the naive reconstruction protocol, where the decryption shares of $v^{(i)}$ are sent only to P_i . Once P_i reconstructs $v^{(i)}$, it sends it to all, which can be verified against $\mathbf{com}_{v^{(i)}}$ available as part of $\langle v^{(i)} \rangle$. As there are at least $t + 1$ honest parties, every honest party will eventually get $t + 1$ correct points $v^{(i)}$, using which $f(X)$ and hence $u^{(0)}, \dots, u^{(t)}$ can be computed.

Note that protocol Π_{PubDec} will work only if commitments are part of $\langle \cdot \rangle$ representation, as otherwise parties *cannot* verify $v^{(i)}$ sent by a *corrupt* P_i . This justifies the need to have each value during the circuit evaluation to be *both* encrypted as per THE and committed. In the *absence* of commitments, we need to use the naive decryption protocol, which will increase the communication complexity of our online phase by a factor of $\mathcal{O}(n)$ per multiplication.

The Preprocessing Phase (Section 6.3.3). Here, the goal is to generate c_M number of triplets of the form $(\langle u \rangle, \langle v \rangle, \langle w \rangle)$, where $w = u \cdot v$ and where (u, v, w) remains random for the adversary. This is done in two stages, where the parties first jointly generate $\langle v \rangle$, followed by the generation of $\langle u \rangle$ and $\langle w \rangle$. To generate $\langle v \rangle$, each P_i picks a random $v^{(i)}$, computes $\langle v^{(i)} \rangle$, proves the correctness of $\langle v^{(i)} \rangle$ to all via ZK, collects signature-shares on $\langle v^{(i)} \rangle$ after proving the correctness and computes a threshold-signature on $\langle v^{(i)} \rangle$ after collecting $t + 1$ signatures-shares. It then publishes $\langle v^{(i)} \rangle$ along with a signature on the blockchain. Let $\mathcal{CR}_1$ be the set of *first* $n - t$ parties P_i , who publish a $\langle v^{(i)} \rangle$ along with a valid signature on the blockchain. Let $v \stackrel{\text{def}}{=} \sum_{P_i \in \mathcal{CR}_1} v^{(i)}$. The parties then locally compute $\langle v \rangle$ from $\{\langle v^{(i)} \rangle\}_{P_i \in \mathcal{CR}_1}$. As $\mathcal{CR}_1$ consists of at least one honest P_i with a random $v^{(i)}$, it follows that v remains random for the adversary.

The generation of $\langle u \rangle$ is similar to $\langle v \rangle$. Namely, each P_i picks a random $u^{(i)}$ and computes $\langle u^{(i)} \rangle$. Additionally, it computes $\langle w^{(i)} \rangle$ from $u^{(i)}$ and $\langle v \rangle$ by deploying the linearity properties of THE and commitments, where $w^{(i)} \stackrel{\text{def}}{=} u^{(i)} \cdot v$. Moreover, P_i rerandomizes $\langle w^{(i)} \rangle$. Then P_i proves the “correctness” of $\langle u^{(i)} \rangle$ and $\langle w^{(i)} \rangle$ to all via ZK and collects signature-shares from respective verifiers. Once P_i has $t + 1$ signature-shares, it computes a valid threshold-signature on $\langle u^{(i)} \rangle$ and $\langle w^{(i)} \rangle$ and publishes them over the blockchain. Let $\mathcal{CR}_2$ be the set of *first* $n - t$ parties P_i , who publish an $\langle u^{(i)} \rangle$ and $\langle w^{(i)} \rangle$, along with a valid signature on the blockchain. Let $u \stackrel{\text{def}}{=} \sum_{P_i \in \mathcal{CR}_1} u^{(i)}$ and $w \stackrel{\text{def}}{=} \sum_{P_i \in \mathcal{CR}_1} w^{(i)}$. The parties then locally compute $\langle u \rangle$ and $\langle w \rangle$ from $\{\langle u^{(i)} \rangle\}_{P_i \in \mathcal{CR}_2}$ and $\{\langle w^{(i)} \rangle\}_{P_i \in \mathcal{CR}_2}$ respectively. As $\mathcal{CR}_2$ consists of at least one honest P_i with a random $u^{(i)}$, it follows that u remains random for the adversary. Moreover, since P_i rerandomizes $\langle w^{(i)} \rangle$, no additional information about w is revealed.

Making Blockchain Communication Independent of the Circuit Size (Section 6.3.1). In the above preprocessing phase protocol, the amount of communication over the blockchain is *proportional* to c_M (the circuit size), as each P_i publishes $\mathcal{O}(c_M)$ number of $\langle u^{(i)} \rangle$, $\langle v^{(i)} \rangle$ and $\langle w^{(i)} \rangle$ over the blockchain. To make the communication over the blockchain *independent* of c_M , we design an independent building block, which we call *agreement on a common subset* (ACS). In the ACS protocol, each P_i has an input $m_i \in \{0, 1\}^L$, which it wants to send to all, where it is guaranteed that $V(m_i) = 1$ if P_i is honest. Here V is a publicly-known predicate, which is determined by the “outer protocol”, where Π_{ACS} will be used as a sub-protocol. For instance, V could be the verification protocol to verify whether the input provided by a party has a valid threshold-signature. Since in an asynchronous network, inputs of up to t parties could be delayed, the goal of Π_{ACS} is to identify a common subset of $n - t$ parties $\mathcal{CR}$, such that the input of every party in $\mathcal{CR}$ will be eventually available with all the parties and where $V(m_i) = 1$ for every $P_i \in \mathcal{CR}$.⁶

A trivial Π_{ACS} would be to ask each P_i to publish m_i over the blockchain and set $\mathcal{CR}$ to be the *first* $n - t$ parties, whose inputs are available over the blockchain and satisfy the predicate V (this is what parties do in the preprocessing outlined earlier). However, this requires communication over the blockchain, which is *proportional* to L . We instead give an instantiation, incurring a communication of $\mathcal{O}(n^2L)$ and $\mathcal{O}(n\kappa)$ bits over the point-to-point channels and blockchain, respectively. Provably, the communication over the point-to-point channels is *optimal*, since the input of every party in $\mathcal{CR}$ needs to be available with all [38]. Our instantiation of Π_{ACS} deploys a polynomial commitment scheme [54].

In our Π_{ACS} , each P_j distributes m_j to all. To prove that it has distributed m_j consistently, P_j publishes a polynomial commitment of m_j on the blockchain, whose size will be *independent* of L , by encoding m_j as a t -degree polynomial. Next, P_j publicly proves that it has distributed a common m_j to at least $n - t$ parties. For this, P_j asks for a signature-share from each P_i , who upon receiving m_j , issues a signature-share, provided $V(m_j) = 1$ and m_j is consistent with the corresponding polynomial commitment over the blockchain. Once P_j has $t + 1$ signature-shares, it computes a valid threshold-signature and publishes over the blockchain. The parties then set $\mathcal{CR}$ to be the *first* $n - t$ parties P_j , who publish a valid signature on the blockchain.

Once a $\mathcal{CR}$ is identified, the rest of the protocol tries to ensure that for every $P_j \in \mathcal{CR}$, all honest parties eventually receive the input m_j , which P_j would have publicly committed via polynomial commitment and where it is guaranteed that at least one honest party would have received m_j from P_j ; the latter holds since P_j would have received $t + 1$ signature-shares to compute a valid signature and one of these signature-shares must be from some honest party, who issues a signature-share only after receiving m_j and verifying its consistency with the corresponding polynomial commitment over the blockchain. Recall that P_j would have encoded m_j into a t -degree polynomial, say $f_{m_j}(X)$. So if all honest parties eventually get $t + 1$ verified points on $f_{m_j}(X)$, they can compute m_j . To execute this idea, we first enable each party P_k to “verifiably” obtain the evaluation point $f_{m_j}(k)$. This is done by asking each P_i who issued a signature-share to P_j , to send $f_{m_j}(k)$ to P_k , along with a *witness* of correct evaluation, computable via polynomial commitment. Once P_k gets $f_{m_j}(k)$, it relays it to all, along with a witness of correct evaluation. As there are at least $t + 1$ honest parties P_k , every honest party will eventually receive $t + 1$ evaluations of $f_{m_j}(X)$ along with matching valid witnesses, which it can use to compute m_j .

⁶ In the context of preprocessing phase discussed earlier, we need to run two instances of Π_{ACS} in sequence. In the first instance, the input m_i consists of the various $\langle v^{(i)} \rangle$ along with a threshold-signature, computed by P_i . In the second instance, m_i consists of the various $\langle u^{(i)} \rangle$ and $\langle w^{(i)} \rangle$ along with a threshold-signature, computed by P_i .

3 Preliminaries

We consider the standard asynchronous model of [19], where there exists a set of n mutually distrusting parties $\mathcal{P} = \{P_1, \dots, P_n\}$, connected by pairwise private and authenticated channels. Additionally, the parties have access to a public asynchronous blockchain, whose requirements are modeled through an ideal blockchain functionality (details provided in the sequel). The distrust among the parties is modeled by a centralized adversary that can corrupt any subset of up to t parties. The channels among the parties are asynchronous, where messages can be delayed arbitrarily, but finitely, with the guarantee that every sent message is delivered eventually. We consider the following different types of corruptions.

- **Fail-Stop [15]**: Here, the adversary will completely know the state of the corrupt parties during the execution of a protocol. The adversary can crash and stop the functioning of any corrupt party at any time during the execution, after which the crashed party remains inactive for the rest of the execution and does not send and receive any messages. However, as long as any corrupt party is functioning, it follows the prescribed protocol executions.
- **Omission [15]**: An omission adversary is similar to a fail-stop adversary, except that for every corrupt party, the adversary can decide to omit sending (or receiving) *any* subset of the messages the party is supposed to send (or receive) during the execution of a protocol. However, any message sent by the corrupt party will be computed as per the protocol instructions.
- **Byzantine**: Here, the adversary can dictate the corrupt parties to deviate from the prescribed protocol instructions in any arbitrary manner.

The above order is, in fact, as per the strength of corruption.

3.1 Notation

The communication complexity of any protocol is defined to be the total number of bits communicated by the *honest* parties, namely the parties not under the control of the adversary. The notation $\mathbb{B}(L)$ will denote the cost of communicating $\mathcal{O}(L)$ bits over the blockchain. On the other hand, the notation $\mathbb{P}(L)$ will denote that $\mathcal{O}(L)$ bits are communicated among the honest parties over point-to-point channels. In some of our protocols, we will perform computations over a finite field $\mathbb{F}$, where $|\mathbb{F}| > n$ and elements $1, \dots, n$ belong to $\mathbb{F}$.

3.2 The Security Model

We define the security of our protocols based on the standard *real-world/ideal-world* paradigm [17, 43], where the security of a protocol (for some computation) is argued by “comparing” the capabilities of the adversary in two separate worlds. In the real-world, the parties execute the protocol and exchange messages among themselves, computed as per the given protocol. In the ideal-world, the parties do not interact with each other, but rather with a *trusted third-party* (an ideal functionality), which assists the parties to get the result of the computation based on the inputs provided by the parties. Informally, a protocol is considered to be secure if whatever an adversary can do in the real protocol (where no trusted third party is present) can also be done in the ideal-world. The work of [28] has rigorously formalized the asynchronous network model with eventual message delivery in the UC framework, starting with the basic task of (asynchronous) point-to-point communication and formalizing asynchronous MPC. The same model was also followed in the work of [27], though the treatment was slightly *less* formal to avoid notational clutter and to make the protocols and proofs easier to understand. We now recall the high-level description of the framework followed in [27] and refer to [28, 56] for the complete formal details. The current description is recalled from [25].

The Real World: An execution of a protocol Π in the real-world (also called the real model), consists of n *interactive Turing machines* (ITMs) representing the parties $P_1, \dots, P_n$ respectively. Additionally, there is an ITM for representing the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ is *static* and decides the set of the corrupt parties $\mathcal{C}$ before the beginning of the protocol execution, where $|\mathcal{C}| \leq t$. The parties not under the control of $\mathcal{A}$ are called *honest*. The adversary controls the operations of the corrupted parties, as well as the delivery of the messages between the parties. The details follow.

Each ITM is initialized with random coins and their possible inputs. Additionally, $\mathcal{A}$ may have some auxiliary input z . The protocol operates *asynchronously* by a sequence of *activations*, where at each point a single ITM is active. Once activated, a party can perform some local computation, write on its output tape, or send messages to other parties. On the other hand, if the adversary is activated, it can send messages on behalf of the corrupted parties. To model the worst-case scenario, the adversary is given the provision to schedule the delivery of the messages exchanged between the parties. Once $\mathcal{A}$ delivers a message to some party, this party gets activated. The adversary cannot omit, change, or inject messages. However, the adversary can reorder the messages sent by the honest parties. That is, it can decide which message will be delivered and when. Moreover, even though the adversary can delay the delivery of the messages *arbitrarily*, it *cannot* delay them *indefinitely*. That is, every message sent by a party is *eventually* delivered. These requirements on adversarial scheduling are formalized using the *eventual-delivery secure message-transmission* ideal functionality in [28].⁷ The protocol execution is complete once all honest parties obtain their respective outputs. We let $\text{REAL}_{\Pi, \mathcal{A}(z), \mathcal{C}}(\vec{x})$ denote the random variable, consisting of the output of the honest parties and the view of the adversary $\mathcal{A}$, controlling the parties in $\mathcal{C}$ during the execution of a protocol Π , upon inputs $\vec{x} = (x^{(1)}, \dots, x^{(n)})$ for the parties (where party P_i has the input $x^{(i)}$) and auxiliary input z for $\mathcal{A}$.

The Ideal World: A computation in the ideal-world (also called the ideal model) consists of n *dummy* parties $P_1, \dots, P_n$, an ideal-world adversary $\mathcal{S}$ (also called as simulator) and an ideal functionality $\mathcal{F}$. We consider static corruptions, and so the set of corrupted parties $\mathcal{C}$ is fixed at the beginning of the computation and is known to $\mathcal{S}$. Moreover, following the notion of [3], we consider *corruption-aware* functionalities, where the identity of $\mathcal{C}$ is known to $\mathcal{F}$. The ideal functionality models the desired behavior of the computation. Namely, $\mathcal{F}$ receives the inputs from the respective dummy parties, performs the desired computation on the received inputs, and sends the outputs to the respective parties. The ideal-world adversary *does not* see and *cannot* delay the communication between the parties and $\mathcal{F}$, however it can communicate with $\mathcal{F}$ on the behalf of the parties in $\mathcal{C}$.

Since $\mathcal{F}$ models the desired behavior of a real-world protocol which is *asynchronous*, ideal functionalities must consider some inherent limitations. For example, in a real-world protocol, the adversary can decide when each honest party learns the output, since the adversary has full control over message scheduling. To model the notion of time in the ideal-world, [56] uses the concept of *number of activations*. Namely, once $\mathcal{F}$ has computed the output for some party, it does not ask “permission” from $\mathcal{S}$ to deliver it to the respective party. Instead, the corresponding party must “request” $\mathcal{F}$ for the output, which can be done only when the concerned party is active. Moreover, the adversary can “instruct” $\mathcal{F}$ to delay the output for each party by ignoring the corresponding requests, but only for a polynomial number of activations. If the party is activated “sufficiently many” times, the party will eventually receive the output from $\mathcal{F}$ and hence ideal computation eventually terminates.⁸ That is, each honest party eventually obtains its desired output. As in [27], we use the term $\mathcal{F}$ sends a request-based delayed output to P_i , to describe the above interaction between the $\mathcal{F}, \mathcal{S}$ and P_i .

Similar to the real-world, we let $\text{IDEAL}_{\mathcal{F}, \mathcal{S}(z), \mathcal{C}}(\vec{x})$ denote the random variable, consisting of the output of the honest parties and the view of the adversary $\mathcal{S}$, controlling the parties in $\mathcal{C}$, upon inputs $\vec{x} = (x^{(1)}, \dots, x^{(n)})$ for the parties (where party P_i has input $x^{(i)}$) and auxiliary input z for $\mathcal{S}$.

Definition 3.1. Let $\mathcal{F}$ be a functionality and let Π be an n -party protocol involving $\mathcal{P}$. We say that Π realizes $\mathcal{F}$ with computational security if and only if for every real-world PPT adversary $\mathcal{A}$, there exists a PPT ideal-world adversary $\mathcal{S}$, such that for every $\mathcal{C} \subset \mathcal{P}$ where $|\mathcal{C}| \leq t$, every $\vec{x} \in \{0, 1\}^*$ where $|x^{(1)}| = \dots = |x^{(n)}|$ and every $z \in \{0, 1\}^*$, the following two distributions are identical, except with a negligible probability in the security parameter κ , denoted as:

$$\left\{ \text{REAL}_{\Pi, \mathcal{A}(z), \mathcal{C}}(\vec{x}) \right\} \stackrel{c}{=} \left\{ \text{IDEAL}_{\mathcal{F}, \mathcal{S}(z), \mathcal{C}}(\vec{x}) \right\}.$$

⁷ In [28], while describing their protocols, the authors have used the ideal asynchronous message-transmission functionality to capture the point-to-point communication between the parties. However, this brings in a lot of additional technicalities and notations. We believe that doing the same will make our protocols harder to read and understand. Hence, as done in [27], we will avoid using the functionality for point-to-point communication. However, we confirm that this will not violate the overall UC-security of our protocols.

⁸ In [28], the order of activation is maintained and tracked in the protocols and proofs. We avoid doing so for our protocols, as it will make the proofs hard to read and comprehend. However, we confirm that this will not violate the correctness of our UC claims and their proofs.

Definition 3.2. Let $\mathcal{F}$ be a functionality and let Π be an n -party protocol involving $\mathcal{P}$. We say that Π realizes $\mathcal{F}$ with perfect security if and only if for every real-world unbounded adversary $\mathcal{A}$, there exists a PPT ideal-world adversary $\mathcal{S}$, such that for every $\mathcal{C} \subset \mathcal{P}$ where $|\mathcal{C}| \leq t$, every $\vec{x} \in (\{0,1\}^*)^n$ where $|x^{(1)}| = \dots = |x^{(n)}|$ and every $z \in \{0,1\}^*$, the following two distributions are identical, denoted as:

$$\left\{ \text{REAL}_{\Pi, \mathcal{A}(z), \mathcal{C}}(\vec{x}) \right\} \equiv \left\{ \text{IDEAL}_{\mathcal{F}, \mathcal{S}(z), \mathcal{C}}(\vec{x}) \right\}.$$

The Universal-Composability (UC) Framework: While the real-world/ ideal-world based security paradigm is used to define the security of a protocol in the “stand-alone” setting (namely, when only one instance of the protocol is running), the more powerful UC framework [17, 18] is used to define the security of a protocol when multiple instances of the protocol might be running in parallel, possibly along with other protocols. Informally, the security in the UC-framework is still argued by comparing the real-world and the ideal-world. However, now in both worlds, the computation takes place in the presence of an additional interactive process (modeled as an ITM), called the *environment* and denoted by $\mathcal{Z}$. Roughly speaking, $\mathcal{Z}$ models the “external environment” in which protocol execution takes place. The interaction between $\mathcal{Z}$ and the various entities takes place as follows in the two worlds.

In the real-world, the environment gives inputs to the honest parties, receives their outputs, and can communicate with the adversary at any point during the execution. During the protocol execution, the environment gets activated first. Once activated, the environment can either activate one of the parties by providing some input or activate $\mathcal{A}$ by sending it a message. Once a party completes its operations upon getting activated, the control is returned to the environment. Once $\mathcal{A}$ gets activated, it can communicate with the environment (apart from sending messages to the honest parties). The environment also fully controls the corrupt parties, that send to $\mathcal{Z}$ all the messages they receive, and follow the orders of $\mathcal{Z}$. The protocol execution is completed once $\mathcal{Z}$ stops activating other parties and outputs a single bit.

In the ideal-model, the environment $\mathcal{Z}$ gives inputs to the (dummy) honest parties, receives their outputs, and can communicate with $\mathcal{S}$ at any point during the execution. The dummy parties act as channels between $\mathcal{Z}$ and $\mathcal{F}$. That is, they send the inputs received from $\mathcal{Z}$ to $\mathcal{F}$ and transfer the output they receive from $\mathcal{F}$ to $\mathcal{Z}$. The activation sequence in this world is similar to the one in the real-world. The protocol execution is completed once $\mathcal{Z}$ stops activating other parties and outputs a single bit.

A protocol is said to be computationally (perfectly) UC-secure, if for every real-world PPT (unbounded) adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$, such that for any PPT environment $\mathcal{Z}$, the real-world is indistinguishable from the ideal-world, except with a negligible probability in the security parameter κ (zero probability).

The Hybrid Model: In a $\mathcal{G}$ -hybrid model, a protocol execution proceeds as in the real-world. However, the parties have access to an ideal functionality $\mathcal{G}$ for some specific task. The parties, during the protocol execution, communicate with $\mathcal{G}$ as in the ideal-world. The UC framework guarantees that an ideal functionality in a hybrid model can be replaced with a protocol that UC-securely realizes $\mathcal{G}$. This is specifically due to the following composition theorem from [17, 18].

Theorem 3.3 ([17, 18]). Let Π be a protocol that UC-securely realizes a functionality $\mathcal{F}$ in the $\mathcal{G}$ -hybrid model and let ρ be a protocol that UC-securely realizes $\mathcal{G}$. Moreover, let Π^ρ denote the protocol that is obtained from Π by replacing every ideal call to $\mathcal{G}$ with the protocol ρ . Then protocol Π^ρ UC-securely realizes $\mathcal{F}$ in the model where the parties do not have access to the ideal functionality $\mathcal{G}$.

Below, we present two ideal functionalities: (a) asynchronous blockchain and (b) asynchronous reliable broadcast (or Acast), and compare them.

3.3 The Ideal Blockchain Functionality

Conceptually, asynchronous blockchain, atomic broadcast (ABC), total order broadcast (TOB), and state machine replication (SMR) are the same: a collection of parties maintains a globally consistent, ordered log of messages/transactions despite network asynchrony and adversarial behavior. The two key guarantees provided by these can be captured by the following properties:

- **Persistence:** The local ledger/blockchain views of any two honest parties are always consistent with a common global total order. Formally, at any point of time, if TO_i and TO_j denote the local ledger views of two honest parties P_i and P_j , then either TO_i is a prefix of TO_j or vice versa.

- **Liveness:** Every message provided by an honest party is eventually appended to the local ledger/blockchain of all honest parties.

Following the simplified ledger/blockchain abstraction from [35], we model *only* the above two core properties in an asynchronous network through the ideal functionality $\mathcal{F}_{\text{BC}}$, presented in Fig 3.4. As mentioned in [35], for readers used to the formalization of a blockchain via a chain of blocks and properties like common prefix, chain quality and chain growth [39, 67], the formalization via $\mathcal{F}_{\text{BC}}$ might look overly simplified. That is meant as a feature. In particular, we abstract away several blockchain-specific technical details that are not relevant for our feasibility and impossibility results, and provide a simple, easy-to-use ledger functionality for submitting messages on a global ledger or a blockchain. The functionality provides the interface to any sender party to add a message to the global blockchain through a `Send` instruction; we assume that the functionality does the necessary “sanity checking” for the supplied input, before it proceeds to append the supplied input to the blockchain (for instance, if the blockchain is for a cryptocurrency, then check for valid signatures, double spend, etc). The functionality maintains a queue of pending inputs to be appended to the blockchain. To model the inherent limitations of an asynchronous network, where an adversary has full control over message scheduling, we utilize the notion of *request-based delayed output*⁹ from [27]. In this model, the functionality does not “push” updates to the parties immediately. Instead, each party P_i maintains a local blockchain view TO_i , which the party can access only by explicitly issuing a `Get` request. The functionality provides the adversary an interface to instruct which input from the queue to be added to the local copy of the blockchain of any party P_i , denoted by TO_i , through the `Add` instruction. Moreover, an adversary can instruct the functionality to delay appending any input from the queue; however, an adversary *cannot* do that indefinitely, and eventually every message from the queue gets appended to the blockchain. While the adversary has full “control” over which input transactions are added to the local copies of blockchain for various parties, the functionality ensures *persistence* and *liveness* property as defined above.

Functionality 3.4: $\mathcal{F}_{\text{BC}}$ – Asynchronous blockchain

$\mathcal{F}_{\text{BC}}$ proceeds as follows, running with the set of parties $\mathcal{P}$ and an adversary $\mathcal{S}$.

- **Initialization:** For every P_i , set the *total-ordered* set $\text{TO}_i = ()$ and $\text{Sent} = \emptyset$.
- Upon receiving $(P_i, \text{Send}, \text{sid}, m)$ from P_i , send $(P_i, \text{Send}, \text{sid}, m)$ to $\mathcal{S}$ and add (P_i, sid, m) to Sent .
- Upon receiving $(P_i, \text{Get}, \text{sid})$ from P_i , send $(P_i, \text{Get}, \text{sid})$ to $\mathcal{S}$ and send a request-based delayed output TO_i to P_i .¹⁰
- Upon receiving $(\text{Add}, P_i, (P_j, \text{sid}, m))$ from $\mathcal{S}$ where $(P_j, \text{sid}, m) \in \text{Sent}$, append (P_j, sid, m) to TO_i , satisfying the following constraints.

Persistence: If P_i and P_j are honest, then either TO_i is a prefix of TO_j or vice-versa.

Liveness: For every $(P_j, \text{sid}, m) \in \text{Sent}$, the message $(P_j, \text{sid}, m) \in \text{Sent}$ is eventually added to TO_i of every honest P_i . % This implicitly means that for every $(P_j, \text{sid}, m) \in \text{Sent}$, the adversary $\mathcal{S}$ has to eventually issue the $(\text{Add}, P_i, (P_j, \text{sid}, m))$ instruction, corresponding to every honest P_i .

Realizing $\mathcal{F}_{\text{BC}}$: Realizing $\mathcal{F}_{\text{BC}}$ over secure point-to-point channels is a problem of immense interest in the distributed computing spaces for more than forty years. Against a Byzantine adversary, we can realize $\mathcal{F}_{\text{BC}}$ in practice using protocols such as Tusk [34], Cordial Miners [57], or Ditto [41] for an honest super majority ($n > 3t$). For a fail-stop adversary, we can realize $\mathcal{F}_{\text{BC}}$ in practice using protocols such as Rabia [66] or QuePaxa [79] for an honest majority ($n > 2t$).

We note that our results hold even when get/read accesses to the ledger TO_i are synchronous, as long as, due to frontrunning [31] and other attacks [59, 65], writing to blockchains (i.e., `Send` and `Add` calls) is asynchronous. As a result, even more synchronous, slow, low-throughput blockchains that typically produce new blocks every round, such as Bitcoin and Ethereum, are also relevant here.

3.4 The Ideal Asynchronous Reliable Broadcast Functionality

A related primitive traditionally used in AVSS and AMPC protocols is that of *asynchronous reliable broadcast* (Acast). An Acast protocol allows a designated *sender* P_S to identically send its input m to all.

⁹ See Section 3.2 for formal description.

¹⁰ See Section 3.2 for the formal meaning of request-based delayed output.

If P_S is *honest*, then all honest parties eventually complete the protocol with output m . On the other hand, if P_S is *corrupt* and any honest party outputs some m^* , then eventually every honest party outputs m^* . The ideal functionality for asynchronous reliable broadcast is presented in Functionality 3.5. It can be securely realized with $n > 3t$ [14] and $n > 2t$ [4] against Byzantine and omission adversaries, respectively.

Functionality 3.5: $\mathcal{F}_{\text{Acast}}$ – Asynchronous reliable broadcast

$\mathcal{F}_{\text{Acast}}$ proceeds as follows, running with parties $\mathcal{P} = \{P_1, \dots, P_n\}$ and an adversary $\mathcal{S}$. Let $\mathcal{C}$ denote the set of corrupt parties. Upon receiving $(\text{Sender}, \text{Acast}, \text{sid}, m)$ from $P_S \in \mathcal{P}$, do the following:

- If $P_S \notin \mathcal{C}$, then send $(\text{Sender}, \text{Acast}, \text{sid}, m)$ to $\mathcal{S}$.
 - Send a request-based delayed output $(P_S, \text{Acast}, \text{sid}, m)$ to each $P_i \in \mathcal{P} \setminus \mathcal{C}$ (no need to send m to the parties in $\mathcal{C}$, as $\mathcal{S}$ gets m on their behalf).
-

Note that $\mathcal{F}_{\text{Acast}}$ is a “weaker” primitive than $\mathcal{F}_{\text{BC}}$. Namely, $\mathcal{F}_{\text{Acast}}$ can be trivially realized by asking the designated sender P_S to send its input m to all via $\mathcal{F}_{\text{BC}}$ by issuing a $(P_S, \text{Send}, \text{sid}, m)$ instruction, which will ensure that (P_S, sid, m) is eventually added to TO_i of every honest P_i .

3.5 Ideal Functionality for Asynchronous Verifiable Secret Sharing

The ideal functionality for AVSS is presented in Functionality 3.6. The functionality “verifies” if D has provided a t -degree polynomial as input and, upon verification, distributes distinct points on the polynomial as shares to respective parties. In essence, the functionality verifiably generates a Shamir-sharing [72] of an input held by D through a t -degree polynomial provided by D .

Functionality 3.6: $\mathcal{F}_{\text{VSS}}$ – VSS

$\mathcal{F}_{\text{VSS}}$ proceeds as follows, running with the parties in $\mathcal{P}$ and an adversary $\mathcal{S}$.

- Upon receiving $(\text{Dealer}, \text{sid}, q(\cdot))$ from $D \in \mathcal{P}$ where $q(\cdot)$ is a t -degree polynomial over $\mathbb{F}$, send a request-based delayed output $(\text{Dealer}, D, \text{sid}, q(i))$ to each party $P_i \in \mathcal{P}$.
-

4 Setting with no Setup Assumption

Here the parties have access to $\mathcal{F}_{\text{BC}}$, but otherwise provided with *no* set up.

4.1 Impossibility of AVSS Against Omission Adversary

We adapt the definition of AVSS from [19] against *omission* adversaries.

Definition 4.1 (AVSS [19]). *Let (sh, rec) be a pair of n -party protocols for sharing and reconstruction, respectively, where a designated dealer $D \in \mathcal{P}$ has input s for sh . Moreover, each honest party that completes protocol sh outputs a share, computed based on its view of sh and subsequently invokes rec with its view from sh as the local input. We say that (sh, rec) is an AVSS scheme if for every PPT omission adversary $\mathcal{A}$ corrupting up to t parties, all the following hold.*

- **Termination:**
 1. *If D is honest, then every honest party will eventually complete sh .*
 2. *If some honest party has completed sh , then all honest parties will eventually complete sh .*
 3. *If any honest party completes sh , then it will complete rec .*
- **Correctness:** *If any honest party completes sh , then all honest parties output s during rec , except with a negligible probability.*
- **Privacy:** *If D is honest and as long as no honest party invokes rec , then the view of $\mathcal{A}$ during sh reveals no additional information about s , except with a negligible probability.*

We next show that, *without* setup, we *cannot* realize AVSS against an omission adversary in the $\mathcal{F}_{\text{BC}}$ -hybrid model with $n \leq 3t$. The proof is via contradiction. Assuming the existence of an AVSS protocol, we define carefully chosen omission adversarial strategies and schedulers and compare the induced views of honest parties. We finally show a violation of the privacy of the AVSS protocol.

Theorem 4.2. *For $n \leq 3t$, there is no AVSS scheme without a trusted setup against a PPT omission adversary in the $\mathcal{F}_{\text{BC}}$ -hybrid model.*

Proof. Assume there exists an AVSS scheme (sh, rec) against omission adversaries for $n \leq 3t$, without any setup. First, we consider the minimal case when $n = 3$ and $t = 1$ and $\mathcal{P} = \{P_1^*, P_2^*, P_3^*\}$ with P_1^* be the dealer with input s and let r_i be the randomness held by P_i^* during sh . Let τ_{ij} denote the messages sent by party P_i^* to party P_j^* over the point-to-point channel and let B_i denote the messages published by party P_i^* over the blockchain via $\mathcal{F}_{\text{BC}}$ during sh . Consider the following adversarial strategies for sh , where the dealer's input is s and the randomness of the parties is the same in all three executions:

- **Adversary $\mathcal{A}_1$ (Fig. 1a):** Here adversary corrupts P_1^* , who drops the messages τ_{13} and τ_{31} , but behaves honestly otherwise. Moreover, P_1^* also drops B_1 messages once P_2^* terminates (P_1^* sends B_1 messages honestly as per the protocol until P_2^* terminates). The scheduler delivers the messages $\tau_{12}, \tau_{21}, \tau_{23}, B_1$ and B_2 without any delay, but delays the messages τ_{32} and B_3 , until P_2^* terminates sh . Note that P_2^* has to eventually terminate sh , as its view is identical to the execution under $\mathcal{A}_3$ defined below.
- **Adversary $\mathcal{A}_2$ (Fig. 1b):** Here adversary corrupts P_2^* , who drops the messages τ_{32} , but otherwise behaves honestly. The scheduler delivers the messages $\tau_{12}, \tau_{21}, \tau_{23}, B_1$ and B_2 without any delay, but delays the messages τ_{13}, τ_{31} and B_3 until P_1^* terminates sh . Note that P_1^* eventually terminates sh , as P_1^* is honest.
- **Adversary $\mathcal{A}_3$ (Fig. 1c):** Here adversary corrupts P_3^* , who drops the messages $\tau_{31}, \tau_{32}, \tau_{13}$ and B_3 . The scheduler delivers all the messages without any delay. Note that both P_1^* and P_2^* will eventually complete sh , as they are honest and receiving messages from $n - t$ parties.

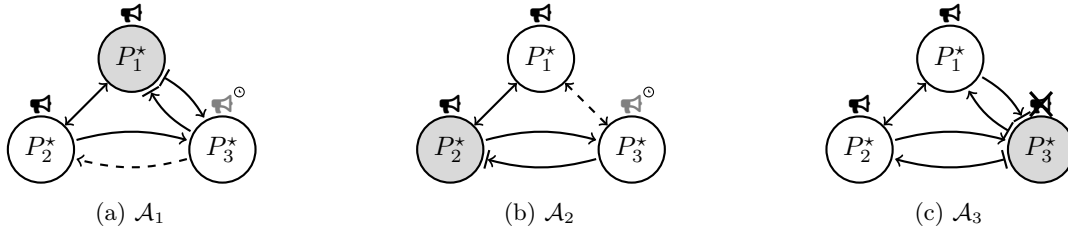


Fig. 1: Adversarial strategies during sh . The corrupted party is denoted by a gray circle. The solid lines denote communication without any delay. The dotted lines denote communication delay. The solid arrows with a cut in the front denote the receiver drops all the communication from the sender as an omission adversary. The solid arrow with a cut in the back denote the sender drops all the communication as an omission adversary to the receiver. The dark speaker symbol denotes broadcast without any delay, a gray speaker with a clock symbol denotes a delayed broadcast, and a speaker with a cross denotes no broadcast.

We show that the view of P_2^* is identical across all the adversaries. Let $\text{view}_{\mathcal{A}_j}^{\text{sh}}(P_i^*, r_i, s)$ denote the view of P_i^* during the execution of sh with P_1^* 's input being s and P_i^* 's randomness being r_i , in the presence of adversary $\mathcal{A}_j$.

Claim 4.3. *Fix the secret of the dealer as s , and the randomness of party P_i^* as r_i . Then every prefix of the view of P_2^* is identical under $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$. That is, $\text{view}_{\mathcal{A}_1}^{\text{sh}}(P_2^*, r_2, s) = \text{view}_{\mathcal{A}_2}^{\text{sh}}(P_2^*, r_2, s) = \text{view}_{\mathcal{A}_3}^{\text{sh}}(P_2^*, r_2, s)$.*

Proof. We first consider the views of P_2^* under $\mathcal{A}_1$ and $\mathcal{A}_2$. The proof for $\mathcal{A}_1$ and $\mathcal{A}_3$ works similarly. We prove the claim by induction on the length of prefixes of the view of P_2^* . A view prefix of length k consists of the first k events for P_2^* during the execution.

Base Case ($k = 0$): The empty view prefix is identical under $\mathcal{A}_1$ and $\mathcal{A}_2$.

Induction Hypothesis: Assume that the view prefixes of length k are identical.

Induction Step: The $(k + 1)$ -th event in P_2^* 's view is either a local computation, a point-to-point message receipt, or a delivery of a broadcast message from the $\mathcal{F}_{\text{BC}}$ channel.

- **Case I:** If the $(k + 1)$ -th event is a local computation, then the outcome is a *deterministic function of the view prefix of length k and P_2^* 's randomness*. By the induction hypothesis, the view prefixes of length k are identical, and the same randomness r_2 is used. Hence, the new event is identical.
- **Case II:** Let the $(k + 1)$ -th event be a receipt of a message. Under both $\mathcal{A}_1$ and $\mathcal{A}_2$, no message from P_3^* is delivered to either P_1^* or P_2^* neither in point-to-point channel nor in $\mathcal{F}_{\text{BC}}$ channel, and scheduler delivers messages from P_1^* without any delay. Since the randomness of the parties is the same across both adversaries, P_2^* will receive identical messages under both $\mathcal{A}_1$ and $\mathcal{A}_2$.

Therefore, by induction, the entire view of P_2^* is identical under $\mathcal{A}_1$ and $\mathcal{A}_2$. $\square$

Due to the termination property, under $\mathcal{A}_3$, the honest parties P_1^* and P_2^* must complete sh with some shares, say s_1 and s_2 respectively. Now by Claim 4.3, P_2^* 's view is identical across $\mathcal{A}_1$ and $\mathcal{A}_3$. So P_2^* must complete sh under $\mathcal{A}_1$ with s_2 as the share. Since P_3^* is honest under $\mathcal{A}_1$, by the termination property, P_3^* will also eventually complete sh with some share, say s_3 . Now note that under $\mathcal{A}_1$, P_1^* drops the messages τ_{13} and τ_{31} during sh. So share s_3 during sh is some *deterministic function*, say f , of its own randomness r_3 and the messages received. i.e., $s_3 = f(\tau_{23}, B_1, B_2, r_3)$.

Let us consider the execution of rec under $\mathcal{A}_1$, where the corrupt P_1^* drops the messages it is supposed to send, i.e. τ_{12}, τ_{13} and B_1 . The scheduler delivers all the messages without any delay. Since P_2^* , as an honest party, completed sh under $\mathcal{A}_1$, by the third requirement of the termination property of AVSS, it must also complete rec (even when P_1^* , the dealer, is corrupt) and output s during rec with a very high probability (over all possible randomnesses of the parties). Formally if κ is the security parameter, then $\Pr[\text{rec}(s_2, s_3) = s] = \Pr[\text{rec}(s_2, f(\tau_{23}, B_1, B_2, r_3)) = s] = 1 - \text{negl}(\kappa)$.

Now consider a different execution of the AVSS protocol where P_3^* participates with a different randomness r'_3 and the randomness of the remaining parties and the input of P_1^* remain the same. Moreover, the adversary strategies $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{A}_3$ remains the same. We can conclude that P_3^* will complete sh under $\mathcal{A}_1$, possibly with a different share $s'_3 = f(\tau_{23}, B_1, B_2, r'_3)$. Repeating the argument as above, under $\mathcal{A}_1$ during rec, P_2^* will reconstruct the secret s in this case with overwhelming probability (over all possible randomnesses of the parties) i.e., $\Pr[\text{rec}(s_2, s'_3) = s] = \Pr[\text{rec}(s_2, f(\tau_{23}, B_1, B_2, r'_3)) = s] = 1 - \text{negl}(\kappa)$. This shows that the reconstruction of s is independent of P_3^* 's randomness. Consequently, P_2^* can randomly pick $\tilde{r}_3$ as the randomness of P_3^* and compute its share locally as $\tilde{s}_3 = f(\tau_{23}, B_1, B_2, \tilde{r}_3)$ and then reconstruct the secret s by applying the reconstruction algorithm on s_2 and $\tilde{s}_3$ with an overwhelming probability.

By Claim 4.3, P_2^* 's view is identical under $\mathcal{A}_1$ and $\mathcal{A}_2$ during sh. So P_2^* can reconstruct s with the same probability under $\mathcal{A}_2$ during sh, which is a *violation* of the privacy property of AVSS. Thus no such (sh, rec) exists.

To prove the theorem for an arbitrary number of parties $n \leq 3t$, we apply the standard player partitioning argument. Let $\mathcal{P} = \{P_1, \dots, P_n\}$ denote the set of parties, and assume that at most t parties are corrupted. Since $n \leq 3t$, the set $\mathcal{P}$ can be partitioned into three pairwise disjoint subsets P_1^*, P_2^*, P_3^* such that $|P_i^*| \leq t$ for all $i \in \{1, 2, 3\}$. Each subset P_i^* is then treated as a single aggregated party. Consequently, the general n -party setting reduces to the corresponding simplified 3-party setting, which has already been established above. $\square$

We next discuss why Theorem 4.2 is “tight” in all aspects.

Theorem 4.2 does not hold against a fail-stop adversary. The adversarial strategies in the proof of Theorem 4.2 use the capabilities of an omission adversary. For instance, $\mathcal{A}_1$ instructs P_1^* to selectively crash towards *only* P_3^* , but behave honestly towards P_2^* *throughout* sh. Such a strategy is not valid for a fail-stop adversary, where a party once crashed, stop functioning for the entire execution. In the next section, we present a perfectly secure AVSS protocol against fail-stop adversaries with $n > 2t$, showing the tightness of our lower bound.

Theorem 4.2 does not hold with a setup. In the *presence* of a setup, say for PKI, a private

communication from P_1^* to P_3^* can be enabled via P_2^* . Specifically, P_1^* can encrypt messages for P_3^* using P_3^* 's public key and communicate them via P_2^* , which P_3^* alone can decrypt. So, despite the communication channel from P_1^* to P_3^* being disabled either via delaying or via P_1^* 's selective dropping, P_3^* 's share s_3 can depend on secret information, which is a function of the secret s , communicated to it by P_1^* through P_2^* .

4.2 Perfectly Secure VSS Against fail-stop Adversary with $n > 2t$

The high-level idea of the protocol has already been discussed in Section 2.2. The formal steps are presented in Protocol 4.4.

Protocol 4.4: Π_{PVSS} – Protocol for securely realizing $\mathcal{F}_{\text{VSS}}$ against fail-stop adversaries

Distributing shares and acknowledgment – Upon having the input $s \in \mathbb{F}$, dealer $D \in \mathcal{P}$ does as follows.

- Pick a uniformly random t -degree polynomial $q(\cdot)$ over $\mathbb{F}$ such that $q(0) = s$. Send $(\text{Dealer}, \text{sid}, q(j))$ to every $P_j \in \mathcal{P}$.
- After sending $(\text{Dealer}, \text{sid}, q(j))$ to every $P_j \in \mathcal{P}$, send $(\text{Dealer}, \text{sid}, \text{OK})$ to every $P_j \in \mathcal{P}$.

Forwarding acknowledgment and output computation — Every $P_j \in \mathcal{P}$ (including D) does below, upon receiving $(\text{Dealer}, \text{sid}, q(j))$ from D and $(\text{Dealer}, \text{sid}, \text{OK})$ from *any* party in $\mathcal{P}$: (a) Send $(\text{Dealer}, \text{sid}, \text{OK})$ to every party in $\mathcal{P}$. (b) Output $(\text{Dealer}, \text{sid}, q(j))$ and terminate.

Lemma 4.5. *Protocol Π_{PVSS} UC securely realizes $\mathcal{F}_{\text{VSS}}$ with perfect security. The protocol incurs a communication of $\mathcal{O}(n \log |\mathbb{F}| + n^2)$ bits.*

Proof. Dealer D sends the shares to all parties, which incurs a communication of $\mathcal{O}(n \log |\mathbb{F}|)$ bits. D also sends an OK message to all parties. After receiving the share and the OK message, each party sends an OK message to every other party. This incurs a communication of $\mathcal{O}(n^2)$ bits. Therefore, in total, the Π_{PVSS} protocol incurs a communication of $\mathcal{O}(n \log |\mathbb{F}| + n^2)$ bits.

For security, let $\mathcal{A}$ be an arbitrary real-world fail-stop adversary attacking the protocol Π_{PVSS} , and let $\mathcal{Z}$ be an arbitrary environment. We show that there exist a simulator $\mathcal{S}_{\text{PVSS}}$ such that, for any set of corrupted parties $\mathcal{C} \subseteq \mathcal{P}$ with $|\mathcal{C}| \leq t$ and for all inputs, the joint distribution of the view of the adversary and the output of all honest parties in a real-world execution of Π_{PVSS} with $\mathcal{A}$ is perfectly indistinguishable from the joint distribution of the view of the adversary and the of all honest parties in an ideal-world execution with $\mathcal{S}_{\text{PVSS}}$ interacting with $\mathcal{F}_{\text{VSS}}$. Consequently, the environment $\mathcal{Z}$ cannot distinguish between the real and ideal executions, thus proving the lemma.

Simulator 4.6: $\mathcal{S}_{\text{PVSS}}$: Simulator for the Protocol Π_{PVSS}

Let $\mathcal{C}$ denote the set of parties corrupted by the adversary $\mathcal{A}$, where $|\mathcal{C}| \leq t$, and let $\mathcal{H}$ be the set of honest parties. The simulator $\mathcal{S}_{\text{PVSS}}$ interacts with $\mathcal{A}$ and the ideal functionality $\mathcal{F}_{\text{VSS}}$ and simulates the view of $\mathcal{A}$, including its interaction with the environment $\mathcal{Z}$ and all messages sent by honest parties. To simulate $\mathcal{Z}$, the simulator forwards every message it receives from $\mathcal{Z}$ to $\mathcal{A}$, and vice versa. The simulator simulates the steps of the protocol as follows, depending upon whether $D \in \mathcal{C}$ or not.

- $D \notin \mathcal{C}$: The simulator picks a random t -degree polynomial $q(\cdot)$ over $\mathbb{F}$ with $q(0) = 0$ and sends $(\text{Dealer}, \text{sid}, q(i))$ and $(\text{Dealer}, \text{sid}, \text{OK})$ to $\mathcal{A}$ for each $P_i \in \mathcal{C}$, on behalf of D .
 - $D \in \mathcal{C}$: The simulator plays the role of the honest parties as per the steps of the protocol and interacts with $\mathcal{A}$. Upon receiving $(\text{Dealer}, \text{sid}, s_j)$ from $\mathcal{A}$ on behalf of the dealer for every $P_j \in \mathcal{H}$, interpolate the points $\{(j, s_j)\}_{P_j \in \mathcal{H}}$ and compute the t -degree polynomial $q(\cdot)$. Then upon receiving $(\text{Dealer}, \text{sid}, \text{OK})$ from $\mathcal{A}$ on behalf of the dealer for any $P_j \in \mathcal{H}$, send $q(\cdot)$ to $\mathcal{F}_{\text{VSS}}$.
-

We next prove a sequence of claims, which helps us to show that the joint distribution of the parties is identical in both the real-world, as well as simulated world. We first consider the case of an *honest* dealer. We show that the view generated by $\mathcal{S}_{\text{PVSS}}$ for $\mathcal{A}$ is perfectly indistinguishable from $\mathcal{A}$'s view during the real execution of Π_{PVSS} .

Claim 4.7. *If the dealer is honest, then the view of $\mathcal{A}$ in the simulated execution of Π_{PVSS} with $\mathcal{S}_{\text{PVSS}}$ is identically distributed as the view of $\mathcal{A}$ in the real execution of Π_{PVSS} involving honest parties.*

Proof. If $D \notin \mathcal{C}$, then the view of $\mathcal{A}$ during a real execution of Π_{PVSS} consists of t points on a random t -degree $q(\cdot)$, where $q(0) = s$ and the message OK. On the other hand, in the simulated execution, the view of $\mathcal{A}$ consists of t points on a random t -degree $q(\cdot)$, where $q(0) = 0$ and the message OK. Since the polynomial $q(\cdot)$ is picked uniformly at random in both the executions, it follows from the properties of t -degree polynomials over $\mathbb{F}$ [72] that the distribution of shares seen by $\mathcal{A}$ in both the executions are identical, this proving the claim. $\square$

We next claim that if the dealer is honest, then conditioned on the view of the adversary (which is identical in both the executions as per the previous claim), the outputs of the honest parties are identically distributed in the real world as well as in the ideal world.

Claim 4.8. *If D is honest, then conditioned on the view of $\mathcal{A}$, the output of the honest parties during the execution of Π_{PVSS} involving $\mathcal{A}$ has the same distribution as the output of the honest parties in the ideal world involving $\mathcal{S}_{\text{PVSS}}$ and $\mathcal{F}_{\text{VSS}}$.*

Proof. Let $D \notin \mathcal{C}$ and let view be an arbitrary view of $\mathcal{A}$. Moreover, let $\{q(i)\}_{P_i \in \mathcal{C}}$ be the shares of corrupt parties as per view . Notice that $q(\cdot)$ selected by D is uniformly distributed, conditioned on the shares $\{q(i)\}_{P_i \in \mathcal{C}}$ and $q(0)$; in fact, if $|\mathcal{C}| = t$, then $q(\cdot)$ gets fixed based on $\{q(i)\}_{P_i \in \mathcal{C}}$ and $q(0)$. Let us fix any such polynomial $q(\cdot)$. In the ideal world, each honest P_i obtains the request-based delayed output $q(i)$ from $\mathcal{F}_{\text{VSS}}$. We wish to show that each honest party P_i eventually outputs $q(i)$ in the real world as well. However, this simply follows from the fact that if D is honest throughout Π_{PVSS} , then each honest P_i eventually receives $q(i)$ from D and the message OK, either from D or from some other honest party. $\square$

We next prove certain claims with respect to a *corrupt* dealer.

Claim 4.9. *If the dealer is corrupt, then the view of $\mathcal{A}$ in the simulated execution of Π_{PVSS} with $\mathcal{S}_{\text{PVSS}}$ is identically distributed as the view of $\mathcal{A}$ in the real execution of Π_{PVSS} involving honest parties.*

Proof. The proof follows from the fact that if $D \in \mathcal{C}$, then $\mathcal{S}_{\text{PVSS}}$ participates in a full execution of the protocol Π_{PVSS} by playing the role of the honest parties. Hence, there is a one-to-one correspondence between simulated executions and real executions. $\square$

We next claim that if the dealer is corrupt, then conditioned on the view of the adversary (which is identical in both the executions as per the previous claim), the outputs of the honest parties are identically distributed in the real world as well as in the ideal world.

Claim 4.10. *If D is corrupt, then conditioned on the view of $\mathcal{A}$, the output of the honest parties during the execution of Π_{PVSS} involving $\mathcal{A}$ has the same distribution as the output of the honest parties in the ideal world involving $\mathcal{S}_{\text{PVSS}}$ and $\mathcal{F}_{\text{VSS}}$.*

Proof. Let $D \in \mathcal{C}$ be corrupt and let view be an arbitrary view of $\mathcal{A}$. We note that it can be found out whether at least one honest party has received an OK message from the dealer. So we consider the following two different cases.

- *As per view, no $P_j \in \mathcal{H}$ has received an OK message from D :* it is easy to see that in this case, the outputs of the honest parties are identically distributed in both the worlds. This is because in the real world, the honest parties *do not* obtain any output. Correspondingly, the simulator $\mathcal{S}_{\text{PVSS}}$ does not provide any input to $\mathcal{F}_{\text{VSS}}$ in the ideal world on behalf of the D and hence $\mathcal{F}_{\text{VSS}}$ does not generate any output for the honest parties.
- *As per view, there exists some $P_j \in \mathcal{H}$, who has received an OK message from D :* In this case, as per the steps of $\mathcal{S}_{\text{PVSS}}$, it would have received the share s_j for every $P_j \in \mathcal{H}$, where $\{(j, s_j)\}_{P_j \in \mathcal{H}}$ lie on a t -degree polynomial $q(\cdot)$ and forward $q(\cdot)$ to $\mathcal{F}_{\text{VSS}}$. Consequently, $\mathcal{F}_{\text{VSS}}$ would generate a request-based delayed output of $q(j)$ for every $P_j \in \mathcal{H}$. We show that even in the real execution, every $P_j \in \mathcal{H}$ eventually outputs $q(j)$.

Let $P_j \in \mathcal{H}$ who receives OK message from D in the real execution, as per view. Note that as per the steps of Π_{PVSS} , dealer D is supposed to send OK message to P_j , *only* after sending the share $q(k)$ to every $P_k \in \mathcal{H}$. Moreover, since a fail-stop adversary cannot enforce D to deviate from the protocol

instructions, it implies that D would have sent $q(k)$ to every $P_k \in \mathcal{H}$ before sending OK message to P_j . Consequently, every $P_k \in \mathcal{H}$ will eventually receive $q(k)$ from D . Moreover, as per the steps of Π_{PVSS} , party P_j would forward OK to every party in $\mathcal{H}$ and consequently, every $P_k \in \mathcal{H}$ would output $q(k)$ during the real execution as well. □

The lemma now follows from Claims 4.7-4.10. □

5 Setting with Setup and Minicrypt Assumptions

5.1 Impossibility of MPC with Setup and Minicrypt Assumptions

We first show that in the *synchronous* model, there exists an n -party functionality, which *cannot* be securely computed against *semi-honest* adversaries with $n \leq 2t$ and Minicrypt, even with access to $\mathcal{F}_{\text{BC}}$ functionality (Lemma 5.1). We then use the reduction of *asynchronous* MPC with $n \leq 3t$ against *fail-stop* adversaries to *synchronous* MPC with $n \leq 2t$ against *semi-honest* adversaries, due to Canetti [19]. These two results together imply the impossibility of asynchronous MPC with $n \leq 3t$ and Minicrypt against fail-stop adversaries.

Lemma 5.1. *For $n \leq 2t$, there exist functionalities that cannot be securely computed in Minicrypt by any synchronous n -party protocol against a computationally bounded, semi-honest adversary corrupting at most t parties. Moreover, this holds even if the parties have access to any kind of trusted setup realizable within Minicrypt.*

Proof. First consider the case when $n = 2$ and $t = 1$ and $\mathcal{P} = \{P_1^*, P_2^*\}$. Then consider the 1-out-of-2 oblivious transfer (OT) functionality [69]:

$$\mathcal{F}_{\text{OT}}(m_0, m_1; b) := (\perp, m_b),$$

where P_1^* has inputs m_0, m_1 and party P_2^* has a selection bit $b \in \{0, 1\}$. The function outputs m_b for P_2^* . From [53], it is well-known that there exists no protocol with Minicrypt to securely compute $\mathcal{F}_{\text{OT}}$ in the *synchronous* communication setting against a computationally bounded semi-honest adversary corrupting either P_1^* or P_2^* , even if P_1^* and P_2^* have access to any kind of trusted setup.

To prove the lemma for any general n where $n \leq 2t$, we use the standard player partitioning argument. Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be the set of parties and consider the n -party functionality f : $f((m_0, m_1); \perp, \dots, \perp, b) := (\perp, \dots, \perp, m_b)$, where P_1 and P_n has private inputs (m_0, m_1) and $b \in \{0, 1\}$ respectively and rest of the other parties have input $\perp$. Moreover, the function outputs m_b for P_n and $\perp$ for the remaining parties. We claim that there exists no protocol with Minicrypt to securely compute f in the synchronous communication setting against a computationally bounded semi-honest adversary corrupting up to t parties. Moreover, this holds even if the parties have access to any kind of trusted setup. This will complete the proof of the Lemma.

On the contrary, suppose we have a protocol Π to securely compute f with Minicrypt. Then consider the following 2-party OT protocol Π_{OT} with Minicrypt for P_1^* and P_2^* , where P_1^* and P_2^* has inputs (m_0, m_1) and $b \in \{0, 1\}$ respectively. In Π_{OT} , party P_1^* and P_2^* emulates the role of the parties in $\mathcal{S}_1 = \{P_1, \dots, P_t\}$ and $\mathcal{S}_2 = \{P_{t+1}, \dots, P_n\}$ respectively, with P_1 's and P_n 's inputs being set to (m_0, m_1) and b respectively. The output of P_2^* in Π_{OT} is set to be the output of P_n in the emulated execution of Π . It then follows that if Π is a secure protocol against a semi-honest adversary corrupting up to t parties, then Π_{OT} is also a secure protocol against a semi-honest adversary corrupting either P_1^* or P_2^* . However, no such protocol Π_{OT} exists, and hence Π also does not exist. □

Remark 5.2 (Lemma 5.1 holds in the $\mathcal{F}_{\text{BC}}$ -hybrid). *We stress that Lemma 5.1 is true even if the parties have access to $\mathcal{F}_{\text{BC}}$ functionality. This is because $\mathcal{F}_{\text{BC}}$ can be securely realized in the synchronous communication setting against a semi-honest adversary for any $t < n$ by the “trivial” one-round broadcast protocol, where any designated sender party sends its input to all over the point-to-point channels. At the end of the round, the parties can simply order the outputs received from the various sender parties, according to the indices of the sender parties, with P_1 and P_n being considered as the least-indexed and highest-indexed sender party respectively. So if there exists any protocol where parties have access to $\mathcal{F}_{\text{BC}}$,*

the same protocol can be compiled into a protocol over the point-to-point channels. Contra-positively, if there exists no protocol over point-to-point channels, then there exists no protocol even if the parties have access to $\mathcal{F}_{\text{BC}}$.

Lemma 5.3 ([19]). *Let $m \leq 2t$ and $n \leq 3t$. Let $f : (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$ and $f' : (\{0,1\}^*)^m \rightarrow (\{0,1\}^*)^m$ be functions such that for any n -vector $\vec{x} = (x_1, \dots, x_n)$ where each $x_i \in \{0,1\}^*$, the condition $f(\vec{x}) = f'(x_1, \dots, x_m)$ holds. Let $\Pi_{\text{A,FS}}^f$ be an n -party protocol over point-to-point channels for securely computing f in the asynchronous setting against a fail-stop adversary corrupting up to t parties. Then there exists an m -party protocol $\Pi_{\text{S,SH}}^{f'}$ over point-to-point channels for securely computing f' in the synchronous setting against a semi-honest adversary corrupting up to t parties, where $\Pi_{\text{S,SH}}^{f'}$ is based on same primitives and same setup assumptions (if any) used in $\Pi_{\text{A,FS}}^f$.*

Theorem 5.4. *For $n \leq 3t$, there exist functionalities that cannot be securely computed in Minicrypt by any asynchronous n -party protocol against a computationally bounded, fail-stop adversary corrupting at most t parties. This holds even if the parties have access to $\mathcal{F}_{\text{BC}}$ functionality and any kind of trusted setup.*

Proof. Without loss of generality, let $n = 3t$ and $m = n - t = 2t$. From Lemma 5.1, there exists a function $f' : (\{0,1\}^*)^m \rightarrow (\{0,1\}^*)^m$, which cannot be securely computed in the synchronous communication setting with Minicrypt against a semi-honest adversary corrupting up to t parties, even if the parties have access to any kind of trusted setup. Corresponding to f' , let $f : (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$ be the n -party function, such that for any n -vector $\vec{x} = (x_1, \dots, x_n)$ where each $x_i \in \{0,1\}^*$, the condition $f(\vec{x}) = f'(x_1, \dots, x_m)$ holds. We claim that there exists no n -party asynchronous protocol for f with Minicrypt against a computationally bounded, fail-stop adversary corrupting at most t parties, even if the parties have access to the $\mathcal{F}_{\text{BC}}$ functionality and any kind of trusted setup. The proof is by contradiction. Assume such an asynchronous protocol, say $\Pi_{\text{A,FS}}^f$ exists.

We first note that since $n = 3t$, the functionality $\mathcal{F}_{\text{BC}}$ can be securely realized over point-to-point channels by a randomized asynchronous protocol, tolerating a computationally bounded fail-stop adversary corrupting t parties [66, 79]. Hence, we can assume that in protocol $\Pi_{\text{A,FS}}^f$, all communication happens over point-to-point channels. It then follows from Lemma 5.3 that there also exists a synchronous protocol $\Pi_{\text{S,SH}}^{f'}$ for f' with Minicrypt against a computationally bounded semi-honest adversary corrupting at most t parties. However, from Lemma 5.1, no such protocol $\Pi_{\text{S,SH}}^{f'}$ exists, thus proving the theorem. $\square$

5.2 AVSS with Honest Majority and Minicrypt in $\mathcal{F}_{\text{Acast-hybrid}}$

Here we present a computationally secure AVSS against Byzantine adversaries with $n > 2t$ and Minicrypt in the $\mathcal{F}_{\text{Acast-hybrid}}$ model. Throughout this section, we assume that κ is the computational security parameter. We assume the existence of a symmetric-key encryption scheme, with key-generation algorithm SymKeyGen , encryption algorithm SymEnc , and a decryption algorithm SymDec . The scheme is assumed to be secure against a standard chosen-plaintext attack (CPA). We also assume a commitment scheme over $\mathbb{F}$, consisting of the parameter-generation algorithm CommitmentGen , a commitment algorithm Commit , and an opening algorithm Open . The scheme provides the standard hiding and binding properties. Additionally, we assume that the commitment scheme is UC-secure, which can be instantiated in the programmable random oracle model using a collision-resistant hash function. Such UC-secure non-interactive commitments in the random oracle model are known from [52].

Symmetric-key Encryption Scheme. A private key encryption scheme with the underlying message space $\mathcal{M} = \mathbb{F}$, key space $\mathcal{K}$, and ciphertext space $\mathcal{CT}$, is a collection of the following PPT algorithms, parametrized by a computational security parameter κ .

1. $\text{KeyGen}(1^\kappa) \rightarrow k$: The key-generation algorithm outputs a private key $k \in \mathcal{K}$.
2. $\text{Enc}_k(m, r) \rightarrow \mathbf{c}_m$: The encryption algorithm computes a ciphertext $\mathbf{c}_m \in \mathcal{CT}$, which encrypts a plaintext $m \in \mathcal{M}$ under the key k , using the randomness r and outputs $\mathbf{c}_m$.
3. $\text{Dec}_k(\mathbf{c}_x) := x$: The decryption algorithm decrypts a ciphertext $\mathbf{c}_x \in \mathcal{CT}$, using the key k and outputs a plaintext $x \in \mathcal{M}$.

We require the following security properties from an encryption scheme.

- *Correctness*: For any k obtained as the output of **KeyGen**, the following should hold:

$$\text{Dec}_k(\text{Enc}_k(m)) = m.$$

- *Semantic Security*: For all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following experiment is negligible in κ .
 - *Key Generation*: The challenger runs **KeyGen**(1^κ) to obtain k .
 - *Challenge*: $\mathcal{A}$ sends plain-texts $m_0, m_1 \in \mathcal{M}$ to the challenger, who randomly selects $b \in \{0, 1\}$, computes $\mathbf{c}_{m_b} = \text{Enc}_k(m_b, r)$ for some randomness r and sends $\mathbf{c}_{m_b}$ to $\mathcal{A}$.
 - *Output*: The adversary $\mathcal{A}$ outputs $b' \in \{0, 1\}$.

The advantage of $\mathcal{A}$ in the above experiment is defined to be $|\frac{1}{2} - \Pr[b' = b]|$.

Commitment Scheme: Such a scheme generates commitments of size $\mathcal{O}(\kappa)$ bits, provides the hiding and binding properties.

Definition 5.5 (Commitment Scheme). *A (non-interactive) commitment scheme with the underlying message space $\mathcal{M} = \mathbb{F}$ is a collection of the following PPT algorithms, parametrized by a security parameter κ .*

- **CommitmentGen**(1^κ) $\rightarrow$ **params**: the parameter-generation algorithm outputs public parameters **params**.
- **Commit**(**params**, m , r) $:=$ **com** _{m} : the commit algorithm takes input the public parameters **params**, message $m \in \mathcal{M}$ to be committed and a randomness r and outputs a commitment **com** _{m} of m under the randomness r , where **com** _{m} consists of $\mathcal{O}(\kappa)$ bits.
- **Open**(**params**, m , r , **com** _{m}) $\rightarrow \{0, 1\}$: the opening algorithm takes input the public parameters **params**, message m and a randomness r , along with a commitment **com** _{m} and outputs either 0 (reject) or 1 (accept).

We require the following security properties from a commitment scheme.

- *Correctness*: For every **params** obtained by running **CommitmentGen**, every $m \in \mathcal{M}$ and every randomness r , the following should hold.

$$\text{Open}(\text{params}, m, r, \text{Commit}(\text{params}, m, r)) = 1.$$

- *Hiding*: For all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following experiment should be negligible in κ .
 - *Parameter generation*: The challenger runs **CommitmentGen**(1^κ) to obtain **params** and sends **params** to $\mathcal{A}$.
 - *Challenge*: $\mathcal{A}$ sends $m_0, m_1 \in \mathcal{M}$ to the challenger, who randomly selects $b \in \{0, 1\}$, computes **com** _{m_b} = **Commit**(**params**, m_b , r) for some randomness r and sends **com** _{m_b} to $\mathcal{A}$.
 - *Output*: $\mathcal{A}$ outputs $b' \in \{0, 1\}$.

The advantage of $\mathcal{A}$ in the above experiment is defined to be $|\frac{1}{2} - \Pr(b = b')|$.

- *Binding*: For all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following experiment should be negligible in κ .
 - *Parameter generation*: The challenger runs **CommitmentGen**(1^κ) to obtain **params** and sends **params** to $\mathcal{A}$.
 - *Output*: $\mathcal{A}$ outputs (**com** _{m} , m , r , m' , r'), where $m, m' \in \mathcal{M}$ and $m \neq m'$.

The advantage of $\mathcal{A}$ in the above experiment is defined to be

$$\Pr[\text{Open}(m, r, \text{com}_m) = 1 \neq \text{Open}(m', r', \text{com}_m)]$$

In the rest of the paper, while using the commitment scheme, we will assume that the public parameters **params** have already been generated by a trusted setup and avoid mentioning **params** explicitly while invoking algorithms **Commit** and **Open**.

One-to-many zero-knowledge (ZK). In our AVSS, we use zero-knowledge protocols, where a designated prover proves the correctness of a statement to *all*. In the generic ZK functionality (Functionality 5.6), parameterized by a relation R , the designated *prover* sends the functionality a statement x to be proven along with a witness w . In response, the functionality forwards the statement x to all iff $R(x, w) = 1$ (i.e., x is a correct statement and w is a witness for x).

Functionality 5.6: $\mathcal{F}_{\text{ZK}}^{1:M}$ – One-to-many zero-knowledge

$\mathcal{F}_{\text{ZK}}^{1:M}$ proceeds as follows, running with a designated prover $P \in \mathcal{P}$, the set of verifiers $\mathcal{P}$, an adversary $\mathcal{S}$ and parametrized with a relation R . On receiving (ZK-prover, sid, x, w) from P , do the following if $R(x, w) = 1$: Send a request-based delayed output (ZK-proof, sid, x) to $\mathcal{S}$ and to all the parties in $\mathcal{P}$.

For our AVSS, we assume a zero-knowledge (ZK) protocol for the relation R_{PoCSS} (proof-of-correct-secret-sharing). Such a protocol can be constructed using the MPC-in-the-Head paradigm combined with the Fiat–Shamir transform in the random oracle model (ROM), following the techniques of [22]. The relation R_{PoCSS} is parameterized by the symmetric-key encryption scheme, the commitment scheme, and t . The statement consists of commitments $\text{com}_{s_1}, \dots, \text{com}_{s_n}$ and ciphertexts $\mathbf{C}_{s_1}, \dots, \mathbf{C}_{s_n}$. The witness consists of a secret $s \in \mathbb{F}$, shares $s_1, \dots, s_n \in \mathbb{F}$, randomness $r_1, \dots, r_n$ and $r'_1, \dots, r'_n$ and symmetric-keys $k_1, \dots, k_n$, explaining that $((1, s_1), \dots, (n, s_n))$ lie on a t -degree polynomial with constant term s and each $\mathbf{C}_{s_i}$ is an encryption of $s_i \| r_i$ using the randomness r'_i , under the key k_i . Moreover, com_{s_i} is the commitment of s_i , under the randomness r_i . That is,

$$R_{\text{PoCSS}} = \left\{ ((\mathbf{C}_{s_1}, \dots, \mathbf{C}_{s_n}), (\text{com}_{s_1}, \dots, \text{com}_{s_n})), (s, (s_1, \dots, s_n), (r_1, \dots, r_n), (r'_1, \dots, r'_n), (k_1, \dots, k_n)) \mid \right. \\ \left. \exists f(x) \in \mathcal{P}^{s,t} \wedge s_i = f(i) \wedge \text{com}_{s_i} = \text{Commit}(s_i, r_i) \wedge \mathbf{C}_{s_i} = \text{Enc}_{k_i}(s_i \| r_i, r'_i) \right\},$$

where $\mathcal{P}^{s,t}$ is the set of all t -degree polynomials with s as the constant term.

Setup for pair-wise symmetric key. Our AVSS assumes a one-time key setup, modeled by functionality 5.7, where for every pair of parties, a common key for the symmetric encryption scheme is generated. Additionally, the public parameters for the commitment scheme are made available to all. This is a one-time setup, which can be reused across multiple AVSS instances.

Functionality 5.7: $\mathcal{F}_{\text{Setup}}$ – One-time setup generation

The functionality interacts with the parties in $\mathcal{P}$ and an adversary $\mathcal{S}$ as follows: On receiving **Setup** from at least $t + 1$ parties, $\mathcal{F}_{\text{Setup}}$ does the following.

- For every $P_i, P_j \in \mathcal{P}$, compute $\text{SymKeyGen}(1^\kappa) \rightarrow k_{ij}$ and set $k_{ji} = k_{ij}$.
 - Compute $\text{CommitmentGen}(1^\kappa) \rightarrow \text{params}$.
 - Send a request-based delayed output $(k_{i1}, \dots, k_{in}, \text{params})$ to each P_i .
-

The high-level idea of the AVSS has already been discussed in Section 2.4. The formal steps are presented in Protocol 5.8.

Protocol 5.8: Π_{SKVSS} — Symmetric key based AVSS protocol in $(\mathcal{F}_{\text{Setup}}, \mathcal{F}_{\text{Acast}}, \mathcal{F}_{\text{ZK}}^{1:M})$ -hybrid model

Generating One-Time Setup : Each $P_i \in \mathcal{P}$ sends **Setup** to $\mathcal{F}_{\text{Setup}}$ and requests output from $\mathcal{F}_{\text{Setup}}$ till receiving $(k_{i1}, \dots, k_{in}, \text{params})$.

Sharing — Upon having input $s \in \mathbb{F}$, dealer does the following.¹¹

1. Select a random t -degree polynomial $f(\cdot)$ over $\mathbb{F}$ such that $f(0) = s$.
2. For every $P_i \in \mathcal{P}$, compute share $s_i = f(i)$, commitment $\text{com}_{s_i} = \text{Commit}(s_i, r_i)$ using a uniformly random randomness r_i and ciphertext $\mathbf{C}_{s_i} = \text{Enc}_{k_{1i}}(s_i \| r_i, r'_i)$ using a uniformly random randomness r'_i .

¹¹ To avoid notational clutter, we assume P_1 is the dealer.

3. Send $(\text{Sender}, \text{Acast}, \text{sid}, (\mathbf{C}_{s_1}, \dots, \mathbf{C}_{s_n}, \mathbf{com}_{s_1}, \dots, \mathbf{com}_{s_n}))$ to $\mathcal{F}_{\text{Acast}}$.
4. As a prover, send $(\text{ZK-prover}, \text{sid}, ((\mathbf{C}_{s_1}, \dots, \mathbf{C}_{s_n}), (\mathbf{com}_{s_1}, \dots, \mathbf{com}_{s_n})), (s, (s_1, \dots, s_n), (r_1, \dots, r_n), (r'_1, \dots, r'_n), (k_{11}, \dots, k_{1n})))$ to $\mathcal{F}_{\text{ZK}}^{1:M}$ for R_{PoCSS} .

Verification of shares and output computation — Every $P_i \in \mathcal{P}$ (including dealer) executes the following code.

1. Request output from $\mathcal{F}_{\text{Acast}}$ until receiving $(P_1, \text{Acast}, \text{sid}, (\mathbf{C}_{s_1}, \dots, \mathbf{C}_{s_n}, \mathbf{com}_{s_1}, \dots, \mathbf{com}_{s_n}))$.
 2. Request output from $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoCSS} until receiving $(\text{ZK-proof}, \text{sid}, ((\mathbf{C}_{s_1}, \dots, \mathbf{C}_{s_n}), (\mathbf{com}_{s_1}, \dots, \mathbf{com}_{s_n})))$.
 3. Compute $s_i \| r_i = \text{Dec}_{k_{i1}}(\mathbf{C}_{s_i})$ and output s_i .
-

Lemma 5.9. *Assuming the existence of a CPA-secure symmetric-key encryption scheme and a UC-secure commitment scheme, protocol Π_{SKVSS} UC securely realizes $\mathcal{F}_{\text{VSS}}$ with computational security and Minicrypt in the $(\mathcal{F}_{\text{Setup}}, \mathcal{F}_{\text{Acast}})$ -hybrid model and $\mathcal{F}_{\text{ZK}}^{1:M}$ -hybrid model for the relation R_{PoCSS} .*

Proof. Let $\mathcal{A}$ be an arbitrary real-world Byzantine adversary attacking the protocol Π_{SKVSS} , and let $\mathcal{Z}$ be an arbitrary environment. We show that there exist a simulator $\mathcal{S}_{\text{SKVSS}}$ such that, for any set of corrupted parties $\mathcal{C} \subseteq \mathcal{P}$ with $|\mathcal{C}| \leq t$ and for all inputs, the joint distribution of the view of the adversary and the output of all honest parties in a real-world execution of Π_{SKVSS} with $\mathcal{A}$ is computationally indistinguishable from the joint distribution of the view of the adversary and the output of all honest parties in an ideal-world execution with $\mathcal{S}_{\text{SKVSS}}$ interacting with $\mathcal{F}_{\text{VSS}}$ except with a negligible probability. Consequently, the environment $\mathcal{Z}$ cannot distinguish between the real and ideal executions except with a negligible probability, thus proving the lemma.

Simulator 5.10: $\mathcal{S}_{\text{SKVSS}}$: Simulator for the Protocol Π_{SKVSS}

Let $\mathcal{C}$ denote the set of parties corrupted by the adversary $\mathcal{A}$, where $|\mathcal{C}| \leq t$, and let $\mathcal{H}$ be the set of honest parties. The simulator $\mathcal{S}_{\text{SKVSS}}$ interacts with $\mathcal{A}$ and the ideal functionality $\mathcal{F}_{\text{VSS}}$ and simulates the view of $\mathcal{A}$, including its interaction with the environment $\mathcal{Z}$ and all messages sent by honest parties. To simulate $\mathcal{Z}$, the simulator forwards every message it receives from $\mathcal{Z}$ to $\mathcal{A}$, and vice versa. The simulator simulates the steps of the protocol as follows, depending upon whether the dealer $P_1 \in \mathcal{C}$ or not.

Generating One-Time Setup

Upon receiving **Setup** from $\mathcal{A}$, the simulator itself plays the role of $\mathcal{F}_{\text{Setup}}$ and does the following.

1. Compute $\text{KeyGen}(1^\kappa) \rightarrow \tilde{k}_{ij}$ and sets $\tilde{k}_{ji} = \tilde{k}_{ij}$ for every pair of parties $(P_i, P_j) \in \mathcal{P}$.
2. Compute $\text{CommitmentGen}(1^\kappa) \rightarrow \widetilde{\text{params}}$.
3. Corresponding to every $P_i \in \mathcal{C}$, send $(\tilde{k}_{i1}, \dots, \tilde{k}_{in}, \widetilde{\text{params}})$ to $\mathcal{A}$ on behalf of P_i .

Sharing

- If $P_1 \notin \mathcal{C}$: The simulator interacts with $\mathcal{F}_{\text{VSS}}$ and receives $(\text{Dealer}, D, \text{sid}, \tilde{s}_i)$ from $\mathcal{F}_{\text{VSS}}$ on behalf of each $P_i \in \mathcal{C}$. The simulator then computes $\widetilde{\text{com}}_{\tilde{s}_i} = \text{Commit}(\tilde{s}_i, \tilde{r}_i)$ using uniform randomness $\tilde{r}_i$ and $\widetilde{\mathbf{C}}_{\tilde{s}_i} = \text{Enc}_{\tilde{k}_{i1}}(\tilde{s}_i \| \tilde{r}_i, \tilde{r}'_i)$ using uniform randomness $\tilde{r}'_i$ for every $P_i \in \mathcal{C}$. It also computes $\widetilde{\text{com}}_{\tilde{s}_j} = \text{Commit}(0, \tilde{r}_j)$ using uniform randomness $\tilde{r}_j$ and $\widetilde{\mathbf{C}}_{\tilde{s}_j} = \text{Enc}_{\tilde{k}_{1j}}(0 \| \tilde{r}_j, \tilde{r}'_j)$ using uniform randomness $\tilde{r}'_j$ for every $P_j \in \mathcal{H}$. Simulator itself performs the role of $\mathcal{F}_{\text{Acast}}$ and $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoCSS} . Whenever $\mathcal{A}$ requests output from $\mathcal{F}_{\text{Acast}}$, simulator responds with $(\text{Dealer}, \text{sid}, \widetilde{\mathbf{C}}_{\tilde{s}_1}, \dots, \widetilde{\mathbf{C}}_{\tilde{s}_n}, \widetilde{\text{com}}_{\tilde{s}_1}, \dots, \widetilde{\text{com}}_{\tilde{s}_n})$ and whenever $\mathcal{A}$ requests output from $\mathcal{F}_{\text{ZK}}^{1:M}$, simulator responds with $(\text{ZK-proof}, \text{sid}, (\widetilde{\mathbf{C}}_{\tilde{s}_1}, \dots, \widetilde{\mathbf{C}}_{\tilde{s}_n}), (\widetilde{\text{com}}_{\tilde{s}_1}, \dots, \widetilde{\text{com}}_{\tilde{s}_n}))$.

- $P_1 \in \mathcal{C}$: The simulator itself performs the role of $\mathcal{F}_{\text{Acast}}$ and $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoCSS} . Upon receiving $(\text{Dealer}, \text{sid}, \widetilde{\mathbf{C}}_{\widetilde{s}_1}, \dots, \widetilde{\mathbf{C}}_{\widetilde{s}_n}, \widetilde{\mathbf{com}}_{\widetilde{s}_1}, \dots, \widetilde{\mathbf{com}}_{\widetilde{s}_n})$ from $\mathcal{A}$ on behalf of P_1 for $\mathcal{F}_{\text{Acast}}$ and $(\text{ZK-prover}, \text{sid}, (\widetilde{\mathbf{C}}_{\widetilde{s}_1}, \dots, \widetilde{\mathbf{C}}_{\widetilde{s}_n}), (\widetilde{\mathbf{com}}_{\widetilde{s}_1}, \dots, \widetilde{\mathbf{com}}_{\widetilde{s}_n}))$ from $\mathcal{A}$ on behalf of P_1 for $\mathcal{F}_{\text{ZK}}^{1:M}$, the simulator computes $\text{Dec}_{\widetilde{k}_{1i}}(\widetilde{\mathbf{C}}_{\widetilde{s}_i}) = \widetilde{s}_i \parallel \widetilde{r}_i$, for $i = 1, \dots, n$. The simulator then interpolates the points $(1, \widetilde{s}_1), \dots, (n, \widetilde{s}_n)$ to compute a polynomial $\widetilde{f}(\cdot)$ over $\mathbb{F}$ and sends $\widetilde{f}(\cdot)$ to $\mathcal{F}_{\text{VSS}}$, if all the following holds.
 - $\widetilde{f}(\cdot)$ is a t -degree polynomial.
 - $\widetilde{\mathbf{com}}_{\widetilde{s}_i} = \text{Commit}(\widetilde{s}_i, \widetilde{r}_i)$ holds, for $i = 1, \dots, n$.
-

We next prove a sequence of claims, which helps us to show that the joint distribution of the parties are identical in both the real-world, as well as simulated world. We first consider the case of an *honest* dealer. We show that the view generated by $\mathcal{S}_{\text{SKVSS}}$ for $\mathcal{A}$ is indistinguishable from $\mathcal{A}$'s view during the real execution of Π_{SKVSS} except with negligible probability.

Claim 5.11. *If the dealer is honest, then the view of $\mathcal{A}$ in the simulated execution of Π_{SKVSS} with $\mathcal{S}_{\text{SKVSS}}$ is identically distributed as the view of $\mathcal{A}$ in the real execution of Π_{SKVSS} involving honest parties except with negligible probability.*

Proof. It is easy to see that the view of $\mathcal{A}$ during the setup generation phase is identically distributed in both executions. This is because in both executions, $\mathcal{A}$ receives no messages from the honest parties, and the step of $\mathcal{F}_{\text{Setup}}$ is executed by the simulator itself in the simulated execution. Namely, in both executions, $\mathcal{A}$'s view consists of the setup data for the encryption scheme and commitment scheme. So let us fix the data held by $\mathcal{A}$ at the end of the setup generation phase in both the executions.

Now conditioned on the view of $\mathcal{A}$ for the setup generation phase, if $D \notin \mathcal{C}$, then the view of $\mathcal{A}$ during a real execution of Π_{SKVSS} consists of $(\mathbf{C}_{s_1}, \dots, \mathbf{C}_{s_n}, \mathbf{com}_{s_1}, \dots, \mathbf{com}_{s_n})$ and the corresponding ZK proof. $\mathcal{A}$ can decrypt only t of the $\mathbf{C}_{s_i}$'s and can verify against the commitment $\mathbf{com}_{s_i}$, corresponding to every $P_i \in \mathcal{C}$ and the t points lie on a random t -degree $f(\cdot)$, where $f(0) = s$. On the other hand, in the simulated execution, the view of $\mathcal{A}$ consists $(\widetilde{\mathbf{C}}_{\widetilde{s}_1}, \dots, \widetilde{\mathbf{C}}_{\widetilde{s}_n}), (\widetilde{\mathbf{com}}_{\widetilde{s}_1}, \dots, \widetilde{\mathbf{com}}_{\widetilde{s}_n})$ and the corresponding ZK proof. Even in this case, $\mathcal{A}$ can decrypt only t of the $\widetilde{\mathbf{C}}_{\widetilde{s}_i}$'s and can verify against the commitment $\widetilde{\mathbf{com}}_{\widetilde{s}_i}$, corresponding to every $P_i \in \mathcal{C}$ and the t points lie on a random t -degree $\widetilde{f}(\cdot)$, where $\widetilde{f}(0) = s$. By the construction of simulator, $s_i = \widetilde{s}_i$ for every $P_i \in \mathcal{C}$. From the semantic security of encryption scheme and hiding property of the commitment scheme, $\{(\mathbf{C}_{s_j}, \mathbf{com}_{s_j})\}_{P_j \in \mathcal{H}}$ and $\{(\widetilde{\mathbf{C}}_{\widetilde{s}_j}, \widetilde{\mathbf{com}}_{\widetilde{s}_j})\}_{P_j \in \mathcal{H}}$ are completely random and $\{(\mathbf{C}_{s_j}, \mathbf{com}_{s_j})\}_{P_j \in \mathcal{H}}$ is indistinguishable from $\{(\widetilde{\mathbf{C}}_{\widetilde{s}_j}, \widetilde{\mathbf{com}}_{\widetilde{s}_j})\}_{P_j \in \mathcal{H}}$ for the $\mathcal{A}$. Since the polynomial $f(\cdot)$ is picked uniformly at random in both the executions, and the role of $\mathcal{F}_{\text{Acast}}$ and $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoCSS} is played by the simulator itself, it follows from the properties of t -degree polynomials over $\mathbb{F}$ [72] that the distribution of shares seen by $\mathcal{A}$ in both the executions are identical, except with negligible probability, thus proving the claim. $\square$

We next claim that if the dealer is honest, then conditioned on the view of the adversary (which is identical in both the executions as per the previous claim), the outputs of the honest parties are identically distributed in the real world as well as in the ideal world, except with negligible probability.

Claim 5.12. *If D is honest, then conditioned on the view of $\mathcal{A}$, the output of the honest parties during the execution of Π_{SKVSS} involving $\mathcal{A}$ has the same distribution as the output of the honest parties in the ideal world involving $\mathcal{S}_{\text{SKVSS}}$ and $\mathcal{F}_{\text{VSS}}$ except with negligible probability.*

Proof. Let $D \notin \mathcal{C}$ and let view be an arbitrary view of $\mathcal{A}$. Let $\{f(i)\}_{P_i \in \mathcal{C}}$ be the shares of corrupt parties as per view . Observe that $f(\cdot)$ selected by D is uniformly distributed, conditioned on the shares $\{f(i)\}_{P_i \in \mathcal{C}}$ and $f(0)$; in fact, if $|\mathcal{C}| = t$, then $f(\cdot)$ is uniquely determined by $\{f(i)\}_{P_i \in \mathcal{C}}$ together with $f(0)$. Let us fix any such polynomial $f(\cdot)$. In the ideal world, each honest P_i obtains the request-based delayed output $f(i)$ from $\mathcal{F}_{\text{VSS}}$. We now show that each honest P_i eventually outputs the same $f(i)$ in the real world as well, except with negligible probability. Since D remains honest throughout the execution of Π_{SKVSS} , it would have indeed encrypted $s_i = f(i)$ in $\mathbf{C}_{s_i}$ according to the protocol and would have been broadcast along with other values that P_i will eventually receive along with the correct zero knowledge proof, except with negligible probability, and P_i decrypts $\mathbf{C}_{s_i}$ to get $f(i)$ as its output. $\square$

We next prove claims with respect to a *corrupt* dealer. First we show that the view generated by $\mathcal{S}_{\text{SKVSS}}$ for $\mathcal{A}$ is indistinguishable from $\mathcal{A}$'s view during the real execution of Π_{SKVSS} except with negligible probability.

Claim 5.13. *If the dealer is corrupt, then the view of $\mathcal{A}$ in the simulated execution of Π_{SKVSS} with $\mathcal{S}_{\text{SKVSS}}$ is identically distributed as the view of $\mathcal{A}$ in the real execution of Π_{SKVSS} involving honest parties.*

Proof. If $D \in \mathcal{C}$, then it is easy to see that the view of $\mathcal{A}$ is identically distributed in both executions. This is because in both executions, $\mathcal{A}$ does not receive messages from the honest parties, and in the simulated execution, the simulator itself simulated the steps of $\mathcal{F}_{\text{Setup}}$, $\mathcal{F}_{\text{Acast}}$, and $\mathcal{F}_{\text{ZK}}^{1:M}$. $\square$

We next claim that if the dealer is corrupt, then conditioned on the view of the adversary (which is identical in both the executions as per the previous claim), the outputs of the honest parties are identically distributed in the real world as well as in the ideal world, except with negligible probability.

Claim 5.14. *If D is corrupt, then conditioned on the view of $\mathcal{A}$, the output of the honest parties during the execution of Π_{SKVSS} involving $\mathcal{A}$ has the same distribution as the output of the honest parties in the ideal world involving $\mathcal{S}_{\text{SKVSS}}$ and $\mathcal{F}_{\text{VSS}}$ except with negligible probability.*

Proof. Let $D \in \mathcal{C}$ be corrupt and let view be an arbitrary view of $\mathcal{A}$. We have the following two cases.

- *As per view, dealer did not 'correctly' follow the protocol:* This case corresponds to an execution in which the dealer either fails to send the required messages to $\mathcal{F}_{\text{Acast}}$ or $\mathcal{F}_{\text{ZK}}^{1:M}$, or sends inconsistent values to $\mathcal{F}_{\text{ZK}}^{1:M}$, resulting in the failure of the zero-knowledge proof. In such a real-world execution, honest parties will not receive the messages they are supposed to receive from $\mathcal{F}_{\text{Acast}}$ or $\mathcal{F}_{\text{ZK}}^{1:M}$, and consequently will not obtain any output. Correspondingly, in the ideal world, the simulator $\mathcal{S}_{\text{SKVSS}}$ does not provide any input to $\mathcal{F}_{\text{VSS}}$ on behalf of the D , and hence $\mathcal{F}_{\text{VSS}}$ does not generate any output for the honest parties.
- *As per view, dealer 'correctly' follows the protocol:* This case corresponds to an execution in which the dealer sends all required messages to $\mathcal{F}_{\text{Acast}}$ and $\mathcal{F}_{\text{ZK}}^{1:M}$. Then, according to view, shares of parties lie on a t -degree polynomial $f(\cdot)$. In the ideal world, the simulator forwards this $f(\cdot)$ to $\mathcal{F}_{\text{VSS}}$, and hence every honest party P_i eventually outputs $f(i)$. Correspondingly, in the real-world execution, each honest party P_i will receive messages from $\mathcal{F}_{\text{Acast}}$ and $\mathcal{F}_{\text{ZK}}^{1:M}$. By the security guarantees of the encryption scheme, commitment scheme, and $\mathcal{F}_{\text{ZK}}^{1:M}$, the value $f(i)$ is correctly encrypted as $\mathbf{C}_{s_i}$, except with negligible probability. Each honest party P_i eventually decrypts this ciphertext to obtain $f(i)$ as its output, except with negligible probability.

$\square$

The lemma now follows from Claims 5.11-5.14. $\square$

6 Setting with Setup and Public-key Cryptography

In this section, we present an n -party asynchronous MPC protocol tolerating a computationally bounded Byzantine adversary corrupting t parties, where $n = 2t + 1$. The protocol uses public-key primitives and assumes that the parties have access to a trusted setup for using those primitives, along with access to the $\mathcal{F}_{\text{BC}}$ functionality. Throughout this section, we assume that the adversary is a PPT adversary and κ is the computational security parameter. We assume that there exists a publicly known function $f : \mathbb{F}^n \rightarrow \mathbb{F}$, where $y = f(x_1, \dots, x_n)$, with every party P_i having the input x_i . Moreover, f is represented by a publicly known arithmetic circuit ckt over $\mathbb{F}$, consisting of c_M multiplication gates.

6.1 Cryptographic Primitives Deployed

We first review the existing primitives deployed in the protocol.

Threshold Homomorphic Encryption (THE): It is a form of public-key encryption scheme where the encryption key is available publicly, but decryption key is secret-shared in such a way that decrypting a cipher-text requires the participation of at least $t + 1$ parties. Given an encryption key ek , any party

can compute the encryption $\mathbf{c}_x$ of any plain-text x (under appropriate randomness) using algorithm Enc . Given a cipher-text $\mathbf{c}_x$ and a key dk_i for distributed decryption, party P_i can compute a decryption share x_i of $\mathbf{c}_x$ using algorithm ShareDec . Moreover, given at least $t + 1$ correct decryption shares of $\mathbf{c}_x$, they can be combined using algorithm ShareCombine to obtain a plain-text x . We assume that each cipher-text and decryption-share consists of $\mathcal{O}(\kappa)$ bits.

We assume that the scheme is linearly homomorphic. Namely, given cipher-texts $\mathbf{c}_x = \text{Enc}_{\text{ek}}(x, \star)$ and $\mathbf{c}_y = \text{Enc}_{\text{ek}}(y, \star)$, there exists some operation on cipher-texts, say $\boxplus$ and $\boxminus$, such that $\mathbf{c}_x \boxplus \mathbf{c}_y = \text{Enc}_{\text{ek}}(x + y, \star)$ and $\mathbf{c}_x \boxminus \mathbf{c}_y = \text{Enc}_{\text{ek}}(x - y, \star)$. Likewise, given a public constant $c \in \mathbb{F}$, there exists an operation $\boxtimes$, such that $c \boxtimes \mathbf{c}_x = \text{Enc}_{\text{ek}}(c \cdot x, \star)$ and $c \boxplus \mathbf{c}_x = \text{Enc}_{\text{ek}}(c + x, \star)$. We require the standard *semantic-security* and *correctness* guarantees from the scheme.

Definition 6.1 (Threshold Homomorphic Encryption (THE)). A public key threshold homomorphic encryption (THE) scheme with the underlying message space $\mathcal{M} = \mathbb{F}$, encryption key space $\mathcal{EK}$, decryption key space $\mathcal{DK}$, ciphertext space $\mathcal{CT}$ and distributed decryption key space $\mathcal{DK}_i$ for $i = 1, \dots, n$, is a collection of the following PPT algorithms, parametrized by a computational security parameter κ .

1. $\text{THEKeyGen}(1^\kappa, n, t) \rightarrow (\text{ek}, \text{dk}, \text{dk}_1, \dots, \text{dk}_n)$: The key-generation algorithm outputs an encryption key $\text{ek} \in \mathcal{EK}$, a decryption key $\text{dk} \in \mathcal{DK}$ and n keys $\text{dk}_1, \dots, \text{dk}_n$ for distributed decryption, where $\text{dk}_i \in \mathcal{DK}_i$, for $i = 1, \dots, n$.
2. $\text{Enc}_{\text{ek}}(m, r) \rightarrow \mathbf{c}_m$: The encryption algorithm computes a ciphertext $\mathbf{c}_m \in \mathcal{CT}$, which encrypts a plaintext $m \in \mathcal{M}$ under the encryption key ek , using the randomness r and outputs $\mathbf{c}_m$, consisting of $\mathcal{O}(\kappa)$ bits.¹²
3. $\text{Dec}_{\text{dk}}(\mathbf{c}_x) := x$: The decryption algorithm decrypts a ciphertext $\mathbf{c}_x \in \mathcal{CT}$, using the decryption key dk and outputs a plaintext $x \in \mathcal{M}$.
4. $\text{ShareDec}_{\text{dk}_i}(\mathbf{c}_x) := x_i$: The share decryption algorithm takes input a ciphertext $\mathbf{c}_x$ and a key dk_i for distributed decryption and outputs a decryption share x_i of $\mathbf{c}_x$, consisting of $\mathcal{O}(\kappa)$ bits.
5. $\text{ShareCombine}(\mathbf{c}_x, \{x_i\}_{P_i \in \mathcal{D}}) := x$: The share combine algorithm takes input a ciphertext $\mathbf{c}_x$ and a set of decryption shares corresponding to the parties in $\mathcal{D}$ where $|\mathcal{D}| \geq t + 1$ and outputs a plaintext $x \in \mathcal{M}$.

We require the following homomorphic properties to be satisfied by a THE scheme.

- *Addition*: given cipher-texts $\mathbf{c}_x = \text{Enc}_{\text{ek}}(x, \star)$ and $\mathbf{c}_y = \text{Enc}_{\text{ek}}(y, \star)$, there exists some operation on cipher-texts, say $\boxplus$, such that $\mathbf{c}_x \boxplus \mathbf{c}_y = \text{Enc}_{\text{ek}}(x + y, \star)$, where $\mathbf{c}_x \boxplus \mathbf{c}_y$ is of size $\mathcal{O}(\kappa)$ bits.
- *Subtraction*: given cipher-texts $\mathbf{c}_x = \text{Enc}_{\text{ek}}(x, \star)$ and $\mathbf{c}_y = \text{Enc}_{\text{ek}}(y, \star)$, there exists some operation on cipher-texts, say $\boxminus$, such that $\mathbf{c}_x \boxminus \mathbf{c}_y = \text{Enc}_{\text{ek}}(x - y, \star)$, where $\mathbf{c}_x \boxminus \mathbf{c}_y$ is of size $\mathcal{O}(\kappa)$ bits.
- *Multiplication by a public constant*: Given a publicly known constant $c \in \mathbb{Z}_p$ and cipher-text $\mathbf{c}_x = \text{Enc}_{\text{ek}}(x, \star)$, there exists an operation $\boxtimes$ such that $c \boxtimes \mathbf{c}_x = \text{Enc}_{\text{ek}}(c \cdot x, \star)$.
- *Addition by a public constant*: Given a publicly known constant $c \in \mathbb{Z}_p$ and cipher-text $\mathbf{c}_x = \text{Enc}_{\text{ek}}(x, \star)$, the operation $c \boxplus \mathbf{c}_x$ results in $\text{Enc}_{\text{ek}}(c + x, \star)$.

The homomorphic properties of the cryptosystem allow the parties to homomorphically compute any linear function of the underlying plaintexts. More specifically, given $\mathbf{c}_{x^{(1)}} = \text{Enc}_{\text{ek}}(x^{(1)}, \star), \dots, \mathbf{c}_{x^{(L)}} = \text{Enc}_{\text{ek}}(x^{(L)}, \star)$ and publicly-known constants $c_1, \dots, c_L \in \mathbb{Z}_p$, then the parties can locally compute $\mathbf{c}_y = \text{Enc}_{\text{ek}}(y, \star)$, where $y \stackrel{\text{def}}{=} c_1 \cdot x^{(1)} + \dots + c_L \cdot x^{(L)}$.

We require the following security properties from a THE scheme.

- *Key Generation Security*: Let S and D_i be the random variables which denote the probability distribution with which the decryption key dk and the i^{th} key dk_i for the distributed decryption is selected from $\mathcal{DK}$ and $\mathcal{DK}_i$ by THEKeyGen , for $i = 1, \dots, n$. Moreover, for a set $I \subseteq \{1, \dots, n\}$, let D_I denote the random variable which denote the probability distribution with which the set of keys for the distributed decryption, belonging to the indices in I , are selected from the corresponding space $\mathcal{DK}_i$ by THEKeyGen . Then the following two properties hold.
 - *Correctness*: For any set $I \subseteq \{1, \dots, n\}$ with $|I| \geq t + 1$, the condition $H(S|D_I) = 0$ holds. Here $H(X|Y)$ denotes the conditional entropy of a random variable X with respect to a random variable Y [29].

¹² In the rest of the paper, unless it is explicitly specified, we assume that some randomness has been used for encryption. The notation $\mathbf{c}_x$ will represent an encryption of plaintext m .

- *Privacy*: For any set $I \subset \{1, \dots, n\}$ with $|I| \leq t$, the condition $H(S|D_I) = H(S)$ holds.
- *Semantic Security*: For any $I \subset \{1, \dots, n\}$ with $|I| \leq t$ and all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following experiment is negligible in κ .
 - *Key Generation*: The challenger runs $\text{THEKeyGen}(1^\kappa, n, t)$ to obtain $(\text{ek}, \text{dk}, \text{dk}_1, \dots, \text{dk}_n)$ and sends ek and $\{\text{dk}_i\}_{i \in I}$ to $\mathcal{A}$.
 - *Challenge*: $\mathcal{A}$ sends plain-texts $m_0, m_1 \in \mathcal{M}$ to the challenger, who randomly selects $b \in \{0, 1\}$, computes $\mathbf{c}_{m_b} = \text{Enc}_{\text{ek}}(m_b, r)$ for some randomness r and sends $\mathbf{c}_{m_b}$ to $\mathcal{A}$.
 - *Output*: The adversary $\mathcal{A}$ outputs $b' \in \{0, 1\}$.

The advantage of $\mathcal{A}$ in the above experiment is defined to be $|\frac{1}{2} - \Pr(b' = b)|$.

- *Correctness*: For any $(\text{ek}, \text{dk}, \text{dk}_1, \dots, \text{dk}_n)$ obtained as the output of the algorithm THEKeyGen , the following should hold:

$$\text{Dec}_{\text{dk}}(\text{Enc}_{\text{ek}}(m)) = m.$$

- *Correct Share Decryption*: For any $(\text{ek}, \text{dk}, \text{dk}_1, \dots, \text{dk}_n)$ obtained as the output of THEKeyGen and any $\mathbf{c}_x \in \mathcal{CT}$, let $x_i := \text{ShareDec}_{\text{dk}_i}(\mathbf{c}_x)$, for $i = 1, \dots, n$. Then $(x_1, \dots, x_n)$ should form a (n, t) -threshold secret-sharing of $\text{Dec}_{\text{dk}}(\mathbf{c}_x)$. That is, for any $I \subset \{1, \dots, n\}$ where $|I| \geq t + 1$, the following should hold:

$$\text{Dec}_{\text{dk}}(\mathbf{c}_x) = \text{ShareCombine}(\mathbf{c}_x, \{\text{ShareDec}_{\text{dk}_i}(\mathbf{c}_x)\}_{i \in I}).$$

Threshold Signatures (TSIG): It is a threshold variant of a digital signature scheme. Here, the signing key is secret-shared among the parties with threshold t , such that signing any message requires the participation of at least $t + 1$ parties. Given a message m and a key sk_i for distributed signing, party P_i can run the share-signing algorithm ShareSign and compute a signature-share $\sigma_{m,i}$ consisting of $\mathcal{O}(\kappa)$ bits. Given a message m and a set of signature-shares $\{\sigma_{m,i}\}_{P_i \in \mathcal{D}}$ corresponding to the parties in $\mathcal{D}$ where $|\mathcal{D}| \geq t + 1$, any party can run the signature-share combine algorithm SigCombine and compute a signature σ_m on m , consisting of $\mathcal{O}(\kappa)$ bits. Given a message m and a supposed signature σ_m on m , any party can run the signature-verification algorithm SigVer using the publicly known verification key vk , which outputs either 0 (reject) or 1 (accept). We require the standard *correctness* and *unforgeability* guarantees from the scheme.

Definition 6.2 (Threshold Signatures (TSIG)). A threshold signature (TSIG) scheme with the underlying message space $\mathcal{M}$, signing key space $\mathcal{SK}$, verification key space $\mathcal{VK}$, signature space $\mathcal{SIG}$ and distributed signing key space $\mathcal{SK}_i$, for $i = 1, \dots, n$, is a collection of the following PPT algorithms, parametrized by a computational security parameter κ .

1. $\text{TSIGKeyGen}(1^\kappa, n, t) \rightarrow (\text{vk}, \text{sk}, \text{sk}_1, \dots, \text{sk}_n)$: The key-generation algorithm outputs a verification key $\text{vk} \in \mathcal{VK}$, a signing key $\text{sk} \in \mathcal{SK}$ and n keys $\text{sk}_1, \dots, \text{sk}_n$ for distributed signing, where $\text{sk}_i \in \mathcal{SK}_i$, for $i = 1, \dots, n$.
2. $\text{Sign}_{\text{sk}}(m) := \sigma_m$: The signing algorithm takes input a message $m \in \mathcal{M}$ for signing and the signing key sk and outputs a signature σ_m on m consisting of $\mathcal{O}(\kappa)$ bits, where $\sigma_m \in \mathcal{SIG}$.
3. $\text{ShareSign}_{\text{sk}_i}(m) := \sigma_{m,i}$: The share-signing algorithm takes input a message $m \in \mathcal{M}$ and a key sk_i for distributed signing and outputs a signature-share $\sigma_{m,i}$ consisting of $\mathcal{O}(\kappa)$ bits.
4. $\text{SigCombine}(m, \{\sigma_{m,i}\}_{P_i \in \mathcal{D}}) := \sigma_m$: The signature-share combine algorithm takes input a message $m \in \mathcal{M}$ for signing and a set of signature-shares corresponding to the parties in $\mathcal{D}$ where $|\mathcal{D}| \geq t + 1$ and outputs a signature $\sigma_m \in \mathcal{SIG}$.
5. $\text{SigVer}_{\text{vk}}(m, \sigma_m) \rightarrow \{0, 1\}$: The signature-verification algorithm takes input a message $m \in \mathcal{M}$ and a signature σ_m on m for verification, along with the verification key vk and outputs either 0 (reject) or 1 (accept).

We require the following security properties from a threshold signature scheme.

- *Key Generation Security*: Let S and D_i be the random variables which denote the probability distribution with which the signing key sk and the i^{th} key sk_i for the distributed signing is selected from $\mathcal{SK}$ and $\mathcal{SK}_i$ by TSIGKeyGen , for $i = 1, \dots, n$. Moreover, for a set $I \subseteq \{1, \dots, n\}$, let D_I denote the random variable which denote the probability distribution with which the set of keys for the distributed signing, belonging to the indices in I , are selected from the corresponding space $\mathcal{SK}_i$ by TSIGKeyGen . Then the following two properties hold.

- *Correctness*: For any set $I \subseteq \{1, \dots, n\}$ with $|I| \geq t + 1$, the condition $H(S|D_I) = 0$ holds. Here $H(X|Y)$ denotes the conditional entropy of a random variable X with respect to a random variable Y .
- *Privacy*: For any set $I \subseteq \{1, \dots, n\}$ with $|I| \leq t$, the condition $H(S|D_I) = H(S)$ holds.
- *Correctness*: For any $(vk, sk, sk_1, \dots, sk_n)$ obtained as the output of the algorithm TSIGKeyGen , the following should hold:

$$\text{SigVer}_{vk}(m, \text{Sign}_{sk}(m)) = 1.$$

- *Correct Share Signing*: For any $(vk, sk, sk_1, \dots, sk_n)$ obtained as the output of TSIGKeyGen and any $\mathcal{D} \subseteq \{P_1, \dots, P_n\}$ where $|\mathcal{D}| \geq t + 1$, the following should hold for any signature $\sigma_m \in \text{SIG}$:

$$\text{SigVer}_{vk}(m, \sigma_m) = \text{SigVer}_{vk}(m, \text{SigCombine}(m, \{\sigma_{m,i}\}_{P_i \in \mathcal{D}})).$$

- *Unforgeability*: The scheme should be unforgeable against adaptive chosen message attack (CMA) when the adversary is given up to t keys for distributed signing. This is formalized by the threshold version of the CMA security experiment (see, for instance, [30]).

Homomorphic Commitments: We require the commitment scheme of Definition 5.5 to additionally be homomorphic. Namely, given commitments $\text{com}_x = \text{Commit}(x, r_x)$ and $\text{com}_y = \text{Commit}(y, r_y)$, there exist operations $\oplus, \ominus$ and $\odot$, such that $\text{com}_x \oplus \text{com}_y = \text{Commit}(x + y, r_x + r_y)$, $\text{com}_x \ominus \text{com}_y = \text{Commit}(x - y, r_x - r_y)$, $c \odot \text{com}_x = \text{Commit}(c \cdot x, c \cdot r_x)$ and $c \oplus \text{com}_x = \text{Commit}(c + x, c + r_x)$, where $c \in \mathbb{F}$ is a publicly known constant.

Definition 6.3 (Homomorphic Commitment). A homomorphic commitment scheme is a commitment scheme, as per Definition 5.5, with underlying message space $\mathcal{M} = \mathbb{F}$, that additionally satisfies the following properties.

- *Addition*: given commitments $\text{com}_x = \text{Commit}(x, r_x)$ and $\text{com}_y = \text{Commit}(y, r_y)$, there exists an operation, say $\oplus$, such that $\text{com}_x \oplus \text{com}_y = \text{Commit}(x + y, r_x + r_y)$.
- *Subtraction*: given commitments $\text{com}_x = \text{Commit}(x, r_x)$ and $\text{com}_y = \text{Commit}(y, r_y)$, there exists an operation, say $\ominus$, such that $\text{com}_x \ominus \text{com}_y = \text{Commit}(x - y, r_x - r_y)$.
- *Multiplication by a public constant*: given any publicly known constant $c \in \mathcal{M}$ and a commitment com_x , there exists an operation, say $\odot$, such that $c \odot \text{com}_x = \text{Commit}(c \cdot x, c \cdot r_x)$.
- *Addition by a public constant*: given any publicly known constant $c \in \mathcal{M}$ and a commitment com_x , the operation $c \oplus \text{com}_x$ results in $\text{Commit}(c + x, c + r_x)$.

We require that performing homomorphic operations on commitments results in commitments of size $\mathcal{O}(\kappa)$ bits.

Polynomial Commitment: It allows a committer to commit to a polynomial with a short string (independent of the size of the polynomial) that can be used by a verifier to confirm claimed evaluations of the committed polynomial. Given the public parameters params and a t -degree polynomial $f(X)$, any party can run the polynomial-commitment algorithm Commit and compute a commitment $\text{PolyCom}_{f(X)}$ of $f(X)$ of size $\mathcal{O}(\kappa)$ bits. Given $f(X)$ and $\text{PolyCom}_{f(X)}$, any party can run the witness-creation algorithm CreateWitness and compute a witness w_i for the evaluation $f(i)$ of $f(X)$ at the index i , where w_i is of size $\mathcal{O}(\kappa)$ bits. Finally, given $\text{PolyCom}_{f(X)}$, i , $f(i)$, w_i , any party can run the evaluation-point verification algorithm VerifyEval , which outputs either 0 (reject) or 1 (accept). We require the standard *polynomial binding* and *evaluation binding* properties.

Definition 6.4 (Polynomial Commitment). A polynomial commitment scheme with the underlying message space $\mathcal{M} = \mathbb{F}$, is a collection of the following PPT algorithms, parametrized by a security parameter κ .¹³

- $\text{PolynomialCommitmentGen}(1^\kappa, t) \rightarrow (\text{params})$: the parameter generation algorithm outputs public parameters params , to commit to polynomials of degree up to t with coefficients over $\mathcal{M}$.

¹³ As defined in [54], a generic polynomial commitment scheme may consist of additional algorithms; we specify only the algorithms needed for our protocols.

- $\text{Commit}(\text{params}, f(X)) \rightarrow \text{PolyCom}_{f(X)}$: the commit algorithm takes input the public parameters params , a polynomial $f(X)$ over $\mathcal{M}$ of degree up to t and outputs a commitment $\text{PolyCom}_{f(X)}$ of $f(X)$ of size $\mathcal{O}(\kappa)$ bits.
- $\text{VerifyPoly}(\text{params}, \text{PolyCom}_{f(X)}, f(X)) \rightarrow \{0, 1\}$: the polynomial-verification algorithm takes input the public parameters params , a polynomial $f(X)$ over $\mathcal{M}$ of degree up to t and a polynomial commitment $\text{PolyCom}_{f(X)}$ and outputs either 0 (reject) or 1 (accept).
- $\text{CreateWitness}(\text{params}, \text{PolyCom}_{f(X)}, f(X), i) \rightarrow w_i$: the witness-creation algorithm takes input the public parameters params , polynomial $f(X)$ over $\mathcal{M}$ of degree up to t and a polynomial commitment $\text{PolyCom}_{f(X)}$ and outputs a witness w_i for the evaluation $f(i)$ of $f(X)$ at the index i , where w_i is of size $\mathcal{O}(\kappa)$ bits.
- $\text{VerifyEval}(\text{params}, \text{PolyCom}_{f(X)}, i, f(i), w_i) \rightarrow \{0, 1\}$: the evaluation verification algorithm takes input the public parameters params , a polynomial commitment $\text{PolyCom}_{f(X)}$, index i , an evaluation $f(i)$ and a witness w_i and outputs either 0 (reject) or 1 (accept).

We require the following security properties from a polynomial commitment scheme.¹⁴

- *Correctness*: For every params obtained by running $\text{PolynomialCommitmentGen}$, every polynomial $f(X)$ over $\mathcal{M}$ of degree up to t and every evaluation index i , the following should hold.
 - The output of $\text{VerifyPoly}(\text{params}, \text{Commit}(\text{params}, f(X)), f(X))$ is 1.
 - The output of the algorithm $\text{VerifyEval}(\text{params}, \text{Commit}(\text{params}, f(X)), i, f(i), \text{CreateWitness}(\text{params}, \text{Commit}(\text{params}, f(X)), f(X), i))$ is 1.
- *Polynomial Binding*: For every PPT adversary $\mathcal{A}$, the advantage of the $\mathcal{A}$ in the following experiment should be negligible in κ .
 - *Parameter generation*: The challenger runs $\text{PolynomialCommitmentGen}(1^\kappa)$ to obtain params and sends params to $\mathcal{A}$.
 - *Output*: $\mathcal{A}$ outputs $\text{PolyCom}_{f(X)}, f(X)$ and $f'(X)$, where $f(X)$ and $f'(X)$ are polynomials of degree up to t over $\mathcal{M}$ such that $f(X) \neq f'(X)$.

The advantage of $\mathcal{A}$ in the above experiment is defined to be:

$$\Pr \left[\text{VerifyPoly}(\text{params}, \text{PolyCom}_{f(X)}, f(X)) = 1 \wedge \text{VerifyPoly}(\text{params}, \text{PolyCom}_{f(X)}, f'(X)) = 1 \right].$$

- *Evaluation Binding*: For every PPT adversary $\mathcal{A}$, the advantage of the $\mathcal{A}$ in the following experiment should be negligible in κ .
 - *Parameter generation*: The challenger runs $\text{PolynomialCommitmentGen}(1^\kappa)$ to obtain params and sends params to $\mathcal{A}$.
 - *Output*: $\mathcal{A}$ outputs $\text{PolyCom}_{f(X)}, i, y_i, y'_i, w_i, w'_i$ such that $y_i \neq y'_i$.

The advantage of $\mathcal{A}$ in the above experiment is defined to be:

$$\Pr \left[\text{VerifyEval}(\text{params}, \text{PolyCom}_{f(X)}, i, y_i, w_i) = 1 \wedge \text{VerifyEval}(\text{params}, \text{PolyCom}_{f(X)}, i, y'_i, w'_i) = 1 \right].$$

In the rest of the paper, while using the polynomial commitment scheme, we will assume that the public parameters params have already been generated by a trusted setup and avoid mentioning params explicitly while invoking the underlying algorithms.

ZK Proofs and Certificates: In the zero-knowledge functionality (Functionality 6.5), parameterized by a relation R , the designated prover sends the functionality a statement x to be proven along with a witness w . In response, the functionality forwards the statement x to the designated verifier iff $R(x, w) = 1$ (i.e., if and only if x is a correct statement and w is a witness for x).

Functionality 6.5: $\mathcal{F}_{\text{ZK}}$ – The ideal functionality for zero-knowledge

$\mathcal{F}_{\text{ZK}}$ proceeds as follows, running with a designated prover $P \in \mathcal{P}$, a designated verifier $V \in \mathcal{P}$, an adversary $\mathcal{S}$ and parameterized with a relation R . Upon receiving $(\text{ZK-prover}, \text{sid}, x, w)$ from P , do the following only if $R(x, w) = 1$.

- Send $(\text{ZK-proof}, \text{sid}, x)$ to $\mathcal{S}$.

¹⁴ A generic polynomial commitment scheme has additional security properties; we list only the security properties relevant for our construction.

- Send a request-based delayed output (ZK-proof, sid, x) to V .

We assume the presence of UC-secure ZK protocols for the following relations, with a communication complexity of $\mathcal{O}(\kappa)$ bits. Such protocols can be designed using standard techniques, see for instance [81].

1. **Proof for the equality of encrypted and committed value (PoE).** The statement consists of a commitment $\mathbf{com}_m$, along with encryptions $\mathbf{c}_m, \mathbf{c}_r$ and an encryption key $\mathbf{ek}$. The witness consists of a plain-text m and randomness r, r_1 and r_2 , explaining that m encrypted in $\mathbf{c}_m$ is also committed in $\mathbf{com}_m$, under the randomness r encrypted in $\mathbf{c}_r$. That is,

$$R_{\text{PoE}} = \{(\mathbf{com}_m, \mathbf{c}_m, \mathbf{c}_r, \mathbf{ek}), (m, r, r_1, r_2) \mid \mathbf{com}_m = \text{Commit}(m, r) \wedge \mathbf{c}_m = \text{Enc}_{\mathbf{ek}}(m, r_1) \wedge \mathbf{c}_r = \text{Enc}_{\mathbf{ek}}(r, r_2)\}.$$

2. **Proof for the correct pre-multiplication (PoCM).** The statement consists of $(\mathbf{com}_u, \mathbf{c}_u, \mathbf{c}_{r_u})$, $(\mathbf{com}_v, \mathbf{c}_v, \mathbf{c}_{r_v})$ and $(\mathbf{com}_{u \cdot v}, \mathbf{c}_{u \cdot v}, \mathbf{c}_{r_{u \cdot v}})$, along with an encryption key $\mathbf{ek}$. The witness consists of $(u, r_u, r_1, r_2, \alpha, \beta, \gamma)$, explaining that commitment $\mathbf{com}_{u \cdot v}$ and cipher-texts $\mathbf{c}_{u \cdot v}$ and $\mathbf{c}_{r_{u \cdot v}}$ are generated by “pre-multiplying” $\mathbf{com}_v, \mathbf{c}_v$ and $\mathbf{c}_{r_v}$ respectively with the same u , followed by “rerandomization” with 0, where u is committed in $\mathbf{com}_u$ under the randomness r_u , such that u and r_u are encrypted in $\mathbf{c}_u$ and $\mathbf{c}_{r_u}$. That is,

$$R_{\text{PoCM}} = \left\{ \left((\mathbf{com}_u, \mathbf{c}_u, \mathbf{c}_{r_u}), (\mathbf{com}_v, \mathbf{c}_v, \mathbf{c}_{r_v}), (\mathbf{com}_{u \cdot v}, \mathbf{c}_{u \cdot v}, \mathbf{c}_{r_{u \cdot v}}), (u, r_u, r_1, r_2, \alpha, \beta, \gamma) \mid \right. \right. \\ \left. \mathbf{com}_u = \text{Commit}(u, r_u) \wedge \mathbf{c}_u = \text{Enc}_{\mathbf{ek}}(u, r_1) \wedge \mathbf{c}_{r_u} = \text{Enc}_{\mathbf{ek}}(r_u, r_2) \right. \\ \left. \wedge \mathbf{com}_{u \cdot v} = u \odot \mathbf{com}_v \oplus \text{Commit}(0, \alpha) \wedge \mathbf{c}_{u \cdot v} = u \boxdot \mathbf{c}_v \boxplus \text{Enc}_{\mathbf{ek}}(0, \beta) \right. \\ \left. \wedge \mathbf{c}_{u \cdot r_v + \alpha} = u \boxdot \mathbf{c}_{r_v} \boxplus \text{Enc}_{\mathbf{ek}}(\alpha, \gamma) \right\}$$

3. **Proof for correct decryption (PoCD).** The statement consists of a ciphertext $\mathbf{c}_x$, a decryption share x_i and an encryption key $\mathbf{ek}$. The witness is a key $\mathbf{dk}_i$ for distributed decryption, where $i \in \{1, \dots, n\}$, explaining that x_i is the decryption-share of $\mathbf{c}_x$, computed using $\mathbf{dk}_i$. That is,

$$R_{\text{PoCD}} = \{((\mathbf{ek}, \mathbf{c}_x, x_i), \mathbf{dk}_i) \mid x_i := \text{ShareDec}_{\mathbf{dk}_i}(\mathbf{c}_x)\}.$$

4. **Proof of correct signature (PoCS).** The statement consists of a message m , a signature share $\sigma_{m,i}$ and a verification key $\mathbf{vk}$. The witness consists of a key $\mathbf{sk}_i$ for distributed signing, where $i \in \{1, \dots, n\}$, explaining that $\sigma_{m,i}$ is the signature-share of m , computed using $\mathbf{sk}_i$. That is,

$$R_{\text{PoCS}} = \{((\mathbf{vk}, m, \sigma_{m,i}), \mathbf{sk}_i) \mid \sigma_{m,i} := \text{ShareSign}_{\mathbf{sk}_i}(m)\}.$$

One-to-Many ZK Proofs and Certificates of Claims: Recall the one-to-many extension of ZK functionality $\mathcal{F}_{\text{ZK}}^{1:M}$ (Functionality 5.6), which allows a prover to prove a statement to *all*. The work of [27, 51] has shown the following.

Theorem 6.6 ([27, 51]). *Let $n \geq 2t + 1$, let R be a binary relation and let $\mathcal{F}_{\text{ZK}}$ be an ideal functionality for R , such that there exists a UC-secure protocol for realizing $\mathcal{F}_{\text{ZK}}$ with communication complexity $\mathcal{O}(\kappa)$ bits. Assuming the existence of a TSIG scheme, $\mathcal{F}_{\text{ZK}}^{1:M}$ for R can be securely realized with UC security in the $\mathcal{F}_{\text{ZK}}$ -hybrid model with communication complexity $\mathcal{O}(n\kappa)$ bits.*

6.2 Encrypted-and-Committed Values

In our protocols, computation happens over encrypted-and-committed values, where any $s \in \mathbb{F}$ is said to be encrypted-and-committed, if the following hold.

- Every (honest) party in $\mathcal{P}$ has $\mathbf{com}_s = \text{Commit}(s, r)$, for some r .
- Every (honest) party in $\mathcal{P}$ has $\mathbf{c}_s = \text{Enc}_{\mathbf{ek}}(s, \star)$ and $\mathbf{c}_r = \text{Enc}_{\mathbf{ek}}(r, \star)$.

We denote by $\langle s \rangle$ the triplet $(\mathbf{com}_s, \mathbf{c}_s, \mathbf{c}_r)$. Due to the linearity of THE and commitment, the parties can perform linear operations over encrypted-and-committed values. Namely, let $\langle a \rangle = (\mathbf{com}_a, \mathbf{c}_a, \mathbf{c}_{r_a})$ and $\langle b \rangle = (\mathbf{com}_b, \mathbf{c}_b, \mathbf{c}_{r_b})$. Moreover, let $d \stackrel{\text{def}}{=} c_1 \cdot a + c_2 \cdot b$, where c_1, c_2 are public constants from $\mathbb{F}$. Then, parties can locally compute $\langle d \rangle$, given $\langle a \rangle$ and $\langle b \rangle$. Namely, $\langle d \rangle = (\mathbf{com}_d, \mathbf{c}_d, \mathbf{c}_{r_d})$, where $\mathbf{com}_d = (c_1 \odot \mathbf{com}_a) \oplus (c_2 \odot \mathbf{com}_b)$, $\mathbf{c}_d = (c_1 \boxdot \mathbf{c}_a) \boxplus (c_2 \boxdot \mathbf{c}_b)$ and $\mathbf{c}_{r_d} = (c_1 \boxdot \mathbf{c}_{r_a}) \boxplus (c_2 \boxdot \mathbf{c}_{r_b})$. In the rest of the paper, we say “parties locally compute $\langle d \rangle$ from $\langle a \rangle$ and $\langle b \rangle$ ” to mean the above.

6.3 The Preprocessing Phase Protocol

Our goal is to securely realize the functionality $\mathcal{F}_{\text{Pre}}$ for the preprocessing phase, which generates a *one-time* setup for homomorphic commitment scheme, polynomial commitment scheme, TSIG and THE. The setup can be *reused* across multiple instances of the MPC protocol. Additionally, the functionality also generates c_M number of random encrypted-and-committed multiplication-triples.

Functionality 6.7: $\mathcal{F}_{\text{Pre}}$ – The ideal functionality for preprocessing

The functionality interacts with the parties in $\mathcal{P}$ and the adversary $\mathcal{S}$ as follows.

Setup Generation : On receiving the input **Setup** from at least $t + 1$ parties, the functionality does the following.

- Compute $\text{THEKeyGen}(1^\kappa, n, t) \rightarrow (\text{ek}, \text{dk}, \text{dk}_1, \dots, \text{dk}_n)$.
- Compute $\text{TSIGKeyGen}(1^\kappa, n, t) \rightarrow (\text{vk}, \text{sk}, \text{sk}_1, \dots, \text{sk}_n)$.
- Compute $\text{CommitmentGen}(1^\kappa) \rightarrow \text{params}_1$.
- Compute $\text{PolynomialCommitmentGen}(1^\kappa, t) \rightarrow \text{params}_2$.
- For $i = 1, \dots, n$, send $(\text{vk}, \text{ek}, \text{sk}_i, \text{dk}_i, \text{params}_1, \text{params}_2)$ to party P_i .

Triple Generation : On receiving the input **(sid, Triples)** from at least $t + 1$ parties, the functionality does the following.

- For $l = 1, \dots, c_M$, randomly select $u^{(l)}, v^{(l)} \in \mathbb{F}$ and compute $w^{(l)} = u^{(l)} \cdot v^{(l)}$. Compute $(\langle u^{(l)} \rangle, \langle v^{(l)} \rangle, \langle w^{(l)} \rangle)$ by picking appropriate randomness from the underlying randomness space uniformly at random.
 - For $i = 1, \dots, n$, send $(\text{sid}, \text{Triples}, \{(\langle u^{(l)} \rangle, \langle v^{(l)} \rangle, \langle w^{(l)} \rangle)\}_{l=1, \dots, c_M})$ to P_i .
-

Before presenting our protocol for $\mathcal{F}_{\text{Pre}}$, we first present our ACS protocol.

6.3.1 Agreement on Common Subset (ACS) We present a protocol Π_{ACS} for agreement on a common subset (ACS) in the $\mathcal{F}_{\text{BC}}$ -hybrid model. The high-level idea of the protocol has already been discussed in Section 2.5.

Protocol 6.8: Π_{ACS} — Protocol for agreement on a common subset

Public Information: Boolean function V , such that the output of V is guaranteed to be true for the inputs provided by the honest parties.

Distributing the Input — Each $P_i \in \mathcal{P}$ on having an input $m_i \in \{0, 1\}^L$ does the following.

1. Divide m_i into blocks $m_{i,0}, \dots, m_{i,t}$, each consisting of $\frac{L}{t+1}$ bits.¹⁵ Interpret each of the blocks $m_{i,0}, \dots, m_{i,t}$ as an element over the field $\text{GF}(2^c)$, where $c = \frac{L}{t+1}$ and $|\text{GF}(2^c)| > n$. Compute the polynomial $f_{m_i}(X) \stackrel{\text{def}}{=} m_{i,0} + m_{i,1}X + \dots + m_{i,t}X^t$ and **PolyCom** $_{f_{m_i}(X)} = \text{Commit}(f_{m_i}(X))$. % **Encode the input as a degree- t polynomial. Also commit to the polynomial.**
2. Send **(Input, sid, $f_{m_i}(X)$)** to every party in $\mathcal{P}$. % **Distribute input polynomial to respective parties.**
3. Send **(P_i , Send, sid, **PolyCom** $_{f_{m_i}(X)}$)** to $\mathcal{F}_{\text{BC}}$. % **Publish the polynomial commitment in blockchain.**

Verifying the Inputs Committed Over Blockchain — Every $P_i \in \mathcal{P}$ sends **(P_i , Get, sid)** to $\mathcal{F}_{\text{BC}}$. Upon receiving **TO_i** from $\mathcal{F}_{\text{BC}}$, for every $P_j \in \mathcal{P}$ such that **(P_j , Send, sid, **PolyCom** $_{f_{m_j}(X)}$)** is present in **TO_i**, do the following.

1. Upon receiving **(Input, sid, $f_{m_j}(X)$)** from P_j , compute $\sigma_{1\|j,i} = \text{ShareSign}_{\text{sk}_i}(\text{sid}\|1\|j)$, provided $\text{VerifyPoly}(\text{PolyCom}_{f_{m_j}(X)}, f_{m_j}(X)) = 1$ and $V(m_j) = 1$, where $m_j \in \{0, 1\}^L$ is the input, corresponding to $f_{m_j}(X)$. % **Verify the consistency of the received point.**
2. Send **(sid, $\sigma_{1\|j,i}$)** to P_j and **(ZK-prover, sid_{ij}, ((vk, sid $\|1\|j$, $\sigma_{1\|j,i}$), $\text{sk}_i))$** to $\mathcal{F}_{\text{ZK}}$ for the relation R_{PoCS} , where P_i plays the role of prover and P_j plays the role of verifier and where $\text{sid}_{ij} = \text{sid}\|i\|j$. % **Verify the correctness of the signature shares.**

Publicly Announcing the Delivery of Input — Each $P_j \in \mathcal{P}$ executes the following code.

¹⁵ For simplicity, we assume that L is a multiple of $t + 1$.

1. Upon receiving $(\text{sid}, \sigma_{1||j,i})$ from party P_i , request output from $\mathcal{F}_{\text{ZK}}$ as a verifier where P_i plays the role. Upon receiving $(\text{ZK-proof}, \text{sid}_{ij}, (\text{vk}, 1||j, \sigma_{1||j,i}))$ from $\mathcal{F}_{\text{ZK}}$, include P_i to an accumulative set $\mathcal{D}_j$ (initialized to $\emptyset$). **% Collect approvals from the verifiers.**
2. Once $|\mathcal{D}_j| \geq t + 1$, compute $\sigma_{1||j} = \text{SigCombine}(1, \{\sigma_{1||j,i}\}_{P_i \in \mathcal{D}_j})$. **% Aggregate the approvals into a threshold signature.**
3. Send $(P_j, \text{Send}, \text{sid}, \sigma_{1||j})$ to $\mathcal{F}_{\text{BC}}$. **% Publish the threshold signature in blockchain.**

Identifying the Set of Core Parties — Each $P_i \in \mathcal{P}$ initializes $\mathcal{CR}_i = \emptyset$ and executes the following code, till $|\mathcal{CR}_i| = n - t$. **% Persistence property will ensure that all honest parties agree on the same core set.**

1. Send $(P_i, \text{Get}, \text{sid})$ to $\mathcal{F}_{\text{BC}}$.
2. Upon receiving TO_i from $\mathcal{F}_{\text{BC}}$, include P_j to $\mathcal{CR}_i$, provided $(P_j, \text{Send}, \text{sid}, \sigma_{1||j}) \in \text{TO}_i$ and $\text{SigVer}_{\text{vk}}(1||j, \sigma_{1||j}) = 1$. **% Include the parties having valid signature into the core set.**

Relaying Points on the Polynomials of the Parties in Core Set —

Every $P_i \in \mathcal{P}$ does the following upon computing $\mathcal{CR}_i$ where $|\mathcal{CR}_i| = n - t$.

1. For every $P_j \in \mathcal{CR}_i$, if P_i has sent $(\text{sid}, \sigma_{1||j,i})$ to P_j , then for $k = 1, \dots, n$, compute $m_{j,k,i} = f_{m_j}(k)$ and $w_{j,k,i} = \text{CreateWitness}(\text{PolyCom}_{f_{m_j}(X)}, f_{m_j}(X), k)$. Send $(\text{sid}, P_j, P_k, P_i, m_{j,k,i}, w_{j,k,i})$ to P_k . **% Forward locally verified evaluation to all other parties.**

Computing and Relaying Designated Point on the Polynomials —

Every $P_k \in \mathcal{P}$ executes the following code corresponding to every $P_j \in \mathcal{CR}_k$ upon computing a $\mathcal{CR}_k$ where $|\mathcal{CR}_k| = n - t$.

1. If P_k has sent $(\text{sid}, \sigma_{1||j,k})$ to P_j , then set $m_{j,k} = f_{m_j}(k)$ and $w_{j,k} = \text{CreateWitness}(\text{PolyCom}_{f_{m_j}(X)}, f_{m_j}(X), k)$.
2. Else upon receiving $(\text{sid}, P_j, P_k, P_i, m_{j,k,i}, w_{j,k,i})$ from P_i , set $m_{j,k} = m_{j,k,i}$ and $w_{j,k} = w_{j,k,i}$, provided $\text{VerifyEval}(\text{PolyCom}_{f_{m_j}(X)}, k, m_{j,k,i}, w_{j,k,i}) = 1$. **% If there are several P_i for which the condition of this step is true, then consider only the least-indexed P_i .**

Send $(\text{sid}, P_j, P_k, m_{j,k}, w_{j,k})$ to every party in $\mathcal{P}$.

Output Computation — Every $P_i \in \mathcal{P}$ executes the following code for every $P_j \in \mathcal{CR}_i$, upon computing a set $\mathcal{CR}_i$ where $|\mathcal{CR}_i| = n - t$.

1. Upon receiving $(\text{sid}, P_j, P_k, m_{j,k}, w_{j,k})$ from any P_k , include P_k to an accumulative set $\mathcal{E}_{j,i}$ (initialized to $\emptyset$), provided $\text{VerifyEval}(\text{PolyCom}_{f_{m_j}(X)}, k, m_{j,k}, w_{j,k}) = 1$.
2. Once $|\mathcal{E}_{j,i}| = t + 1$, interpolate a t -degree polynomial through the points $\{(k, m_{j,k})\}_{P_k \in \mathcal{E}_{j,i}}$. Interpret each coefficient of the polynomial as a block of $\frac{L}{t+1}$ bits and set m_j to be the concatenation of these blocks.

Upon computing m_j for every $P_j \in \mathcal{CR}_i$, output $\{m_j\}_{P_j \in \mathcal{CR}_i}$ and $\mathcal{CR}_i$.

Lemma 6.9. *Let V be a Boolean function and let each $P_i \in \mathcal{P}$ has input $m_i \in \{0, 1\}^L$ for Π_{ACS} , where $V(m_i) = 1$ for each honest P_i . Assuming the existence of threshold signature and polynomial commitment schemes, protocol Π_{ACS} achieves the following in $\mathcal{F}_{\text{BC}}$ -hybrid model except with a negligible probability, incurring a communication of $\mathbb{P}(n^2L + n^3\kappa) + \mathbb{B}(n\kappa)$ bits, provided $L \geq (t + 1) \log n$.*

- All honest parties agree upon a common subset $\mathcal{CR} \subset \mathcal{P}$, where $|\mathcal{CR}| = n - t$.
- For every $P_j \in \mathcal{CR}$, there exists an $m_j^* \in \{0, 1\}^L$ where $V(m_j^*) = 1$ and where $m_j^* = m_j$ for an honest P_j , such that all honest parties output $\{m_j^*\}_{P_j \in \mathcal{CR}}$.

Proof. The agreement upon a common subset $\mathcal{CR} \subset \mathcal{P}$ directly follows from the properties of the ideal functionality $\mathcal{F}_{\text{BC}}$. In the protocol Π_{ACS} , every honest party P_i determines its local set $\mathcal{CR}_i$ by selecting the first $n - t$ parties whose valid threshold signature $\sigma_{1||j}$ appears on its local ledger TO_i . Since $\mathcal{F}_{\text{BC}}$ guarantees *persistence* (i.e., the local ledgers of honest parties are prefix-consistent), all honest parties will identify the exact same set of $n - t$ indices in the same order. Moreover, due to the *liveness* of $\mathcal{F}_{\text{BC}}$, any valid signature sent by an honest party is eventually added to the ledger, ensuring that the set $\mathcal{CR}$ is eventually formed. Thus, all honest parties agree on a common subset $\mathcal{CR}$.

Consider any $P_j \in \mathcal{CR}$. By the protocol description of Π_{ACS} , P_j is included in $\mathcal{CR}$ only if a valid signature $\sigma_{1||j}$ is recorded in the blockchain. A valid signature $\sigma_{1||j}$ requires aggregation of at least $t + 1$ valid signature-shares $\{\sigma_{1||j}\}$. Since there are at most t corrupt parties, at least one signature-share must have been generated by an honest party P_k . An honest P_k issues a signature share only if it successfully verifies that the committed polynomial $f_{m_j}(X)$ corresponds to an input m_j satisfying the predicate $V(m_j) = 1$. Thus, for every $P_j \in \mathcal{CR}$, there exists a valid input $m_j^* \in \{0, 1\}^L$ such that $V(m_j^*) = 1$.

It remains to show that every honest party eventually outputs the correct input m_j for every $P_j \in \mathcal{CR}$. As established above, for any $P_j \in \mathcal{CR}$, there exists at least one honest party P_k who has verified P_j 's input and thus possesses the polynomial $f_{m_j}(X)$. In the relaying phase, every such honest party P_k computes the point $m_{j,h,k} = f_{m_j}(h)$ and a witness $w_{j,h,k}$ for every other party P_h and sends it to them. Consequently, every honest party P_h will eventually receive at least one valid point-witness pair corresponding to its index h on the polynomial $f_{m_j}(X)$. Upon receiving this, P_h broadcasts this verified point and witness to all other parties. Since there are at least $t + 1$ honest parties, every honest party will eventually receive at least $t + 1$ verified points on the polynomial $f_{m_j}(X)$. This allows them to interpolate the unique t -degree polynomial and reconstruct the input m_j .

In the protocol, each party sends its input to every party, which incurs a communication of $\mathbb{P}(n^2L)$ bits. In addition, each party makes public a polynomial commitment over the blockchain, which incurs a total communication of $\mathbb{B}(n\kappa)$ bits. Note that we require $L \geq (t + 1) \log n$, since we require $|\mathbb{F}| > n$ and we also require L to be a multiple of $t + 1$. In the protocol, each party sends a signature-share to every other party, which incurs a total communication of $\mathbb{P}(n^2\kappa)$ bits. Moreover, each party makes public a signature on the message “1” over the blockchain, which incurs a total communication of $\mathbb{B}(n\kappa)$ bits. Finally, every party may send up to n evaluation-points of size $\frac{L}{t+1}$ bits each and a witness of size $\mathcal{O}(\kappa)$ bits each to every other party. This incurs a total communication of $\mathbb{P}(n^2 \cdot \frac{nL}{t+1} + n^3\kappa) = \mathbb{P}(n^2L + n^3\kappa)$ bits. $\square$

6.3.2 Ideal Functionality for One-Time Setup. To securely realize $\mathcal{F}_{\text{Pre}}$, we assume an ideal functionality $\mathcal{F}_{\text{Setup}}$ which generates a one-time setup for THE, TSIG, polynomial commitment, and commitment scheme.

Functionality 6.10: $\mathcal{F}_{\text{Setup}}$ – The ideal functionality for one-time setup

$\mathcal{F}_{\text{Setup}}$ proceeds as follows, running with parties $\mathcal{P}$ and an adversary $\mathcal{S}$.

Setup Generation : On receiving `Init` from at least $t + 1$ parties, the functionality does the following.

- Compute $\text{THEKeyGen}(1^\kappa, n, t) \rightarrow (\text{ek}, \text{dk}, \text{dk}_1, \dots, \text{dk}_n)$.
 - Compute $\text{TSIGKeyGen}(1^\kappa, n, t) \rightarrow (\text{vk}, \text{sk}, \text{sk}_1, \dots, \text{sk}_n)$.
 - Compute $\text{CommitmentGen}(1^\kappa) \rightarrow \text{params}_1$.
 - Compute $\text{PolynomialCommitmentGen}(1^\kappa, t) \rightarrow \text{params}_2$.
 - For $i = 1, \dots, n$, send $(\text{vk}, \text{ek}, \text{sk}_i, \text{dk}_i, \text{params}_1, \text{params}_2)$ to party P_i .
-

6.3.3 Protocol for Securely Realizing $\mathcal{F}_{\text{Pre}}$. Protocol Π_{Pre} securely realizes $\mathcal{F}_{\text{Pre}}$ (see Section 2.5 for the high-level overview). The protocol is designed in $(\mathcal{F}_{\text{Setup}}, \mathcal{F}_{\text{BC}})$ -hybrid model, $\mathcal{F}_{\text{ZK}}$ -hybrid model for R_{PoCS} and $\mathcal{F}_{\text{ZK}}^{1:M}$ -hybrid model for R_{PoE} and R_{PoCM} . The protocol uses Π_{ACS} as a building block.

Protocol 6.11: Π_{Pre} — Protocol for securely realizing $\mathcal{F}_{\text{Pre}}$

Generating One-Time Setup : Each $P_i \in \mathcal{P}$ sends `Init` to $\mathcal{F}_{\text{Setup}}$ and requests output from $\mathcal{F}_{\text{Setup}}$ till receiving $(\text{vk}, \text{ek}, \text{sk}_i, \text{dk}_i, \text{params}_1, \text{params}_2)$.

Generating Second Component of the Multiplication Triples : Each $P_i \in \mathcal{P}$ executes the following code.

- For $\ell = 1, \dots, c_M$, randomly pick $v^{(i,\ell)} \in \mathbb{F}$. Compute $\text{com}_{v^{(i,\ell)}} = \text{Commit}(v^{(i,\ell)}, r^{(i,\ell)})$ using a uniformly random randomness $r^{(i,\ell)}$.
- For $\ell = 1, \dots, c_M$, compute $\mathbf{c}_{v^{(i,\ell)}} = \text{Enc}_{\text{ek}}(v^{(i,\ell)}, r_1^{(i,\ell)})$ and $\mathbf{c}_{r^{(i,\ell)}} = \text{Enc}_{\text{ek}}(r^{(i,\ell)}, r_2^{(i,\ell)})$ using uniformly random randomness $r_1^{(i,\ell)}$ and $r_2^{(i,\ell)}$.

- For $\ell = 1, \dots, c_M$, set $\langle v^{(i,\ell)} \rangle = (\mathbf{com}_{v^{(i,\ell)}}, \mathbf{c}_{v^{(i,\ell)}}, \mathbf{c}_{r^{(i,\ell)}})$. % Form the committed-and-encrypted representation of the value.
- For $\ell = 1, \dots, c_M$, as a prover, send $(\text{ZK-prover}, \text{sid}_i, (\text{ek}, \langle v^{(i,\ell)} \rangle), (v^{(i,\ell)}, r^{(i,\ell)}, r_1^{(i,\ell)}, r_2^{(i,\ell)}))$ to $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoE} , where $\text{sid}_i = \text{sid} \parallel i$. % Generate a proof of consistency between the ciphertext and the commitment.
- For $\ell = 1, \dots, c_M$, request output from $\mathcal{F}_{\text{ZK}}^{1:M}$ for R_{PoE} for session id sid_j , corresponding to every $P_j \in \mathcal{P}$. Upon receiving $(\text{ZK-proof}, \text{sid}_j, \langle v^{(j,\ell)} \rangle)$ from $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoE} corresponding to P_j , do the following
 - Let $m_j = \{\langle v^{(j,\ell)} \rangle\}_{\ell=1, \dots, c_M}$. Compute $\sigma_{m_j, i} = \text{ShareSign}_{\text{sk}_i}(m_j)$ and send $(\text{sid}, \sigma_{2 \parallel j, i})$ to P_j , where $m_j = \{\langle v^{(j,\ell)} \rangle\}_{\ell=1, \dots, c_M}$.
 - Send $(\text{ZK-prover}, \text{sid}_{ij}, ((\text{vk}, m_j, \sigma_{m_j, i}), \text{sk}_i))$ to $\mathcal{F}_{\text{ZK}}$ for the relation R_{PoCS} , where P_i plays the role of prover and P_j plays the role of verifier and where $\text{sid}_{ij} = \text{sid} \parallel i \parallel j$. % Certify successful verification using signature-shares.
- Upon receiving $(\text{sid}, \sigma_{m_i, j})$ from party P_j , request output from $\mathcal{F}_{\text{ZK}}$ as a verifier where P_j plays the role of prover. Upon receiving $(\text{ZK-proof}, \text{sid}_{ji}, (\text{vk}, m_i, \sigma_{m_i, j}))$ from $\mathcal{F}_{\text{ZK}}$, include P_j to an accumulative set $\mathcal{D}_i$ (initialized to $\emptyset$). Once $|\mathcal{D}_i| \geq t + 1$, compute $\sigma_{m_i} = \text{SigCombine}(m_i, \{\sigma_{m_i, j}\}_{P_j \in \mathcal{D}_i})$. % Upon receiving enough approvals, aggregate the approvals into a threshold signature.
- Set $m_i^{\text{aug}} = \{\langle v^{(i,\ell)} \rangle\}_{\ell=1, \dots, c_M} \parallel \sigma_{m_i}$ and participate in an instance $\Pi_{\text{ACS}}^{(1)}$ of protocol Π_{ACS} with input m_i^{aug} , where the verification predicate V for this instance of Π_{ACS} is set as follows.
 - For $j = 1, \dots, n$, $V(m_j^{\text{aug}}) = 1$ iff $m_j^{\text{aug}} = \{\langle v^{(j,\ell)} \rangle\}_{\ell=1, \dots, c_M} \parallel \sigma_{m_j}$ and $\text{SigVer}_{\text{vk}}(\{\langle v^{(j,\ell)} \rangle\}_{\ell=1, \dots, c_M}, \sigma_{m_j}) = 1$.
- Let $\mathcal{CR}_1$ be the set of $n-t$ common parties obtained from the instance $\Pi_{\text{ACS}}^{(1)}$ and let $\{\{\langle v^{(j,\ell)} \rangle\}_{\ell=1, \dots, c_M} \parallel \sigma_{m_j}\}_{P_j \in \mathcal{CR}_1}$ be the output from the instance $\Pi_{\text{ACS}}^{(1)}$. For $\ell = 1, \dots, c_M$, set $\langle v^{(\ell)} \rangle = (\mathbf{com}_{v^{(\ell)}}, \mathbf{c}_{v^{(\ell)}})$, where

$$\mathbf{com}_{v^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_1} \mathbf{com}_{v^{(j,\ell)}}, \quad \mathbf{c}_{v^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_1} \mathbf{c}_{v^{(j,\ell)}}, \quad \mathbf{c}_{r^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_1} \mathbf{c}_{r^{(j,\ell)}}.$$

% Locally sum up the certified values to get the second component of Beaver's triplet.

Generating First and Third Components of the Triples : Upon computing $\{\langle v^{(\ell)} \rangle\}_{\ell=1, \dots, c_M}$, every $P_i \in \mathcal{P}$ does the following.

- For $\ell = 1, \dots, c_M$, randomly pick $u^{(i,\ell)} \in \mathbb{F}$ and randomness $\bar{r}^{(i,\ell)}, \bar{r}_1^{(i,\ell)}, \bar{r}_2^{(i,\ell)}, \alpha^{(i,\ell)}, \beta^{(i,\ell)}, \gamma^{(i,\ell)}$.
- For $\ell = 1, \dots, c_M$, compute the following:
 - $\mathbf{com}_{u^{(i,\ell)}} = \text{Commit}(u^{(i,\ell)}, \bar{r}^{(i,\ell)}); \mathbf{c}_{u^{(i,\ell)}} = \text{Enc}_{\text{ek}}(u^{(i,\ell)}, \bar{r}_1^{(i,\ell)});$
 - $\mathbf{c}_{\bar{r}^{(i,\ell)}} = \text{Enc}_{\text{ek}}(\bar{r}^{(i,\ell)}, \bar{r}_2^{(i,\ell)});$
 - $\mathbf{com}_{u^{(i,\ell)} \cdot v^{(\ell)}} = u^{(i,\ell)} \odot \mathbf{com}_{v^{(\ell)}} \oplus \text{Commit}(0, \alpha^{(i,\ell)});$
 - $\mathbf{c}_{u^{(i,\ell)} \cdot v^{(\ell)}} = u^{(i,\ell)} \boxplus \mathbf{c}_{v^{(\ell)}} \boxplus \text{Enc}_{\text{ek}}(0, \beta^{(i,\ell)});$
 - $\mathbf{c}_{u^{(i,\ell)} \cdot r^{(\ell)} + \alpha^{(i,\ell)}} = u^{(i,\ell)} \boxplus \mathbf{c}_{r^{(\ell)}} \boxplus \text{Enc}_{\text{ek}}(\alpha^{(i,\ell)}, \gamma^{(i,\ell)}).$
- For $\ell = 1, \dots, c_M$, set $\langle u^{(i,\ell)} \rangle = (\mathbf{com}_{u^{(i,\ell)}}, \mathbf{c}_{u^{(i,\ell)}}, \mathbf{c}_{\bar{r}^{(i,\ell)}})$ and $\langle w^{(i,\ell)} \rangle = (\mathbf{com}_{u^{(i,\ell)} \cdot v^{(\ell)}}, \mathbf{c}_{u^{(i,\ell)} \cdot v^{(\ell)}}, \mathbf{c}_{u^{(i,\ell)} \cdot r^{(\ell)} + \alpha^{(i,\ell)}})$, where $w^{(i,\ell)} \stackrel{\text{def}}{=} u^{(i,\ell)} \cdot v^{(\ell)}$. % Form the committed-and-encrypted representation of the values.
- As a prover send $(\text{ZK-prover}, \text{sid}_i, (\text{ek}, \langle u^{(i,\ell)} \rangle, \langle v^{(\ell)} \rangle, \langle w^{(i,\ell)} \rangle), (u^{(i,\ell)}, \bar{r}^{(i,\ell)}, \bar{r}_1^{(i,\ell)}, \bar{r}_2^{(i,\ell)}, \alpha^{(i,\ell)}, \beta^{(i,\ell)}, \gamma^{(i,\ell)}))$ to $\mathcal{F}_{\text{ZK}}^{1:M}$ for R_{PoCM} . % Generate proof for correct multiplication.
- For $\ell = 1, \dots, c_M$, request output from $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoCM} for session id sid_j , corresponding to every $P_j \in \mathcal{P}$. Upon receiving $(\text{ZK-proof}, \text{sid}_j, \langle u^{(j,\ell)} \rangle, \langle v^{(\ell)} \rangle, \langle w^{(j,\ell)} \rangle)$ from $\mathcal{F}_{\text{ZK}}^{1:M}$, do the following.
 - Let $M_j = \{\langle u^{(j,\ell)} \rangle, \langle v^{(\ell)} \rangle, \langle w^{(j,\ell)} \rangle\}_{\ell=1, \dots, c_M}$. Compute the signature share $\sigma_{M_j, i} = \text{ShareSign}_{\text{sk}_i}(M_j)$ and send $(\text{sid}, \sigma_{M_j, i})$ to P_j .
 - Send $(\text{ZK-prover}, \text{sid}_{ij}, ((\text{vk}, M_j, \sigma_{M_j, i}), \text{sk}_i))$ to $\mathcal{F}_{\text{ZK}}$ for the relation R_{PoCS} , where P_i plays the role of prover and P_j plays the role of verifier and where $\text{sid}_{ij} = \text{sid} \parallel i \parallel j$. % Certify verified Beaver shares via signature-shares.
- Upon receiving $(\text{sid}, \sigma_{M_i, j})$ from party P_j , request output from $\mathcal{F}_{\text{ZK}}$ as a verifier where P_j plays the role of prover. Upon receiving $(\text{ZK-proof}, \text{sid}_{ji}, (\text{vk}, M_i, \sigma_{M_i, j}))$ from $\mathcal{F}_{\text{ZK}}$, include P_j to an accumulative set $\mathcal{E}_i$ (initialized to $\emptyset$). Once $|\mathcal{E}_i| \geq t + 1$, compute $\sigma_{M_i} = \text{SigCombine}(M_i, \{\sigma_{M_i, j}\}_{P_j \in \mathcal{E}_i})$. % Upon receiving sufficient approvals, aggregate the approvals to generate a threshold signature.

- Set $M_i^{\text{aug}} = \{\langle u^{(i,\ell)} \rangle, \langle V^{(\ell)} \rangle, \langle w^{(i,\ell)} \rangle\}_{\ell=1,\dots,c_M} \parallel \sigma_{M_i}$ and participate in an instance $\Pi_{\text{ACS}}^{(2)}$ of protocol Π_{ACS} with input M_i^{aug} , where the verification predicate V for this instance of Π_{ACS} is set as follows.
 - For $j = 1, \dots, n$, $V(M_j^{\text{aug}}) = 1$ if and only if $M_j^{\text{aug}} = \{\langle u^{(j,\ell)} \rangle, \langle v^{(\ell)} \rangle, \langle w^{(j,\ell)} \rangle\}_{\ell=1,\dots,c_M} \parallel \sigma_{M_j}$ and $\text{SigVer}_{\text{vk}}(\{\langle u^{(j,\ell)} \rangle, \langle v^{(\ell)} \rangle, \langle w^{(j,\ell)} \rangle\}_{\ell=1,\dots,c_M}, \sigma_{M_j}) = 1$.
- Let $\mathcal{CR}_2$ be the set of $n - t$ parties obtained during $\Pi_{\text{ACS}}^{(2)}$ and let $\{\{\langle u^{(j,\ell)} \rangle, \langle v^{(\ell)} \rangle, \langle w^{(j,\ell)} \rangle\}_{\ell=1,\dots,c_M} \parallel \sigma_{M_j}\}_{P_j \in \mathcal{CR}_2}$ be the output from the instance. Set $\langle u^{(\ell)} \rangle = (\text{com}_{u^{(\ell)}}, \mathbf{c}_{u^{(\ell)}}, \mathbf{c}_{\bar{r}^{(\ell)}})$ and $\langle w^{(\ell)} \rangle = (\text{com}_{w^{(\ell)}}, \mathbf{c}_{w^{(\ell)}}, \mathbf{c}_{\bar{r}^{(\ell)}})$ and output $(\text{ek}, \text{vk}, \text{sk}_i, \text{dk}_i, \text{params}_1, \text{params}_2)$ and $\{\langle u^{(\ell)} \rangle, \langle v^{(\ell)} \rangle, \langle w^{(\ell)} \rangle\}_{\ell=1,\dots,c_M}$, where

$$\text{com}_{u^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_2} \text{com}_{u^{(j,\ell)}}, \quad \mathbf{c}_{u^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_2} \mathbf{c}_{u^{(j,\ell)}}, \quad \mathbf{c}_{\bar{r}^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_2} \mathbf{c}_{\bar{r}^{(j,\ell)}},$$

and

$$\begin{aligned} \text{com}_{w^{(\ell)}} &\stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_2} \text{com}_{u^{(j,\ell)} \cdot v^{(\ell)}}, \quad \mathbf{c}_{w^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_2} \mathbf{c}_{u^{(j,\ell)} \cdot v^{(\ell)}}, \\ \mathbf{c}_{\bar{r}^{(\ell)}} &\stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_2} \mathbf{c}_{u^{(j,\ell)} \cdot r^{(\ell)} + \alpha^{(j,\ell)}}. \end{aligned}$$

% Locally sum up the certified values to get the first and third component of Beavers triplet.

Lemma 6.12. *Assuming the existence of a THE, TSIG and polynomial commitment scheme, protocol Π_{Pre} securely realizes $\mathcal{F}_{\text{Pre}}$ with UC security in the $(\mathcal{F}_{\text{SetUp}}, \mathcal{F}_{\text{BC}})$ -hybrid model and $\mathcal{F}_{\text{ZK}}^{1:M}$ -hybrid model for R_{PoE} and R_{PoCM} .*

The protocol incurs a communication cost of $\mathbb{P}(n^2 c_M \kappa) + \mathbb{B}(n\kappa)$ bits, provided $c_M \kappa \geq (t + 1) \log n$. There will be $\mathcal{O}(nc_M)$ calls to $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relations R_{PoE} and R_{PoCM} and $\mathcal{O}(n^2)$ calls to $\mathcal{F}_{\text{ZK}}$ for the relation R_{PoCS} .

Proof. The number of calls to $\mathcal{F}_{\text{ZK}}^{1:M}$ and $\mathcal{F}_{\text{ZK}}$ for various relations follows easily from the protocol description. In the protocol, there are two instances of Π_{ACS} , where each party has an input of size $\mathcal{O}(c_M \kappa)$ bits. From Lemma 6.9, this incurs a communication of $\mathbb{P}(n^2 c_M \kappa + n^3 \kappa) + \mathbb{B}(n\kappa)$ bits, provided $c_M \kappa \geq (t + 1) \log n$.

For security, let $\mathcal{A}$ be an arbitrary real-world adversary, attacking protocol Π_{Pre} , and let $\mathcal{Z}$ be an arbitrary environment. We show the existence of a simulator $\mathcal{S}_{\text{Pre}}$, such that for any set of corrupted parties $\mathcal{C} \subset \mathcal{P}$ with $|\mathcal{C}| \leq t$ and for all inputs, the output of all honest parties and the adversary in an execution of the real protocol with $\mathcal{A}$ is indistinguishable from the output in an execution with $\mathcal{S}_{\text{Pre}}$ involving $\mathcal{F}_{\text{Pre}}$ in the ideal model, except with a negligible probability. This further implies that $\mathcal{Z}$ cannot distinguish between the two executions, except with a negligible probability.

The high-level idea of the simulator is as follows. During the simulated execution, the simulator itself performs the role of the ideal functionalities $\mathcal{F}_{\text{SetUp}}, \mathcal{F}_{\text{ZK}}^{1:M}, \mathcal{F}_{\text{ZK}}$ and $\mathcal{F}_{\text{BC}}$ (for the instances of Π_{ACS}) whenever required. Performing the role of $\mathcal{F}_{\text{SetUp}}$ allows the simulator to learn the complete setup for THE, TSIG, commitment, and polynomial commitment scheme.

During the generation of second component of the multiplication triples, whenever $\mathcal{A}$ sends an input in an encrypted-and-committed form to $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoE} for verification on the behalf of a corrupt party, the simulator performs the verification as per $\mathcal{F}_{\text{ZK}}^{1:M}$ and accordingly records these inputs on the behalf of the corrupt party. Notice that this allows the simulator to implicitly learn all the values picked by the corrupt parties during this phase. Subsequently, when $\mathcal{A}$ requests signature shares on these inputs via $\mathcal{F}_{\text{ZK}}$ for relation R_{PoCS} , the simulator verifies the input against the recorded values and generates valid signature shares using the honest parties' keys (which it possesses). On the other hand, for the honest parties, the simulator picks random values as their inputs and simulates the steps of the honest parties as per the protocol for this phase. To identify the common $\mathcal{CR}_1$ set during the simulated execution, the simulator itself performs the role of $\mathcal{F}_{\text{BC}}$ and simulates the honest parties as per the steps of the protocol Π_{ACS} . It monitors the inputs sent by $\mathcal{A}$ for corrupt parties and only accepts them if they carry a valid threshold signature (which confirms that the values are verified and extracted). This allows the simulator to learn the set $\mathcal{CR}_1$ and hence the second component of all the multiplication triples in an encrypted-and-committed fashion.

During the generation of the first and third components of the multiplication triples, whenever $\mathcal{A}$ sends an input in an encrypted-and-committed form to $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoCM} for verification on behalf of a corrupt party, the simulator performs the verification as per $\mathcal{F}_{\text{ZK}}^{1:M}$ and accordingly records these inputs on behalf of the corrupt party. Notice that this allows the simulator to implicitly learn all the values picked by the corrupt parties during this phase. Similarly, the simulator issues valid signature shares for these corrupt inputs only upon successful verification. On the other hand, for the honest parties, the simulator picks random inputs for the first component of the multiplication triples and accordingly simulates the steps of the honest parties as per the protocol for this phase. To identify the common $\mathcal{CR}_2$ set during the simulated execution, the simulator itself performs the role of $\mathcal{F}_{\text{BC}}$ and simulates the honest parties as per the steps of the protocol Π_{ACS} .

Simulator 6.13: $\mathcal{S}_{\text{Pre}}$: Simulator for the Protocol Π_{Pre}

$\mathcal{S}_{\text{Pre}}$ constructs virtual real-world honest parties and invokes the real-world adversary $\mathcal{A}$. The simulator simulates the view of $\mathcal{A}$, namely its communication with $\mathcal{Z}$, the messages sent by the honest parties, and the interaction with various functionalities. In order to simulate $\mathcal{Z}$, the simulator forwards every message it receives from $\mathcal{Z}$ to $\mathcal{A}$ and vice-versa. Let $\mathcal{C}$ be the set of corrupt parties where $|\mathcal{C}| = t$. The simulator then simulates the various phases of the protocol as follows.

Setup Phase

Upon receiving Init from $\mathcal{A}$, the simulator itself plays the role of $\mathcal{F}_{\text{SetUp}}$ and does the following.

- Compute $\text{THEKeyGen}(1^\kappa, n, t) \rightarrow (\widetilde{\text{ek}}, \widetilde{\text{dk}}, \widetilde{\text{dk}}_1, \dots, \widetilde{\text{dk}}_n)$.
- Compute $\text{TSIGKeyGen}(1^\kappa, n, t) \rightarrow (\widetilde{\text{vk}}, \widetilde{\text{sk}}, \widetilde{\text{sk}}_1, \dots, \widetilde{\text{sk}}_n)$.
- Compute $\text{CommitmentGen}(1^\kappa) \rightarrow \widetilde{\text{params}}_1$.
- Compute $\text{PolynomialCommitmentGen}(1^\kappa, t) \rightarrow \widetilde{\text{params}}_2$.
- Corresponding to every $P_i \in \mathcal{C}$, send $(\widetilde{\text{vk}}, \widetilde{\text{ek}}, \widetilde{\text{sk}}_i, \widetilde{\text{dk}}_i, \widetilde{\text{params}}_1, \widetilde{\text{params}}_2)$ to $\mathcal{A}$ on behalf of P_i .
- The simulator retains the master secret keys $\widetilde{\text{dk}}$ and $\widetilde{\text{sk}}$ and the honest parties' key share to facilitate simulation.

Generating Second Component of the Multiplication Triples

- The simulator itself performs the role of $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoE} , the role of $\mathcal{F}_{\text{ZK}}$ for the relation R_{PoCS} and also the role of $\mathcal{F}_{\text{BC}}$ during the instance $\Pi_{\text{ACS}}^{(1)}$ of Π_{ACS} .
- The simulator simulates the operations of the honest parties during this phase, by picking random values. Namely, for every $P_j \notin \mathcal{C}$, the simulator randomly picks $\widetilde{v}^{(j,\ell)} \in \mathbb{F}$ and randomness $\widetilde{r}^{(j,\ell)}, \widetilde{r}_1^{(j,\ell)}$ and $\widetilde{r}_2^{(j,\ell)}$ and computes $\mathbf{com}_{\widetilde{v}^{(j,\ell)}} = \text{Commit}(\widetilde{v}^{(j,\ell)}, \widetilde{r}^{(j,\ell)})$ and encryptions $\mathbf{c}_{\widetilde{v}^{(j,\ell)}} = \text{Enc}_{\widetilde{\text{ek}}}(\widetilde{v}^{(j,\ell)}, \widetilde{r}_1^{(j,\ell)})$ and $\mathbf{c}_{\widetilde{r}^{(j,\ell)}} = \text{Enc}_{\widetilde{\text{ek}}}(\widetilde{r}^{(j,\ell)}, \widetilde{r}_2^{(j,\ell)})$, for $\ell = 1, \dots, c_M$. The simulator then sets $\langle \widetilde{v}^{(j,\ell)} \rangle = (\mathbf{com}_{\widetilde{v}^{(j,\ell)}}, \mathbf{c}_{\widetilde{v}^{(j,\ell)}}, \mathbf{c}_{\widetilde{r}^{(j,\ell)}})$.
- Whenever $\mathcal{A}$ requests output from $\mathcal{F}_{\text{ZK}}^{1:M}$ for relation R_{PoE} with sid_j for any $P_j \notin \mathcal{C}$, the simulator responds with output $\{\langle \widetilde{v}^{(j,\ell)} \rangle\}_{\ell=1, \dots, c_M}$.
- Whenever $\mathcal{A}$ requests signature shares on $\{\langle \widetilde{v}^{(j,\ell)} \rangle\}$ via $\mathcal{F}_{\text{ZK}}$ for relation R_{PoCS} , the simulator sends valid signature shares on behalf of honest parties using the retained keys.
- Whenever $\mathcal{A}$ sends (ZK-prover, $\text{sid}_i, (\widetilde{\text{ek}}, \{\langle \widetilde{v}^{(i,\ell)} \rangle\}_{\ell=1, \dots, c_M}), (\{\widetilde{v}^{(i,\ell)}, \widetilde{r}^{(i,\ell)}, \widetilde{r}_1^{(i,\ell)}, \widetilde{r}_2^{(i,\ell)}\}_{\ell=1, \dots, c_M}))$ to $\mathcal{F}_{\text{ZK}}^{1:M}$ on behalf of any $P_i \in \mathcal{C}$ for the relation R_{PoE} where P_i is the prover, the simulator records $(\widetilde{v}^{(i,\ell)}, \widetilde{r}^{(i,\ell)}, \widetilde{r}_1^{(i,\ell)}, \widetilde{r}_2^{(i,\ell)})$ on behalf of P_i , provided $\mathbf{com}_{\widetilde{v}^{(i,\ell)}} = \text{Commit}(\widetilde{v}^{(i,\ell)}, \widetilde{r}^{(i,\ell)})$, $\mathbf{c}_{\widetilde{v}^{(i,\ell)}} = \text{Enc}_{\widetilde{\text{ek}}}(\widetilde{v}^{(i,\ell)}, \widetilde{r}_1^{(i,\ell)})$ and $\mathbf{c}_{\widetilde{r}^{(i,\ell)}} = \text{Enc}_{\widetilde{\text{ek}}}(\widetilde{r}^{(i,\ell)}, \widetilde{r}_2^{(i,\ell)})$. Whenever $\mathcal{A}$ requests signature shares via $\mathcal{F}_{\text{ZK}}$ for relation R_{PoCS} on corrupt input m_i , the simulator verifies m_i against the recorded values. If correct, it generates valid signature shares using honest keys and sends them to $\mathcal{A}$.
- Whenever $\mathcal{A}$ sends $\{\langle \widetilde{v}^{(i,\ell)} \rangle\}_{\ell=1, \dots, c_M}$ to $\mathcal{F}_{\text{BC}}$ for $\Pi_{\text{ACS}}^{(1)}$ as input on behalf of any $P_i \in \mathcal{C}$, the simulator plays the role of $\mathcal{F}_{\text{BC}}$ by verifying the attached threshold signature using $\widetilde{\text{vk}}$. It appends the input to the ledger only if the signature is valid.

- When the first honest party completes this phase in the simulated execution, $\mathcal{S}_{\text{Pre}}$ learns the set $\mathcal{CR}_1$ and for $\ell = 1, \dots, c_M$ sets $\langle \tilde{v}^{(\ell)} \rangle = (\mathbf{com}_{\tilde{v}^{(\ell)}}, \mathbf{c}_{\tilde{v}^{(\ell)}}, \mathbf{c}_{\tilde{r}^{(\ell)}})$, where

$$\mathbf{com}_{\tilde{v}^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_1} \mathbf{com}_{\tilde{v}^{(j, \ell)}}, \quad \mathbf{c}_{\tilde{v}^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_1} \mathbf{c}_{\tilde{v}^{(j, \ell)}}, \quad \mathbf{c}_{\tilde{r}^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_1} \mathbf{c}_{\tilde{r}^{(j, \ell)}}.$$

Generating First and Third Components of the Multiplication Triples

- The simulator itself performs the role of $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoCM} and also the role of $\mathcal{F}_{\text{BC}}$ during the instance $\Pi_{\text{ACS}}^{(2)}$ of Π_{ACS} .
- The simulator simulates the operations of the honest parties during this phase, by picking random values. Namely, for every $P_j \notin \mathcal{C}$, the simulator randomly picks $\tilde{u}^{(j, \ell)} \in \mathbb{F}$ and randomness $\tilde{r}^{(j, \ell)}, \tilde{r}_1^{(j, \ell)}, \tilde{r}_2^{(j, \ell)}, \tilde{\alpha}^{(j, \ell)}, \tilde{\beta}^{(j, \ell)}, \tilde{\gamma}^{(j, \ell)}$ and computes the following for $\ell = 1, \dots, c_M$
 - $\mathbf{com}_{\tilde{u}^{(j, \ell)}} = \text{Commit}(\tilde{u}^{(j, \ell)}, \tilde{r}^{(j, \ell)});$
 - $\mathbf{c}_{\tilde{u}^{(j, \ell)}} = \text{Enc}_{\text{ek}}(\tilde{u}^{(j, \ell)}, \tilde{r}_1^{(j, \ell)});$
 - $\mathbf{c}_{\tilde{r}^{(j, \ell)}} = \text{Enc}_{\text{ek}}(\tilde{r}^{(j, \ell)}, \tilde{r}_2^{(j, \ell)});$
 - $\mathbf{com}_{\tilde{u}^{(j, \ell)}, \tilde{v}^{(\ell)}} = \tilde{u}^{(j, \ell)} \odot \mathbf{com}_{\tilde{v}^{(\ell)}} \oplus \text{Commit}(0, \tilde{\alpha}^{(j, \ell)});$
 - $\mathbf{c}_{\tilde{u}^{(j, \ell)}, \tilde{v}^{(\ell)}} = \tilde{u}^{(j, \ell)} \boxtimes \mathbf{c}_{\tilde{v}^{(\ell)}} \boxplus \text{Enc}_{\text{ek}}(0, \tilde{\beta}^{(j, \ell)});$
 - $\mathbf{c}_{\tilde{u}^{(j, \ell)}, \tilde{r}^{(\ell)} + \tilde{\alpha}^{(j, \ell)}} = \tilde{u}^{(j, \ell)} \boxtimes \mathbf{c}_{\tilde{r}^{(\ell)}} \boxplus \text{Enc}_{\text{ek}}(\tilde{\alpha}^{(j, \ell)}, \tilde{\gamma}^{(j, \ell)}).$
- For $\ell = 1, \dots, c_M$, set $\langle \tilde{u}^{(j, \ell)} \rangle = (\mathbf{com}_{\tilde{u}^{(j, \ell)}}, \mathbf{c}_{\tilde{u}^{(j, \ell)}}, \mathbf{c}_{\tilde{r}^{(j, \ell)}})$ and $\langle \tilde{w}^{(j, \ell)} \rangle = (\mathbf{com}_{\tilde{u}^{(j, \ell)}, \tilde{v}^{(\ell)}}, \mathbf{c}_{\tilde{u}^{(j, \ell)}, \tilde{v}^{(\ell)}}, \mathbf{c}_{\tilde{u}^{(j, \ell)}, \tilde{r}^{(\ell)} + \tilde{\alpha}^{(j, \ell)}})$, where $\tilde{w}^{(j, \ell)} \stackrel{\text{def}}{=} \tilde{u}^{(j, \ell)} \cdot \tilde{v}^{(\ell)}$.
- Whenever $\mathcal{A}$ requests output from $\mathcal{F}_{\text{BC}}$ and $\mathcal{F}_{\text{ZK}}^{1:M}$ for relation R_{PoCM} with $\text{sid}_j^{(2)}$ for any $P_j \notin \mathcal{C}$, the simulator responds with output $\{\langle \tilde{u}^{(j, \ell)} \rangle, \langle \tilde{w}^{(j, \ell)} \rangle\}_{\ell=1, \dots, c_M}$. Whenever $\mathcal{A}$ requests signature shares on honest inputs via $\mathcal{F}_{\text{ZK}}$ for R_{PoCS} , the simulator sends valid shares.
- Whenever $\mathcal{A}$ sends $(\text{ZK-prover}, \text{sid}_i^{(2)}, (\text{ek}, \{\langle \tilde{u}^{(i, \ell)} \rangle\}_{\ell=1, \dots, c_M}, \{\langle \tilde{v}^{(\ell)} \rangle\}_{\ell=1, \dots, c_M}, \{\langle \tilde{w}^{(i, \ell)} \rangle\}_{\ell=1, \dots, c_M}), (\{\tilde{u}^{(i, \ell)}\}_{\ell=1, \dots, c_M}, \{\tilde{r}_1^{(i, \ell)}\}_{\ell=1, \dots, c_M}, \{\tilde{r}_2^{(i, \ell)}\}_{\ell=1, \dots, c_M}, \{\tilde{\alpha}^{(i, \ell)}\}_{\ell=1, \dots, c_M}, \{\tilde{\beta}^{(i, \ell)}\}_{\ell=1, \dots, c_M}, \{\tilde{\gamma}^{(i, \ell)}\}_{\ell=1, \dots, c_M}))$ to $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoCM} , the simulator records $\tilde{u}^{(i, \ell)}, \tilde{w}^{(i, \ell)}, \tilde{r}^{(i, \ell)}, \tilde{r}_1^{(i, \ell)}, \tilde{r}_2^{(i, \ell)}, \tilde{\alpha}^{(i, \ell)}, \tilde{\beta}^{(i, \ell)}$ and $\tilde{\gamma}^{(i, \ell)}$ on behalf of P_i , provided all the following hold.
 - $\mathbf{com}_{\tilde{u}^{(i, \ell)}} = \text{Commit}(\tilde{u}^{(i, \ell)}, \tilde{r}^{(i, \ell)});$
 - $\mathbf{c}_{\tilde{u}^{(i, \ell)}} = \text{Enc}_{\text{ek}}(\tilde{u}^{(i, \ell)}, \tilde{r}_1^{(i, \ell)});$
 - $\mathbf{c}_{\tilde{r}^{(i, \ell)}} = \text{Enc}_{\text{ek}}(\tilde{r}^{(i, \ell)}, \tilde{r}_2^{(i, \ell)});$
 - $\mathbf{com}_{\tilde{u}^{(i, \ell)}, \tilde{v}^{(\ell)}} = \tilde{u}^{(i, \ell)} \odot \mathbf{com}_{\tilde{v}^{(\ell)}} \oplus \text{Commit}(0, \tilde{\alpha}^{(i, \ell)});$
 - $\mathbf{c}_{\tilde{u}^{(i, \ell)}, \tilde{v}^{(\ell)}} = \tilde{u}^{(i, \ell)} \boxtimes \mathbf{c}_{\tilde{v}^{(\ell)}} \boxplus \text{Enc}_{\text{ek}}(0, \tilde{\beta}^{(i, \ell)});$
 - $\mathbf{c}_{\tilde{u}^{(i, \ell)}, \tilde{r}^{(\ell)} + \tilde{\alpha}^{(i, \ell)}} = \tilde{u}^{(i, \ell)} \boxtimes \mathbf{c}_{\tilde{r}^{(\ell)}} \boxplus \text{Enc}_{\text{ek}}(\tilde{\alpha}^{(i, \ell)}, \tilde{\gamma}^{(i, \ell)}).$
- Whenever $\mathcal{A}$ requests signature shares via $\mathcal{F}_{\text{ZK}}$ for relation R_{PoCS} on corrupt input M_i , the simulator verifies M_i against against the recorded values. If correct, it generates valid signature shares using honest keys and sends them to $\mathcal{A}$.
- Whenever $\mathcal{A}$ sends $\{\langle \tilde{u}^{(i, \ell)} \rangle, \langle \tilde{w}^{(i, \ell)} \rangle\}_{\ell=1, \dots, c_M}$ to $\mathcal{F}_{\text{BC}}$ as input on behalf of any $P_i \in \mathcal{C}$, the simulator plays the role of $\mathcal{F}_{\text{BC}}$ by verifying the attached threshold signature using $\tilde{\text{vk}}$. It appends the input to the ledger only if the signature is valid.
- When the first honest party completes this phase in the simulated execution, $\mathcal{S}_{\text{Pre}}$ learns the set $\mathcal{CR}_2$ and for $\ell = 1, \dots, c_M$ sets $\langle \tilde{u}^{(\ell)} \rangle = (\mathbf{com}_{\tilde{u}^{(\ell)}}, \mathbf{c}_{\tilde{u}^{(\ell)}}, \mathbf{c}_{\tilde{r}^{(\ell)}})$ and $\langle \tilde{w}^{(\ell)} \rangle = (\mathbf{com}_{\tilde{w}^{(\ell)}}, \mathbf{c}_{\tilde{w}^{(\ell)}}, \mathbf{c}_{\tilde{r}^{(\ell)}})$, where

$$\mathbf{com}_{\tilde{u}^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_2} \mathbf{com}_{\tilde{u}^{(j, \ell)}}, \quad \mathbf{c}_{\tilde{u}^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_2} \mathbf{c}_{\tilde{u}^{(j, \ell)}}, \quad \mathbf{c}_{\tilde{r}^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_2} \mathbf{c}_{\tilde{r}^{(j, \ell)}},$$

and

$$\mathbf{com}_{\tilde{w}^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_2} \mathbf{com}_{\tilde{u}^{(j, \ell)}, \tilde{v}^{(\ell)}}, \quad \mathbf{c}_{\tilde{w}^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_2} \mathbf{c}_{\tilde{u}^{(j, \ell)}, \tilde{v}^{(\ell)}},$$

$$\mathbf{c}_{\tilde{r}^{(\ell)}} \stackrel{\text{def}}{=} \bigoplus_{P_j \in \mathcal{CR}_2} \mathbf{c}_{\tilde{u}^{(j, \ell)}, \tilde{r}^{(\ell)} + \tilde{\alpha}^{(j, \ell)}}.$$

We next prove a sequence of claims, which helps us to show that the joint distribution of the parties are identical in both the real-world, as well as simulated world, except with a negligible probability. We first claim that in any execution of Π_{Pre} , a set $\mathcal{CR}_1$ and $\mathcal{CR}_2$ consisting of $n - t$ parties is eventually generated.

Claim 6.14. *In any execution of protocol Π_{Pre} , a set $\mathcal{CR}_1$ and $\mathcal{CR}_2$ is eventually generated where $|\mathcal{CR}_1| \geq n - t$ and $|\mathcal{CR}_2| \geq n - t$. Moreover, for every $P_j \in \mathcal{CR}_1$, there exists some $\{v^{(j,\ell)}\}_{\ell=1,\dots,c_M} \in \mathbb{F}$ known to P_j for which all honest parties eventually hold $\{\langle v^{(j,\ell)} \rangle\}_{\ell=1,\dots,c_M}$. Furthermore, for every $P_k \in \mathcal{CR}_2$, there exist some $\{u^{(k,\ell)}, w^{(k,\ell)}\}_{\ell=1,\dots,c_M} \in \mathbb{F}$ known to P_k for which all honest parties eventually hold $\{\langle u^{(k,\ell)} \rangle, \langle w^{(k,\ell)} \rangle\}_{\ell=1,\dots,c_M}$.*

Proof. We first note that there always exist at least $n - t$ parties, namely the set of honest parties, who will send their contributions for the multiplication triples in an encrypted-and-committed fashion, prove the correctness of their contributions via $\mathcal{F}_{\text{ZK}}^{1:M}$, obtain valid signatures, and participate in Π_{ACS} . So even if the honest parties do not receive any output from the instances of $\mathcal{F}_{\text{BC}}$ and $\mathcal{F}_{\text{ZK}}^{1:M}$, which are supposed to be invoked by the potentially corrupt parties, from the liveness property of Π_{ACS} (Lemma 6.9), sets $\mathcal{CR}_1$ and $\mathcal{CR}_2$ are guaranteed to be generated eventually.

Now consider an arbitrary $P_j \in \mathcal{CR}_1$. Since $P_j \in \mathcal{CR}_1$, it follows that P_j 's input $\{\langle v^{(j,\ell)} \rangle\}_{\ell=1,\dots,c_M}$ was included in the output of $\Pi_{\text{ACS}}^{(1)}$. This implies P_j provided a valid threshold signature σ_{m_j} on its input. A valid threshold signature implies that at least one honest party verified P_j 's input via $\mathcal{F}_{\text{ZK}}^{1:M}$ for relation R_{PoE} . It then follows from the properties of $\mathcal{F}_{\text{ZK}}^{1:M}$ that P_j would have provided witnesses $(\{v^{(j,\ell)}\}_{\ell=1,\dots,c_M}, \star, \star, \star)$ for verification and hence P_j knows the inputs $\{v^{(j,\ell)}\}_{\ell=1,\dots,c_M}$ as claimed.

Furthermore, consider an arbitrary $P_k \in \mathcal{CR}_2$. Since $P_k \in \mathcal{CR}_2$, it follows that P_k 's input was included in the output of $\Pi_{\text{ACS}}^{(2)}$. This implies that a valid threshold signature, meaning at least one honest party verified P_k 's input via $\mathcal{F}_{\text{ZK}}^{1:M}$ for relation R_{PoCM} . Consequently, P_k knows the values $\{u^{(k,\ell)}\}_{\ell=1,\dots,c_M}$ and $\{w^{(k,\ell)}\}_{\ell=1,\dots,c_M}$ as claimed. $\square$

We next show that the view generated by $\mathcal{S}_{\text{Pre}}$ for $\mathcal{A}$ is indistinguishable from $\mathcal{A}$'s view during the real execution of Π_{Pre} , except with a negligible probability.

Claim 6.15. *The view of $\mathcal{A}$ in the simulated execution with $\mathcal{S}_{\text{Pre}}$ is indistinguishable from the view of $\mathcal{A}$ in the real execution of Π_{Pre} , except with a negligible probability.*

Proof. It is easy to see that the view of $\mathcal{A}$ during the setup generation phase is identically distributed in both executions. This is because in both executions, $\mathcal{A}$ receives no messages from the honest parties, and the step of $\mathcal{F}_{\text{Setup}}$ is executed by the simulator itself in the simulated execution. Namely, in both executions, $\mathcal{A}$'s view consists of the setup data for THE, TSig, commitment scheme, and polynomial commitment scheme. So let us fix the data held by $\mathcal{A}$ at the end of the setup generation phase in both the executions.

Now conditioned on the view of $\mathcal{A}$ for the setup generation phase, during the generation of the second component of multiplication triples, $\mathcal{A}$ learns $\{\langle v^{(j,\ell)} \rangle\}_{\ell=1,\dots,c_M}$ along with a valid threshold signature for every $P_j \notin \mathcal{C}$ during the real execution, while in the simulated execution it learns $\{\langle \tilde{v}^{(j,\ell)} \rangle\}_{\ell=1,\dots,c_M}$ along with a simulated threshold signature for every $P_j \notin \mathcal{C}$. It then follows from the semantic security of THE, the hiding property of the commitment scheme, and the security of the TSIG scheme that the distributions of $\{\langle v^{(j,\ell)} \rangle\}_{P_j \notin \mathcal{C}}$ and $\{\langle \tilde{v}^{(j,\ell)} \rangle\}_{P_j \notin \mathcal{C}}$ are indistinguishable, except with a negligible probability. So let us fix these encrypted-and-committed values. Since the role of $\mathcal{F}_{\text{BC}}$ and $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoE} is played by the simulator itself, it follows easily that the view of $\mathcal{A}$ during the selection of the set $\mathcal{CR}_1$ is identically distributed in both the real as well as the simulated execution, except with a negligible probability.

Next consider the generation of the first and third components of the multiplication triples. In the simulated execution, for every $P_j \notin \mathcal{C}$, the simulator picks random values $\{\tilde{u}^{(j,\ell)}\}_{\ell=1,\dots,c_M}$ and computes $\{\tilde{w}^{(j,\ell)} = \tilde{u}^{(j,\ell)} \cdot \tilde{v}^{(j,\ell)}\}_{\ell=1,\dots,c_M}$ to generate the messages. Similarly to the previous phase, $\mathcal{A}$ learns $\{\langle u^{(j,\ell)} \rangle, \langle w^{(j,\ell)} \rangle\}_{\ell=1,\dots,c_M}$ along with a valid threshold signature in the real execution and $\{\langle \tilde{u}^{(j,\ell)} \rangle, \langle \tilde{w}^{(j,\ell)} \rangle\}_{\ell=1,\dots,c_M}$ along with a simulated threshold signature in the simulated execution. From the semantic security of THE, the hiding property of the commitment scheme, and the security of the

TSIG scheme, these distributions are indistinguishable, except with a negligible probability. Furthermore, since the simulator knows the second component $\{v^\ell\}_{\ell=1,\dots,c_M}$ in an encrypted-and-committed fashion, it can correctly compute valid $\{\langle \tilde{w}^{(\ell)} \rangle\}_{\ell=1,\dots,c_M}$ that satisfy the relation R_{PoCM} . Since the role $\mathcal{F}_{\text{BC}}$ and $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoCM} is played by the simulator itself, it follows that the view of $\mathcal{A}$ during the selection of the set $\mathcal{CR}_2$ is identically distributed in both executions, except with a negligible probability. $\square$

We next claim that conditioned on the view of $\mathcal{A}$, the output of the honest parties is identically distributed in both worlds, except with a negligible probability.

Claim 6.16. *Conditioned on the view of $\mathcal{A}$, the output of the honest parties is identically distributed in the real execution of Π_{Pre} involving $\mathcal{A}$, as well as in the ideal execution involving $\mathcal{S}_{\text{Pre}}$ and $\mathcal{F}_{\text{Pre}}$, except with a negligible probability.*

Proof. Let View be an arbitrary view of $\mathcal{A}$. And let $\mathcal{CR}_1$ and $\mathcal{CR}_2$ be the sets of input-providers as determined by View (from Claim 6.14, such sets are bound to exist). Moreover, according to View , for every $P_j \in \mathcal{CR}_1$, there exists some inputs $\{v^{(j,\ell)}\}_{\ell=1,\dots,c_M} \in \mathbb{F}$, such that the (honest) parties eventually hold $\{\langle v^{(j,\ell)} \rangle\}_{\ell=1,\dots,c_M}$. Furthermore, from Claim 6.15, if $P_j \in \mathcal{CR}_1$, then the corresponding $\{v^{(j,\ell)}\}_{\ell=1,\dots,c_M}$ and the underlying randomness used in $\{\langle v^{(j,\ell)} \rangle\}_{\ell=1,\dots,c_M}$ are included in View . On the other hand, for $P_j \notin \mathcal{C}$, the corresponding $\{v^{(j,\ell)}\}_{\ell=1,\dots,c_M}$ are random, conditioned on $\{\langle v^{(j,\ell)} \rangle\}_{\ell=1,\dots,c_M}$ available with $\mathcal{A}$ as determined by View . Let us fix the $\{v^{(j,\ell)}\}_{\ell=1,\dots,c_M}$ values corresponding to the parties in $\mathcal{CR}_1$. Similar logic applies to the values $\{u^{(k,\ell)}\}_{\ell=1,\dots,c_M}$ and $\{w^{(k,\ell)}\}_{\ell=1,\dots,c_M}$ for parties in $\mathcal{CR}_2$.

It is easy to see that in the ideal world, the output of the honest parties consists of random multiplication triples $\{\langle u^{(\ell)} \rangle, \langle v^{(\ell)} \rangle, \langle w^{(\ell)} \rangle\}_{\ell=1,\dots,c_M}$, where $w^{(\ell)} = u^{(\ell)} \cdot v^{(\ell)}$ for all $\ell = 1, \dots, c_M$. This is because $\mathcal{S}_{\text{Pre}}$ provides the identity of $\mathcal{CR}_1$ and $\mathcal{CR}_2$ and simulates the steps of $\mathcal{F}_{\text{BC}}$, and honest parties have to derive the aggregated triples based on the sums of contributions from these core sets.

We now show that the honest parties output valid multiplication triples even in the real world, except with a negligible probability. For this, we argue that all the values during the preprocessing phase are available to the parties in an encrypted-and-committed fashion. Since the generation of the second component v involves summing valid encrypted-and-committed shares from $\mathcal{CR}_1$, it follows that v will be correctly available in an encrypted-and-committed fashion. During the generation of the first and third components, the honest parties verify the validity of each contribution via $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoCM} and check for the valid signatures in Π_{ACS} . Specifically, for every $P_k \in \mathcal{CR}_2$, the proofs guarantee that $w^{(k,\ell)} = u^{(k,\ell)} \cdot v^{(k,\ell)}$ for all $\ell = 1, \dots, c_M$. Since the final triples are linear combinations (sums) of these contributions, and since the homomorphic properties hold, it follows that the final output satisfies $\langle w^{(\ell)} \rangle = \langle u^{(\ell)} \rangle \cdot \langle v^{(\ell)} \rangle$ for all $\ell = 1, \dots, c_M$, except with a negligible probability.

On the other hand, if $\mathcal{F}_{\text{ZK}}^{1:M}$ delivers any proof on behalf of a corrupt party, then the underlying statement is correct. As there are at least $n - t$ honest parties who contribute to the core sets, and the final output is derived deterministically from contributions of $\mathcal{CR}_1$ and $\mathcal{CR}_2$, the distribution of the final triples is identical to the ideal worlds where the simulator uses the same core sets and corrupt inputs. $\square$

The lemma now follows from Claim 6.15 and Claim 6.16. $\square$

From Theorem 6.6, we can securely realize the calls to $\mathcal{F}_{\text{ZK}}^{1:M}$ assuming a setup for TSIG, which can be made available through $\mathcal{F}_{\text{SetUp}}$. Consequently, we get the following corollary to Lemma 6.12.

Corollary 6.17. *Assuming the existence of a THE, TSIG, and polynomial commitment scheme, protocol Π_{Pre} securely realizes $\mathcal{F}_{\text{Pre}}$ with UC security in the $(\mathcal{F}_{\text{SetUp}}, \mathcal{F}_{\text{BC}})$ -hybrid model. The protocol incurs a communication of $\mathbb{P}(n^2 c_M \kappa) + \mathbb{B}(n\kappa)$ bits, provided $c_M \kappa \geq (t + 1) \log n$.*

6.4 Asynchronous Protocol for Public Decryption with a Linear Overhead

Given $\langle u \rangle$, to publicly reconstruct u , every party P_i can compute a decryption share u_i and send it to every other party and prove the correctness of the decryption share using a ZK proof. Once every party gets $t + 1$ correct decryption shares (which will arrive eventually as we have at least $t + 1$ honest parties), u can be computed. This simple protocol will require a communication of $\mathbb{P}(n^2 \kappa)$ bits. Now consider a scenario where there exist inputs $\langle u^{(0)} \rangle, \dots, \langle u^{(t)} \rangle$ and the goal is to publicly reconstruct $u^{(0)}, \dots, u^{(t)}$. Using the naive protocol outlined earlier $t + 1$ times will incur a communication of $\mathbb{P}(n^3 \kappa)$ bits. Instead, we use protocol Π_{PubDec} which incurs a communication of $\mathbb{P}(n^2 \kappa)$ bits and hence involves a linear overhead.

The protocol is based on an idea used in [10]. Let $f(X)$ be the t -degree polynomial whose coefficients are $u^{(0)}, \dots, u^{(t)}$; note that the parties don't know $f(X)$, since their goal is to reconstruct $u^{(0)}, \dots, u^{(t)}$. Let $v^{(1)}, \dots, v^{(n)}$ distinct points on the polynomial $f(X)$. Each point $v^{(i)}$ is a publicly-known linear function of $u^{(0)}, \dots, u^{(t)}$, where the linear combinners are Lagrange's interpolation coefficients. Hence the parties can locally compute $\langle v^{(1)} \rangle, \dots, \langle v^{(n)} \rangle$. Next, each $v^{(i)}$ is reconstructed *only* towards party P_i by using the naive reconstruction protocol, where the decryption shares of $v^{(i)}$ are sent only to P_i . Once P_i reconstructs $v^{(i)}$, it sends it to all, which can be verified against the commitment of $v^{(i)}$ available as part of $\langle v^{(i)} \rangle$. As there are at least $t + 1$ honest parties, every honest party will eventually get $t + 1$ correct $v^{(i)}$ points, using which $f(X)$ can be interpolated and $u^{(0)}, \dots, u^{(t)}$ can be computed.

Protocol Π_{PubDec} is formally presented in Protocol 6.18.

Protocol 6.18: Π_{PubDec} — Protocol for asynchronous public decryption with a linear overhead in the $\mathcal{F}_{\text{ZK}}$ -hybrid model

Public Input: $\{\langle u^{(i)} \rangle\}_{i \in \{0, \dots, t\}}$, where $\langle u^{(i)} \rangle = (\text{com}_{u^{(i)}}, \mathbf{c}_{u^{(i)}}, \mathbf{c}_{r^{(i)}})$ such that $\text{com}_{u^{(i)}} = \text{Commit}(u^{(i)}, r^{(i)})$

Interpolating Polynomial through Inputs — Every party in $\mathcal{P}$ executes the following code.

1. Let $f(X) \stackrel{\text{def}}{=} u^{(0)} + u^{(1)}X + \dots + u^{(t)}X^t$. For $j = 1, \dots, n$, let $v^{(j)} \stackrel{\text{def}}{=} f(j)$.
2. For $j = 1, \dots, n$, locally compute $\langle v^{(j)} \rangle$ from $\langle u^{(0)} \rangle, \dots, \langle u^{(t)} \rangle$. Let $\langle v^{(j)} \rangle = (\text{com}_{v^{(j)}}, \mathbf{c}_{v^{(j)}}, \mathbf{c}_{R^{(j)}})$, where $\text{com}_{v^{(j)}} = \text{Commit}(v^{(j)}, R^{(j)})$.

Decrypting Points on the Polynomial Towards Designated Parties — Every party $P_i \in \mathcal{P}$ executes the following code.

1. For $j = 1, \dots, n$, compute $v_i^{(j)} := \text{Dec}_i(\mathbf{c}_{v^{(j)}}, \text{dk}_i)$ and $R_i^{(j)} := \text{Dec}_i(\mathbf{c}_{R^{(j)}}, \text{dk}_i)$.
2. For $j = 1, \dots, n$, send $(\text{sid}, v_i^{(j)}, R_i^{(j)})$ to party P_j , $(\text{ZK-prover}, \text{sid}, (\text{ek}, \mathbf{c}_{v^{(j)}}, v_i^{(j)}), \text{dk}_i)$ and $(\text{ZK-prover}, \text{sid}, (\text{ek}, \mathbf{c}_{R^{(j)}}, R_i^{(j)}), \text{dk}_i)$ to $\mathcal{F}_{\text{ZK}}$ for the relation R_{PoCD} , where P_i is the prover and P_j is the verifier.
3. For $j = 1, \dots, n$, upon receiving $(\text{sid}, v_j^{(i)}, R_j^{(i)})$ from P_j , request output from $\mathcal{F}_{\text{ZK}}$ for the relation R_{PoCD} , until $(\text{ZK-proof}, \text{sid}, (\text{ek}, \mathbf{c}_{v^{(i)}}, v_j^{(i)}))$ and $(\text{ZK-proof}, \text{sid}, (\text{ek}, \mathbf{c}_{R^{(i)}}, R_j^{(i)}))$ are received from $\mathcal{F}_{\text{ZK}}$, where P_j is the prover and P_i is the verifier. Upon receiving $(\text{ZK-proof}, \text{sid}, (\text{ek}, \mathbf{c}_{v^{(i)}}, v_j^{(i)}))$ and $(\text{ZK-proof}, \text{sid}, (\text{ek}, \mathbf{c}_{R^{(i)}}, R_j^{(i)}))$, include P_j to an accumulative set $\mathcal{D}_i^{(1)}$ (initialized to $\emptyset$).
4. Once $|\mathcal{D}_i^{(1)}| = t+1$, compute $v^{(i)} = \text{Combine}(\mathbf{c}_{v^{(i)}}, \{v_j^{(i)}\}_{P_j \in \mathcal{D}_i})$ and $R^{(i)} = \text{Combine}(\mathbf{c}_{R^{(i)}}, \{R_j^{(i)}\}_{P_j \in \mathcal{D}_i})$. Send $(\text{sid}, v^{(i)}, R^{(i)})$ to every party in $\mathcal{P}$.

Output Computation — Every party $P_i \in \mathcal{P}$ executes the following code.

1. Upon receiving $(\text{sid}, v^{(j)}, R^{(j)})$ from P_j , include P_j to an accumulative set $\mathcal{D}_i^{(2)}$ (initialized to $\emptyset$), provided $\text{com}_{v^{(j)}} \stackrel{?}{=} \text{Commit}(v^{(j)}, R^{(j)})$ is true.
 2. Once $|\mathcal{D}_i^{(2)}| = t + 1$, interpolate the points $\{(j, v^{(j)})\}_{P_j \in \mathcal{D}_i^{(2)}}$ to interpolate the t -degree polynomial $f(X)$. Output the coefficients of $f(X)$.
-

If there are L encrypted-and-committed inputs which need to be publicly reconstructed, where $L \geq t + 1$, then they can be divided into groups of size $t + 1$, and an instance of protocol Π_{PubDec} can be executed for every group. This will incur a communication of $\mathbb{P}(\frac{L}{t+1} \cdot n^2 \kappa) = \mathbb{P}(nL\kappa)$ bits. The proof of Lemma 6.19 simply follows from the above discussion and the properties of THE and the commitment scheme.

Lemma 6.19. *Given inputs $\langle w^{(1)} \rangle, \dots, \langle w^{(L)} \rangle$, where $L \geq t + 1$, there exists a protocol in $\mathcal{F}_{\text{ZK}}$ -hybrid model for the relation R_{PoCD} , which eventually outputs $w^{(1)}, \dots, w^{(L)}$ for all (honest) parties, except with a negligible probability. The protocol incurs a communication of $\mathbb{P}(nL\kappa)$ bits.*

6.5 Beaver's Multiplication Protocol

Once we have a protocol for public reconstruction of encrypted-and-committed values with a linear overhead, we naturally get a protocol for computing the product of encrypted-and-committed values with a linear overhead, using standard Beaver's trick [9]. In more detail, protocol $\Pi_{\text{BatchBeaver}}$ takes input $t + 1$

pairs of encrypted-and-committed values, along with $t + 1$ encrypted-and-committed multiplication-triples. The protocol outputs the product of the input pairs in encrypted-and-committed fashion. The protocol incurs a communication of $\mathbb{P}(n^2\kappa)$ bits and hence has an amortized communication of $\mathbb{P}(n\kappa)$ bits. The protocol is formally presented in Protocol 6.20.

Protocol 6.20: $\Pi_{\text{BatchBeaver}}$ — Asynchronous protocol for securely computing the product of encrypted-and-committed values with a linear overhead

Public Input: $\{\langle u^{(i)} \rangle, \langle v^{(i)} \rangle, \langle a^{(i)} \rangle, \langle b^{(i)} \rangle, \langle c^{(i)} \rangle\}_{i \in \{1, \dots, t+1\}}$

1. For $i = 1, \dots, t + 1$, let $d^{(i)} = u^{(i)} - a^{(i)}$ and $e^{(i)} = v^{(i)} - b^{(i)}$. The parties locally compute $\langle d^{(i)} \rangle$ and $\langle e^{(i)} \rangle$ from $\langle u^{(i)} \rangle, \langle a^{(i)} \rangle$ and $\langle v^{(i)} \rangle, \langle b^{(i)} \rangle$ respectively.
 2. The parties run an instance of Π_{PubDec} with inputs $\{\langle d^{(i)} \rangle, \langle e^{(i)} \rangle\}_{i \in \{1, \dots, t+1\}}$ and publicly reconstruct $d^{(i)}, e^{(i)}$.
 3. For $i = 1, \dots, t + 1$, let $w^{(i)} = d^{(i)}e^{(i)} + d^{(i)}b^{(i)} + a^{(i)}e^{(i)} + c^{(i)}$. The parties locally compute $\langle w^{(i)} \rangle$ from $d^{(i)}, e^{(i)}, \langle a^{(i)} \rangle, \langle b^{(i)} \rangle$ and $\langle c^{(i)} \rangle$ and output $\{\langle w^{(i)} \rangle\}_{i \in \{1, \dots, t+1\}}$.
-

The proof of Lemma 6.21 simply follows from the properties of protocol Π_{PubDec} .

Lemma 6.21. *Given the inputs $\{\langle u^{(i)} \rangle, \langle v^{(i)} \rangle\}_{i \in \{1, \dots, t+1\}}$ and $\{\langle a^{(i)} \rangle, \langle b^{(i)} \rangle, \langle c^{(i)} \rangle\}_{i \in \{1, \dots, t+1\}}$ where $c^{(i)} = a^{(i)} \cdot b^{(i)}$. Then except with a negligible probability, protocol $\Pi_{\text{BatchBeaver}}$ eventually outputs $\{\langle w^{(i)} \rangle\}_{i \in \{1, \dots, t+1\}}$ for all honest parties, where $w^{(i)} = u^{(i)} \cdot v^{(i)}$.*

6.6 The Asynchronous MPC Protocol

The high-level idea of our AMPC protocol $\Pi_{\text{PubKeyAMPC}}$ has already been discussed in Section 2.5. In this section, we present a protocol for securely realizing the functionality $\mathcal{F}_{\text{AMPC}}$ (Functionality 6.22), the ideal functionality for asynchronous MPC. The functionality waits for a core set $\mathcal{CR}$ of $n - t$ input providers from the adversary and function-input of the respective parties in $\mathcal{CR}$. Upon receiving these inputs, the functionality substitutes 0 as the function-input for the remaining parties, computes the output of the function f , and generates a request-based delayed output for the parties. We note that the ideal-world adversary cannot delay sending $\mathcal{CR}$ to the functionality indefinitely. This is because in the real-world AMPC protocol, the set $\mathcal{CR}$ is eventually decided, irrespective of the message scheduling by the adversary. Also note that in the functionality, it is possible that for some $P_i \notin \mathcal{C}$, the input is reset to 0, even though P_i provided its input to the functionality. Looking ahead, this models the scenario that in the real-world protocol, even if P_i is able to provide its input, P_i 's inclusion in $\mathcal{CR}$ will finally depend upon message scheduling, which is under adversarial control.

Functionality 6.22: $\mathcal{F}_{\text{AMPC}}$ — Asynchronous MPC

$\mathcal{F}_{\text{AMPC}}$ proceeds as follows, running with parties in $\mathcal{P}$ and an adversary $\mathcal{S}$ and parametrized by an n -party function $f : \mathbb{F}^n \rightarrow \mathbb{F}$. For each party P_i , initialize an input value $x^{(i)} = \perp$.

- Upon receiving a message $(\text{sid}, \text{Input}, v)$ from some $P_i \in \mathcal{P}$, if $\mathcal{CR}$ has not been recorded yet or if $P_i \in \mathcal{CR}$, set $x^{(i)} = v$.
 - Upon receiving a message $(\text{sid}, \text{CoreSet}, \mathcal{CR})$ from $\mathcal{S}$, verify that $\mathcal{CR}$ is a subset of $\mathcal{P}$ of size at least $n - t$, else ignore the message. If $\mathcal{CR}$ has not been recorded yet, then record $\mathcal{CR}$ and for every $P_i \notin \mathcal{CR}$, set $x^{(i)} = 0$.
 - If a set $\mathcal{CR}$ has been recorded and the value $x^{(i)}$ has been set to a value different from $\perp$ for every $P_i \in \mathcal{CR}$, then compute $y \stackrel{\text{def}}{=} f(x^{(1)}, \dots, x^{(n)})$ and generate a request-based delayed output $(\text{sid}, \text{Output}, (\mathcal{CR}, y))$ for every $P_i \in \mathcal{P}$.
-

We present our public-key based AMPC protocol $\Pi_{\text{PubKeyAMPC}}$ in the $(\mathcal{F}_{\text{Pre}}, \mathcal{F}_{\text{BC}}, \mathcal{F}_{\text{ZK}}, \mathcal{F}_{\text{ZK}}^{1:M})$ -hybrid model. To begin with, the parties first call the functionality $\mathcal{F}_{\text{Pre}}$ to generate the setup for THE, TSIG, polynomial commitment, and homomorphic commitment. Note that this setup is generated *only once* and can be reused across multiple instances of the protocol $\Pi_{\text{PubKeyAMPC}}$. Functionality $\mathcal{F}_{\text{Pre}}$ also generates random multiplication-triples in an encrypted-and-committed fashion.

The first phase is the input phase, where each party P_i computes $\langle x^{(i)} \rangle$, where $x^{(i)}$ is the input of P_i for the function f . Party P_i then proves the correctness of $\langle x^{(i)} \rangle$ to all via $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoE} . Every party after verifying the correctness of $\langle x^{(i)} \rangle$, gives a signature-share on $\langle x^{(i)} \rangle$ to P_i and proves the correctness of the signature-share via $\mathcal{F}_{\text{ZK}}$ for the relation R_{PoCS} . Once P_i collect $t + 1$ correct signature-shares, it computes a signature on $\langle x^{(i)} \rangle$. Next, the parties participate in an instance of Π_{ACS} , where each P_i participates with input $\langle x^{(i)} \rangle$ and a valid signature on it. The instance of Π_{ACS} will output a common subset of parties $\mathcal{CR}$ of size $n - t$, such that for every $P_j \in \mathcal{CR}$, all (honest) parties will eventually have a $\langle x^{(j)} \rangle$ and a valid signature on $\langle x^{(j)} \rangle$. The presence of a valid signature guarantees that at least one honest party has verified the correctness of $\langle x^{(j)} \rangle$, while issuing a signature-share on it. For the remaining t parties outside $\mathcal{CR}$, their input for the circuit-evaluation is taken as 0.

Next, the parties evaluate each gate in the circuit in a topological order by maintaining the following invariant: given the input(s) of the gate in an encrypted-and-committed fashion, the parties compute the output of the gate in an encrypted-and-committed fashion. Maintaining the invariant is non-interactive for linear gates, while for multiplication gates, the parties deploy the multiplication-triples. Similar to [10, 32], for the sake of efficiency, we assume that ckt is “well-spread”, so that we have sufficiently many independent multiplication gates to evaluate in parallel. Specifically, we assume that the parties can evaluate $t + 1$ multiplication gates simultaneously. Once the output gate is evaluated, the circuit output is publicly reconstructed. Note that a party cannot immediately terminate after learning the circuit output, as its participation might be needed in the instances of various primitives that are deployed in the protocol, so that the other (honest) parties also eventually complete the instantiation of those primitives and learn the function output. For the safe termination, we use the following trick: every party, after (locally) learning the circuit output, make it public through atomic broadcast. Once $t + 1$ parties make public the same circuit output, it is safely taken as the output of the protocol, and the parties terminate all sub-protocol instances.

The AMPC protocol is formally presented in Protocol 6.23.

Protocol 6.23: $\Pi_{\text{PubKeyAMPC}}$ — The public-key based AMPC protocol in the $(\mathcal{F}_{\text{Pre}}, \mathcal{F}_{\text{BC}}, \mathcal{F}_{\text{ZK}}, \mathcal{F}_{\text{ZK}}^{1:M})$ -hybrid model

Preprocessing Phase — Every $P_i \in \mathcal{P}$ does the following.

- Send Setup and $(\text{sid}, \text{Triples})$ to $\mathcal{F}_{\text{Pre}}$.¹⁶
- Request output from $\mathcal{F}_{\text{Pre}}$ until receiving $(\text{vk}, \text{ek}, \text{sk}_i, \text{dk}_i, \text{params}_1, \text{params}_2), (\text{sid}, \text{Triples}, \{(\langle a^{(l)} \rangle, \langle b^{(l)} \rangle, \langle c^{(l)} \rangle)\}_{l=1, \dots, c_M})$ from $\mathcal{F}_{\text{Pre}}$.
- Interpret received multiplication-triples $\{(\langle a^{(l)} \rangle, \langle b^{(l)} \rangle, \langle c^{(l)} \rangle)\}_{l=1, \dots, c_M}$ as $\{(\langle a^{(\ell, 1)} \rangle, \langle b^{(\ell, 1)} \rangle, \langle c^{(\ell, 1)} \rangle), \dots, (\langle a^{(\ell, t+1)} \rangle, \langle b^{(\ell, t+1)} \rangle, \langle c^{(\ell, t+1)} \rangle)\}_{\ell \in \{1, \dots, \frac{c_M}{t+1}\}}$ consisting of $\frac{c_M}{t+1}$ groups, with each group having $t + 1$ encrypted-and-committed multiplication-triples.

Input Phase — Every party $P_i \in \mathcal{P}$ executes the following code.

1. On having the input $x^{(i)}$ for the function f , randomly pick $r^{(i)}, r_1^{(i)}, r_2^{(i)}$. Compute $\text{com}_{x^{(i)}} = \text{Commit}(x^{(i)}, r^{(i)})$ and encryptions $\mathbf{c}_{x^{(i)}} = \text{Enc}_{\text{ek}}(x^{(i)}, r_1^{(i)})$ and $\mathbf{c}_{r^{(i)}} = \text{Enc}_{\text{ek}}(r^{(i)}, r_2^{(i)})$ respectively. Set $\langle x^{(i)} \rangle = (\text{com}_{x^{(i)}}, \mathbf{c}_{x^{(i)}}, \mathbf{c}_{r^{(i)}})$.
2. As a prover, send $(\text{ZK-prover}, \text{sid}_i, (\text{com}_{x^{(i)}}, \mathbf{c}_{x^{(i)}}, \mathbf{c}_{r^{(i)}}), \text{ek}), (x^{(i)}, r^{(i)}, r_1^{(i)}, r_2^{(i)})$ to $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoE} , where $\text{sid}_i = \text{sid} \parallel i$.
3. For $j = 1, \dots, n$, request output from $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoE} with session id sid_j where P_j is the prover. Upon receiving $(\text{ZK-proof}, \text{sid}_j, (\text{com}_{x^{(j)}}, \mathbf{c}_{x^{(j)}}, \mathbf{c}_{r^{(j)}}), \text{ek}))$ from $\mathcal{F}_{\text{ZK}}^{1:M}$, do the following.
 - Let $m_j = (\text{com}_{x^{(j)}}, \mathbf{c}_{x^{(j)}}, \mathbf{c}_{r^{(j)}})$. Compute $\sigma_{m_j, i} = \text{ShareSign}_{\text{sk}_i}(m_j)$ and send $(\text{sid}, \sigma_{m_j, i})$ to P_j .
 - Send $(\text{ZK-prover}, \text{sid}_{ij}, ((\text{vk}, m_j, \sigma_{m_j, i}), \text{sk}_i))$ to $\mathcal{F}_{\text{ZK}}$ for the relation R_{PoCS} , where P_i plays the role of prover and P_j plays the role of verifier and where $\text{sid}_{ij} = \text{sid} \parallel i \parallel j$.
4. Upon receiving $(\text{sid}, \sigma_{m_i, j})$ from party P_j , request output from $\mathcal{F}_{\text{ZK}}$ as a verifier where P_j plays the role of prover. Upon receiving $(\text{ZK-proof}, \text{sid}_{ji}, (\text{vk}, m_i, \sigma_{m_i, j}))$ from $\mathcal{F}_{\text{ZK}}$, include P_j to an accumulative set $\mathcal{D}_i$ (initialized to $\emptyset$). Once $|\mathcal{D}_i| \geq t + 1$, compute $\sigma_{m_i} = \text{SigCombine}(m_i, \{\sigma_{m_i, j}\}_{P_j \in \mathcal{D}_i})$.
5. Set $m_i^{\text{aug}} = (\text{com}_{x^{(i)}}, \mathbf{c}_{x^{(i)}}, \mathbf{c}_{r^{(i)}}) \parallel \sigma_{m_i}$ and participate in an instance of protocol Π_{ACS} with input m_i^{aug} , where the verification predicate V for this instance of Π_{ACS} is set as follows.

¹⁶ The message Setup is sent only once during the first instance of the protocol $\Pi_{\text{PubKeyAMPC}}$.

- For $j = 1, \dots, n$, $V(m_j^{\text{aug}}) = 1$ iff $m_j^{\text{aug}} = (\mathbf{com}_{x^{(j)}}, \mathbf{c}_{x^{(j)}}, \mathbf{c}_{r^{(j)}}) \parallel \sigma_{m_j}$ and $\text{SigVer}_{\text{vk}}((\mathbf{com}_{x^{(j)}}, \mathbf{c}_{x^{(j)}}, \mathbf{c}_{r^{(j)}}), \sigma_{m_j}) = 1$.
- 6. Let $\mathcal{CR}$ be the set of $n - t$ common parties obtained from the instance of Π_{ACS} and let $\{(\mathbf{com}_{x^{(j)}}, \mathbf{c}_{x^{(j)}}, \mathbf{c}_{r^{(j)}}) \parallel \sigma_{m_j}\}_{P_j \in \mathcal{CR}}$ be the output from the instance. For every $P_j \in \mathcal{CR}$, set $\langle x^{(j)} \rangle = (\mathbf{com}_{x^{(j)}}, \mathbf{c}_{x^{(j)}}, \mathbf{c}_{r^{(j)}})$. For every $P_j \notin \mathcal{CR}$, set $x^{(j)} = 0$ and compute a publicly-known default $\langle x^{(j)} \rangle$.

Circuit-Evaluation Phase — The parties do the following for each gate in ckt in a topological ordering.

- *Addition gate*: let u and v be the gate inputs and $w = u + v$ be the gate output. Upon having $\langle u \rangle$ and $\langle v \rangle$, locally compute $\langle w \rangle$ from $\langle u \rangle$ and $\langle v \rangle$.
- *Multiplication-with-a-constant gate with constant $c \in \mathbb{F}$* : Let u be the gate input and $v = c \cdot u$ be the gate output. Upon having $\langle u \rangle$, locally compute $\langle v \rangle$ from $\langle u \rangle$.
- *Addition-with-a-constant gate with constant $c \in \mathbb{F}$* : Let u be the gate input and $v = c + u$ be the gate output. Upon having $\langle u \rangle$, locally compute $\langle v \rangle$ from $\langle u \rangle$.
- *Multiplication gates*: Up to $t + 1$ gates are evaluated simultaneously. Let this be the ℓ^{th} batch of $t + 1$ multiplication gates, where $\ell \in \{1, \dots, \frac{c_M}{t+1}\}$. For $i = 1, \dots, t + 1$, let $u^{(\ell, i)}$ and $v^{(\ell, i)}$ be the gate inputs and let $w^{(\ell, i)} = u^{(\ell, i)} \cdot v^{(\ell, i)}$ be the gate output. Upon having $\{\langle u^{(\ell, i)} \rangle, \langle v^{(\ell, i)} \rangle\}_{i \in \{1, \dots, t+1\}}$, participate in an instance of $\Pi_{\text{BatchBeaver}}$ with inputs $\{\langle u^{(\ell, i)} \rangle, \langle v^{(\ell, i)} \rangle\}_{i \in \{1, \dots, t+1\}}$ and $\{\langle a^{(i)} \rangle, \langle b^{(i)} \rangle, \langle c^{(\ell, i)} \rangle\}_{i \in \{1, \dots, t+1\}}$ and compute $\{\langle w^{(\ell, i)} \rangle\}_{i \in \{1, \dots, t+1\}}$.

Output Computation Phase — Each $P_i \in \mathcal{P}$ does the following after computing $\mathbf{c}_y$, corresponding to the circuit output y .

- Compute $y_i := \text{Dec}_i(\mathbf{c}_y, \text{dk}_i)$.
- For $j = 1, \dots, n$, send $(\text{ZK-prover}, \text{sid}, (\mathbf{c}_y, y_i), \text{dk}_i)$ to $\mathcal{F}_{\text{ZK}}$ for the relation R_{PoCD} , where P_i is the prover and P_j is the verifier.
- For $j = 1, \dots, n$, request output from $\mathcal{F}_{\text{ZK}}$ for the relation R_{PoCD} where P_j is the prover and P_i is the verifier until receiving $(\text{ZK-proof}, \text{sid}, (\mathbf{c}_y, y_j))$. Upon receiving, include P_j to an accumulative set $\mathcal{D}_i$, initialized to $\emptyset$.
- Once $|\mathcal{D}_i| = t + 1$, compute $y = \text{Combine}(\mathbf{c}_y, \{y_j\}_{P_j \in \mathcal{D}_i})$ and send $(\text{sid}_i, \text{Output}, y)$ to $\mathcal{F}_{\text{BC}}$.

Termination Phase — Each $P_i \in \mathcal{P}$ does the following.

- For $j = 1, \dots, n$, request output from $\mathcal{F}_{\text{BC}}$ with session id sid_j where P_j is the sender until receiving $(\text{sid}_j, \text{Output}, \star)$ from $\mathcal{F}_{\text{BC}}$.
- Upon receiving $(\text{sid}_j, \text{Output}, y)$ from $\mathcal{F}_{\text{BC}}$ with session id sid_j for $t + 1$ parties P_j , output y and terminate all protocol instances.

Theorem 6.24. *Let $f : \mathbb{F}^n \rightarrow \mathbb{F}$ be a function, represented by an arithmetic circuit ckt over $\mathbb{F}$, consisting of c_M multiplication gates. Assuming the existence of a THE, TSIG and polynomial commitment scheme, protocol $\Pi_{\text{PubKeyAMPC}}$ securely realizes $\mathcal{F}_{\text{AMPC}}$ with UC security in the $(\mathcal{F}_{\text{Pre}}, \mathcal{F}_{\text{BC}})$ -hybrid model, $\mathcal{F}_{\text{ZK}}$ -hybrid model for R_{PoCD} and R_{PoCS} and $\mathcal{F}_{\text{ZK}}^{1:\text{M}}$ -hybrid model for R_{PoE} .*

The protocol incurs a communication cost of $\mathbb{P}((nc_M + n^3)\kappa) + \mathbb{B}(n\kappa)$ bits. In addition, there are $\mathcal{O}(n)$ calls to $\mathcal{F}_{\text{ZK}}^{1:\text{M}}$ for the relation R_{PoE} and $\mathcal{O}(n^2)$ calls to $\mathcal{F}_{\text{ZK}}$ for the relations R_{PoCS} and R_{PoCD} .

Proof. The number of calls to $\mathcal{F}_{\text{ZK}}^{1:\text{M}}$ and $\mathcal{F}_{\text{ZK}}$ follows easily from the protocol description. During the input phase, the parties need to invoke an instance of Π_{ACS} , where the input of each party consists of $\mathcal{O}(k)$ bits. From Lemma 6.9, this will incur a communication of $\mathbb{P}(n^3\kappa) + \mathbb{B}(n\kappa)$ bits. During the computation phase, the parties need to interact for evaluating the multiplication gates. For each batch of $t + 1$ multiplication gates, an instance of $\Pi_{\text{BatchBeaver}}$ is executed, incurring a communication of $\mathbb{P}(n^2\kappa)$ bits (Lemma 6.21). So evaluating all the c_M multiplication gates incurs a communication of $\mathbb{P}(\frac{c_M}{t+1} \cdot n^2\kappa) = \mathbb{P}(nc_M\kappa)$ bits. During the output computation phase, each party needs to send a decryption share to every other party, incurring a communication of $\mathbb{P}(n^2\kappa)$ bits. During the termination phase, each party may atomic broadcast its version of the circuit output, incurring a total communication of $\mathbb{B}(n\kappa)$ bits. This completes the derivation of communication complexity.

For security, let $\mathcal{A}$ be an arbitrary real-world adversary, attacking protocol $\Pi_{\text{PubKeyAMPC}}$, and let $\mathcal{Z}$ be an arbitrary environment. We show the existence of a simulator $\mathcal{S}_{\text{CktEval}}$, such that for any set of corrupted parties $\mathcal{C} \subset \mathcal{P}$ with $|\mathcal{C}| \leq t$ and for all inputs, the output of all honest parties and the adversary in an execution of the real protocol with $\mathcal{A}$ is indistinguishable from the output in an execution with $\mathcal{S}_{\text{CktEval}}$ involving $\mathcal{F}_{\text{AMPC}}$ in the ideal model, except with a negligible probability. This further implies that $\mathcal{Z}$ cannot distinguish between the two executions, except with a negligible probability.

The high-level idea of the simulator is as follows. During the simulated execution, the simulator itself performs the role of the ideal functionalities $\mathcal{F}_{\text{Pre}}$, $\mathcal{F}_{\text{BC}}$, $\mathcal{F}_{\text{ZK}}^{1:M}$ and $\mathcal{F}_{\text{ZK}}$ whenever required. Performing the role of $\mathcal{F}_{\text{Pre}}$ allows the simulator to learn the complete setup for THE, TSIG, commitment, and polynomial commitment scheme, apart from the encrypted-and-committed form of all the multiplication triples. During the input phase, whenever $\mathcal{A}$ sends an input in an encrypted-and-committed form to $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoE} for verification on behalf of a corrupt party, the simulator performs the verification as per $\mathcal{F}_{\text{ZK}}^{1:M}$ and accordingly records this input on behalf of the corrupt party. Notice that this allows the simulator to implicitly learn the function-input of the corresponding corrupt party. Subsequently, when $\mathcal{A}$ requests signature shares on these verified inputs via $\mathcal{F}_{\text{ZK}}$ for relation R_{PoCS} , the simulator issues valid signature shares using the honest parties' keys (which it possesses). On the other hand, for the honest parties, the simulator picks arbitrary values as their function-inputs, generates valid signatures on them using the master secret key, and simulates the steps of the honest parties as per the protocol for the input phase.

To select the common input-providers during the simulated execution, the simulator itself performs the role of $\mathcal{F}_{\text{BC}}$ and simulates the honest parties as per the steps of the protocol Π_{ACS} . It monitors the inputs sent by $\mathcal{A}$ for corrupt parties and only accepts them if they carry a valid threshold signature (which confirms that the values were verified and extracted). The simulator determines the set $\mathcal{CR}$ based on the $n - t$ valid entries on the ledger. This allows the simulator to learn the common subset of input-providers $\mathcal{CR}$, which the simulator passes to the functionality $\mathcal{F}_{\text{AMPC}}$. Notice that the function-inputs for each corrupt party in $\mathcal{CR}$ will be available with the simulator. This is because every corrupt party P_j that makes it to $\mathcal{CR}$, must have provided a valid signature to be included in the Π_{ACS} output. A valid signature implies that P_j successfully proved the correctness of its input via $\mathcal{F}_{\text{ZK}}^{1:M}$ (where the simulator recorded the input) and obtained a signature share. Hence, along with $\mathcal{CR}$, the simulator can send the corresponding function-inputs of the corrupt parties in $\mathcal{CR}$ to $\mathcal{F}_{\text{AMPC}}$.

Upon receiving the function-output y from $\mathcal{F}_{\text{AMPC}}$, the simulator simulates the steps of the honest parties for the gate evaluations as per the protocol. Finally, for the output gate, the simulator computes arbitrary decryption shares on behalf of the honest parties, which, along with the decryption shares of the corrupt parties, should decrypt the encrypted circuit output to y . This ensures that in the simulated execution, $\mathcal{A}$ learns the circuit output y . For the termination phase, the simulator plays the role of $\mathcal{F}_{\text{BC}}$ and sends y as the output on behalf of the honest parties.

Simulator 6.25: $\mathcal{S}_{\text{CktEval}}$: Simulator for the Protocol $\Pi_{\text{PubKeyAMPC}}$

$\mathcal{S}_{\text{CktEval}}$ constructs virtual real-world honest parties and invokes the real-world adversary $\mathcal{A}$. The simulator simulates the view of $\mathcal{A}$, namely its communication with $\mathcal{Z}$, the messages sent by the honest parties, and the interaction with various functionalities. In order to simulate $\mathcal{Z}$, the simulator $\mathcal{S}_{\text{CktEval}}$ forwards every message it receives from $\mathcal{Z}$ to $\mathcal{A}$ and vice-versa. Let $\mathcal{C}$ be the set of corrupt parties where $|\mathcal{C}| = t$. The simulator then simulates the various phases of the protocol as follows.

Preprocessing Phase

Simulating the call to $\mathcal{F}_{\text{Pre}}$: The simulator simulates the steps of $\mathcal{F}_{\text{Pre}}$, by itself playing the role of $\mathcal{F}_{\text{Pre}}$. Namely, upon receiving **Setup** and **Triples** from $\mathcal{A}$, the simulator does the following.

- Compute $\text{THEKeyGen}(1^\kappa, n, t) \rightarrow (\widetilde{\text{ek}}, \widetilde{\text{dk}}, \widetilde{\text{dk}}_1, \dots, \widetilde{\text{dk}}_n)$.
- Compute $\text{TSIGKeyGen}(1^\kappa, n, t) \rightarrow (\widetilde{\text{vk}}, \widetilde{\text{sk}}, \widetilde{\text{sk}}_1, \dots, \widetilde{\text{sk}}_n)$.
- Compute $\text{CommitmentGen}(1^\kappa) \rightarrow \widetilde{\text{params}}_1$.
- Compute $\text{PolynomialCommitmentGen}(1^\kappa, t) \rightarrow \widetilde{\text{params}}_2$.
- For $l = 1, \dots, c_M$, randomly pick $\widetilde{a}^{(l)}, \widetilde{b}^{(l)} \in \mathbb{F}$ and compute $\widetilde{c}^{(l)} = \widetilde{a}^{(l)} \cdot \widetilde{b}^{(l)}$.
- For $l = 1, \dots, c_M$, compute $\langle \widetilde{a}^{(l)} \rangle, \langle \widetilde{b}^{(l)} \rangle$ and $\langle \widetilde{c}^{(l)} \rangle$ by picking randomness from the underlying randomness space uniformly at random.

- Corresponding to every $P_i \in \mathcal{C}$, send $(\widetilde{\mathbf{vk}}, \widetilde{\mathbf{ek}}, \widetilde{\mathbf{sk}}_i, \widetilde{\mathbf{dk}}_i, \widetilde{\mathbf{params}}_1, \widetilde{\mathbf{params}}_2)$ and $\{(\langle \widetilde{a}^{(l)} \rangle, \langle \widetilde{b}^{(l)} \rangle, \langle \widetilde{c}^{(l)} \rangle)\}_{l \in \{1, \dots, c_M\}}$ to the adversary $\mathcal{A}$ on behalf of P_i . The multiplication-triples $\{(\langle \widetilde{a}^{(l)} \rangle, \langle \widetilde{b}^{(l)} \rangle, \langle \widetilde{c}^{(l)} \rangle)\}_{l \in \{1, \dots, c_M\}}$ are arranged as $\{(\langle \widetilde{a}^{(\ell,1)} \rangle, \langle \widetilde{b}^{(\ell,1)} \rangle, \langle \widetilde{c}^{(\ell,1)} \rangle), \dots, (\langle \widetilde{a}^{(\ell,t+1)} \rangle, \langle \widetilde{b}^{(\ell,t+1)} \rangle, \langle \widetilde{c}^{(\ell,t+1)} \rangle)\}_{\ell \in \{1, \dots, \frac{c_M}{t+1}\}}$, consisting of $\frac{c_M}{t+1}$ groups with each group having $t+1$ multiplication-triples.
- The simulator retains the master secret keys $\widetilde{\mathbf{dk}}$ and $\widetilde{\mathbf{sk}}$ and the honest parties' key shares to facilitate simulation.

Input Phase

- The simulator itself performs the role of $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoE} and also the role of $\mathcal{F}_{\text{BC}}$ during the instance of Π_{ACS} .
- The simulator simulates the operations of the honest parties during this phase, by randomly picking $\widetilde{x}^{(j)} \in \mathbb{F}$ as the input on behalf of every $P_j \notin \mathcal{C}$. Namely, for every $P_j \notin \mathcal{C}$, the simulator randomly picks $\widetilde{x}^{(j)} \in \mathbb{F}$ and randomness $\widetilde{r}^{(j)}, \widetilde{r}_1^{(j)}$ and $\widetilde{r}_2^{(j)}$ and computes $\mathbf{com}_{\widetilde{x}^{(j)}} = \text{Commit}(\widetilde{x}^{(j)}, \widetilde{r}^{(j)})$ and encryptions $\mathbf{c}_{\widetilde{x}^{(j)}} = \text{Enc}_{\widetilde{\mathbf{ek}}}(\widetilde{x}^{(j)}, \widetilde{r}_1^{(j)})$ and $\mathbf{c}_{\widetilde{r}^{(j)}} = \text{Enc}_{\widetilde{\mathbf{ek}}}(\widetilde{r}^{(j)}, \widetilde{r}_2^{(j)})$. The simulator then sets $\langle \widetilde{x}^{(j)} \rangle = (\mathbf{com}_{\widetilde{x}^{(j)}}, \mathbf{c}_{\widetilde{x}^{(j)}}, \mathbf{c}_{\widetilde{r}^{(j)}})$.
- Whenever $\mathcal{A}$ requests output from $\mathcal{F}_{\text{ZK}}^{1:M}$ with sid_j for any $P_j \notin \mathcal{C}$, the simulator responds with output $\langle \widetilde{x}^{(j)} \rangle$.
- Whenever $\mathcal{A}$ requests signature shares on honest inputs $\langle \widetilde{x}^{(j)} \rangle$ via $\mathcal{F}_{\text{ZK}}$ for relation R_{PoCS} , the simulator sends valid signature shares on behalf of honest parties.
- Whenever the adversary $\mathcal{A}$ sends (ZK-prover, $\text{sid}_i, (\mathbf{com}_{x^{(i)}}, \mathbf{c}_{x^{(i)}}, \mathbf{c}_{r^{(i)}}, \widetilde{\mathbf{ek}}), (x^{(i)}, r^{(i)}, r_1^{(i)}, r_2^{(i)})$) to $\mathcal{F}_{\text{ZK}}^{1:M}$ on behalf of any $P_i \in \mathcal{C}$ for the relation R_{PoE} where P_i is the prover, the simulator records $(x^{(i)}, r^{(i)}, r_1^{(i)}, r_2^{(i)})$ on behalf of P_i , provided $\mathbf{com}_{x^{(i)}} = \text{Commit}(x^{(i)}, r^{(i)})$, $\mathbf{c}_{x^{(i)}} = \text{Enc}_{\mathbf{ek}}(x^{(i)}, r_1^{(i)})$ and $\mathbf{c}_{r^{(i)}} = \text{Enc}_{\mathbf{ek}}(r^{(i)}, r_2^{(i)})$.
- Whenever $\mathcal{A}$ requests signature shares via $\mathcal{F}_{\text{ZK}}$ for relation R_{PoCS} on corrupt input m_i on behalf of any $P_j \notin \mathcal{C}$ (as prover), the simulator verifies m_i against the recorded values from the R_{PoE} step. If correct, it generates valid signature shares using honest keys and sends them to $\mathcal{A}$. Whenever $\mathcal{A}$ sends input to $\mathcal{F}_{\text{BC}}$ for Π_{ACS} on behalf of any $P_i \in \mathcal{C}$, the simulator plays the role of $\mathcal{F}_{\text{BC}}$ by verifying the attached threshold signature using $\widetilde{\mathbf{vk}}$. It appends the input to the ledger only if the signature is valid.
- When the first honest party completes the simulated input phase, $\mathcal{S}_{\text{CktEval}}$ learns the set $\mathcal{CR}$.

Interaction with $\mathcal{F}_{\text{AMPC}}$: Once $\mathcal{S}_{\text{CktEval}}$ learns $\mathcal{CR}$, it sends to $\mathcal{F}_{\text{AMPC}}$ inputs $x^{(i)}$ that it has recorded on the behalf of each $P_i \in \mathcal{CR} \cap \mathcal{C}$ and the set of input-providers $\mathcal{CR}$. Upon receiving back the output y from $\mathcal{F}_{\text{AMPC}}$, the simulator starts the simulation of circuit-evaluation phase.

Circuit-Evaluation Phase

The simulator simulates the evaluation of each gate in the circuit in topological order as follows.

- **Linear Gates:** Since evaluating such gates involves only local computation, the simulator does not have to simulate any messages on behalf of the honest parties. The simulator locally performs the corresponding linear operation on the underlying encrypted-and-committed gate inputs, to obtain the corresponding gate-output in an encrypted-and-committed fashion.
- **Multiplication Gates:** The simulator simulates the evaluation of each batch of $t+1$ multiplication gates. For the ℓ^{th} batch of $t+1$ multiplication gates where $\ell \in \{1, \dots, \frac{c_M}{t+1}\}$, on having $\{(\langle \widetilde{u}^{(\ell,i)} \rangle, \langle \widetilde{v}^{(\ell,i)} \rangle)\}_{i \in \{1, \dots, t+1\}}$ corresponding to the gate-inputs, the simulator takes the multiplication-triples $\{(\langle \widetilde{a}^{(\ell,i)} \rangle, \langle \widetilde{b}^{(\ell,i)} \rangle, \langle \widetilde{c}^{(\ell,i)} \rangle)\}_{i \in \{1, \dots, t+1\}}$ and computes the messages of the honest parties as per the steps of the protocol $\Pi_{\text{BatchBeaver}}$ and sends them to $\mathcal{A}$ on behalf of the honest parties.

Once the simulation of the circuit-evaluation phase is done, the simulator will know the circuit output in an encrypted and committed fashion.

Output Computation Phase

Let $\langle \widetilde{y} \rangle = (\star, \mathbf{c}_{\widetilde{y}}, \star)$ be the simulated circuit output. The simulator then does the following.

- Corresponding to every $P_i \in \mathcal{C}$, compute $\widetilde{y}_i := \text{Dec}_i(\mathbf{c}_{\widetilde{y}}, \widetilde{\mathbf{dk}}_i)$.

- For every $P_j \notin \mathcal{C}$, the simulator then computes decryption share $\tilde{y}_j$, such that $(\tilde{y}_1, \dots, \tilde{y}_n)$ forms an (n, t) threshold-secret-sharing of y .¹⁷
- Whenever $\mathcal{A}$ requests output from $\mathcal{F}_{\text{ZK}}$ for the relation R_{PoCD} with session id sid_j where P_j is the prover and P_i is the verifier for any $P_i \in \mathcal{C}$, the simulator responds with the output $(\mathbf{c}_{\tilde{y}}, \tilde{y}_j)$.

Termination Phase

If $\mathcal{A}$ requests output from $\mathcal{F}_{\text{BC}}$ on behalf of any party in $\mathcal{C}$ for any sid_j where $P_j \notin \mathcal{C}$, the simulator responds with output y .

We next prove a sequence of claims, which helps us to show that the joint distribution of the parties is identical in both the real-world, as well as simulated world, except with a negligible probability. We first claim that in any execution of $\Pi_{\text{PubKeyAMPC}}$, a set $\mathcal{CR}$ consisting of $n - t$ parties is eventually generated.

Claim 6.26. *In any execution of protocol $\Pi_{\text{PubKeyAMPC}}$, a set $\mathcal{CR}$ is eventually generated where $|\mathcal{CR}| \geq n - t$, such that for every $P_j \in \mathcal{CR}$, there exists some $x^{(j)} \in \mathbb{F}$ known to P_j for which all honest parties eventually hold $\langle x^{(j)} \rangle$.*

Proof. We first note that there always exist at least $n - t$ parties, namely the set of honest parties, who will correctly generate their input in an encrypted-and-committed fashion via $\mathcal{F}_{\text{BC}}$, prove the correctness of their input via $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoE} , collect $t + 1$ signature shares to generate a valid threshold signature, and participate in the instance of Π_{ACS} . So even if the honest parties do not receive any output from the instances of $\mathcal{F}_{\text{BC}}$ and $\mathcal{F}_{\text{ZK}}^{1:M}$ which are supposed to be invoked by the potentially corrupt parties, from the liveness property of Π_{ACS} (Lemma 6.9), a $\mathcal{CR}$ set of size $n - t$ is guaranteed to be generated eventually.

Now consider an arbitrary $P_j \in \mathcal{CR}$. Since $P_j \in \mathcal{CR}$, it follows that P_j 's input m_j^{aug} (containing $\langle x^{(j)} \rangle$ and the signature σ_{m_j}) was included in the output of the Π_{ACS} . By the validity property of Π_{ACS} , this implies that the attached threshold signature σ_{m_j} is valid. A valid threshold signature implies that at least $t + 1$ signature shares were combined. Since there are at most t corrupt parties, at least one signature share must have been generated by an honest party. An honest party issues a signature share only if it successfully verifies the proof provided by P_j via $\mathcal{F}_{\text{ZK}}^{1:M}$ for the relation R_{PoE} . It then follows from the properties of $\mathcal{F}_{\text{ZK}}^{1:M}$ that P_j would have provided witnesses $(x^{(j)}, \star, \star, \star)$ to $\mathcal{F}_{\text{ZK}}^{1:M}$ for verification and hence P_j knows the input $x^{(j)}$ as claimed. Furthermore, the agreement property of Π_{ACS} ensures that the encrypted-and-committed value $\langle x^{(j)} \rangle$ is eventually held by all honest parties. $\square$

We next show that the view generated by $\mathcal{S}_{\text{CktEval}}$ for $\mathcal{A}$ is indistinguishable from $\mathcal{A}$'s view during the real execution of $\Pi_{\text{PubKeyAMPC}}$, except with a negligible probability.

Claim 6.27. *The view of $\mathcal{A}$ in the simulated execution with $\mathcal{S}_{\text{CktEval}}$ is indistinguishable from the view of $\mathcal{A}$ in the real execution of $\Pi_{\text{PubKeyAMPC}}$, except with a negligible probability.*

Proof. It is easy to see that the view of $\mathcal{A}$ during the preprocessing phase is identically distributed in both the executions. This is because in both the executions, $\mathcal{A}$ receives no messages from the honest parties, and the steps of $\mathcal{F}_{\text{Pre}}$ are executed by the simulator itself in the simulated execution. Namely, in both the executions, $\mathcal{A}$'s view consists of the setup data for THE, TSig, commitment scheme, and polynomial commitment scheme, and c_M random multiplication-triples in an encrypted-and-committed fashion. So let us fix the data held by $\mathcal{A}$ at the end of the preprocessing phase in both the executions.

Now conditioned on the view of $\mathcal{A}$ for the preprocessing phase, for the input phase, $\mathcal{A}$ learns $\langle x^{(j)} \rangle$ for every $P_j \notin \mathcal{C}$ during the real execution, while in the simulated execution it learns $\langle \tilde{x}^{(j)} \rangle$ for every $P_j \notin \mathcal{C}$. It then follows from the semantic security of THE and the hiding property of the commitment scheme that the distributions of $\{\langle x^{(j)} \rangle\}_{P_j \notin \mathcal{C}}$ and $\{\langle \tilde{x}^{(j)} \rangle\}_{P_j \notin \mathcal{C}}$ are indistinguishable, except with a negligible probability. So let us fix these encrypted-and-committed values. In the real execution, $\mathcal{A}$ sees honest parties exchanging valid signed inputs to $\mathcal{F}_{\text{BC}}$. In the simulated execution, the simulator plays the role of $\mathcal{F}_{\text{ZK}}$ for the relation R_{PoCS} and $\mathcal{F}_{\text{BC}}$ for the protocol Π_{ACS} . Since the simulator retains the honest parties' signing keys (from the setup), it generates valid signature shares and threshold signatures on behalf of the

¹⁷ For instance, in the (n, t) threshold-secret-sharing scheme of Shamir [72], fix the points corresponding to the decryption shares $\{\tilde{y}_i\}_{P_i \in \mathcal{C}}$ and the final output y , compute a t -degree polynomial interpolating these points, and compute the decryption shares $\{\tilde{y}_j\}_{P_j \notin \mathcal{C}}$ accordingly.

honest parties. Furthermore the simulator plays the role of $\mathcal{F}_{\text{BC}}$ by enforcing the same verification logic as the honest parties would. Consequently, the view of $\mathcal{A}$ during the selection of the set $\mathcal{CR}$ is identically distributed in both the real as well as the simulated execution, except with a negligible probability.

Next, consider the circuit-evaluation phase. During the evaluation of linear gates, no communication is involved among the parties. During the evaluation of multiplication gates, in the simulated execution, the simulator will know the gate-inputs in an encrypted-and-committed fashion and also the associated multiplication-triples. Hence, the simulator can completely simulate all the messages that are supposed to be sent on behalf of the honest parties during the instances of $\Pi_{\text{BatchBeaver}}$. This ensures that $\mathcal{A}$'s view during the evaluation of multiplication gates is identically distributed, both in the real execution and the simulated execution.

For the output gate, the decryption shares of $\mathbf{c}_y$ received by $\mathcal{A}$ in the real execution from the honest parties correspond to an (n, t) -threshold-secret-sharing of the circuit-output y . From the steps of $\mathcal{S}_{\text{CktEval}}$, it is easy to see that the same holds even in the simulated execution, as $\mathcal{S}_{\text{CktEval}}$ sends to $\mathcal{A}$ decryption shares of $\mathbf{c}_{\tilde{y}}$, which along with the decryption shares held by $\mathcal{A}$ correspond to an (n, t) -threshold-secret-sharing of y . Moreover, from the semantic security of THE, the probability distribution of $\mathbf{c}_y$ is indistinguishable from the distribution of $\mathbf{c}_{\tilde{y}}$, except with a negligible probability.

Finally, it is easy to see that $\mathcal{A}$'s view is identically distributed in both the executions during the termination phase. This is because in both the executions, $\mathcal{A}$ receives y as the output from $\mathcal{F}_{\text{BC}}$ with id sid_j for every $P_j \notin \mathcal{C}$. $\square$

We next claim that conditioned on the view of $\mathcal{A}$, the output of the honest parties is identically distributed in both worlds, except with a negligible probability.

Claim 6.28. *Conditioned on the view of $\mathcal{A}$, the output of the honest parties is identically distributed in the real execution of $\Pi_{\text{PubKeyAMPC}}$ involving $\mathcal{A}$, as well as in the ideal execution involving $\mathcal{S}_{\text{CktEval}}$ and $\mathcal{F}_{\text{AMPC}}$, except with a negligible probability.*

Proof. Let View be an arbitrary view of $\mathcal{A}$. And let $\mathcal{CR}$ be the set of input-providers as determined by View (from Claim 6.26, such a set $\mathcal{CR}$ is bound to exist). Moreover, according to View , for every $P_j \in \mathcal{CR}$, there exists some input $x^{(j)} \in \mathbb{F}$, such that the (honest) parties eventually hold $\langle x^{(j)} \rangle$. Furthermore, from Claim 6.27, if $P_j \in \mathcal{C}$, then the corresponding $x^{(j)}$ and the underlying randomness used in $\langle x^{(j)} \rangle$ is included in View . On the other hand, for $P_j \notin \mathcal{C}$, the corresponding $x^{(j)}$ is random, conditioned on $\langle x^{(j)} \rangle$ available with $\mathcal{A}$ as determined by View . Let us fix the $x^{(j)}$ values corresponding to the parties in $\mathcal{CR}$ and denote by $\vec{x}$ the vector of values $x^{(j)}$, where $x^{(j)} = 0$, if $P_j \notin \mathcal{CR}$.

It is easy to see that in the ideal world, the output of the honest parties is y , where $y \stackrel{\text{def}}{=} f(\vec{x})$. This is because $\mathcal{S}_{\text{CktEval}}$ provides the identity of $\mathcal{CR}$ along with the inputs $x^{(i)}$ corresponding to $P_i \in (\mathcal{CR} \cap \mathcal{C})$ to $\mathcal{F}_{\text{AMPC}}$. We now show that the honest parties output y even in the real world, except with a negligible probability. For this, we argue that all the values during the circuit-evaluation phase of the protocol are available to the parties in an encrypted-and-committed fashion.

Since the evaluation of linear gates needs only local computation, it follows that the output of the linear gates will be correctly available in an encrypted-and-committed fashion. During the evaluation of each batch of multiplication gates, the honest parties will run an instance of $\Pi_{\text{BatchBeaver}}$, using the multiplication-triples in an encrypted-and-committed fashion, distributed by $\mathcal{F}_{\text{Pre}}$. It then follows from Lemma 6.21, that the output of each batch of multiplication gates will be correctly available in an encrypted-and-committed fashion, except with a negligible probability.

For the output gate, the decryption share of each honest party will be eventually delivered by $\mathcal{F}_{\text{ZK}}$ functionality for the relation R_{PoCD} to every honest party. On the other hand, if $\mathcal{F}_{\text{ZK}}$ for the relation R_{PoCD} delivers any decryption-share on behalf of a corrupt party, then it is correct. As there are at least $t + 1$ honest parties, it follows that eventually every honest party will receive a sufficient number of decryption shares to decrypt $\mathbf{c}_y$ and compute y . Hence, during the termination phase, every honest party will eventually send y via $\mathcal{F}_{\text{BC}}$, while the corrupt parties may either send no output message or $y' \neq y$. Since there can be at most t corrupt parties, it follows that no honest party ever outputs $y' \neq y$. On the other hand, the output message for y sent by all honest parties will be eventually delivered, and so all honest parties eventually output y . $\square$

From Claim 6.28 and Claim 6.27, we can conclude that the following holds, hence proving the theorem:

$$\left\{ \text{HYBRID}_{\Pi_{\text{PubKeyAMPC}}, \mathcal{A}(z)}^{\mathcal{F}_{\text{Pre}}, \mathcal{F}_{\text{BC}}, \mathcal{F}_{\text{ZK}}, \mathcal{F}_{\text{ZK}}^{1:M}}(\vec{x}) \right\}_{z \in \{0,1\}^*, \vec{x} \in \mathbb{F}^n} \equiv \left\{ \text{IDEAL}_{\mathcal{F}_{\text{AMPC}}, \mathcal{S}_{\text{CktEval}}(z)}(\vec{x}) \right\}_{z \in \{0,1\}^*, \vec{x} \in \mathbb{F}^n}$$

□

From Corollary 6.17, we can use protocol Π_{Pre} in the $(\mathcal{F}_{\text{Setup}}, \mathcal{F}_{\text{BC}})$ -hybrid model to securely realize $\mathcal{F}_{\text{Pre}}$. Moreover, from Theorem 6.6, we can securely realize the calls to $\mathcal{F}_{\text{ZK}}^{1:M}$ assuming a setup for TSIG, which is made available through $\mathcal{F}_{\text{Setup}}$. Consequently, we get the following corollary to Theorem 6.24.

Corollary 6.29. *Assuming the existence of a THE, TSIG, and polynomial commitment scheme, protocol $\Pi_{\text{PubKeyAMPC}}$ securely realizes $\mathcal{F}_{\text{AMPC}}$ with UC security in the $(\mathcal{F}_{\text{Setup}}, \mathcal{F}_{\text{BC}})$ -hybrid model. The protocol incurs a communication of $\mathbb{P}((n^2 c_M + n^3)\kappa) + \mathbb{B}(n\kappa)$ bits, provided $c_M \kappa \geq (t + 1) \log n$.*

Acknowledgments

Arpita Patra is supported by the Centre of Excellence in Cybersecurity (CySecK - FSID). This work is supported in part by a research gift from Supra Labs and a Stellar Development Foundation Academic Research Award.

References

1. Abraham, I., Asharov, G., Patil, S., Patra, A.: Perfect asynchronous MPC with linear communication overhead. In: Advances in Cryptology - EUROCRYPT. pp. 280–309 (2024)
2. "aptos: The foundation for a new digital economy". <https://aptosnetwork.com/> (2022)
3. Asharov, G., Lindell, Y.: A Full Proof of the BGW Protocol for Perfectly Secure Multiparty Computation. J. Cryptol. **30**(1), 58–151 (2017)
4. Attiya, H., Welch, J.L.: Distributed computing - fundamentals, simulations, and advanced topics (2. ed.). Wiley series on parallel and distributed computing, Wiley (2004)
5. Backes, M., Bendun, F., Choudhury, A., Kate, A.: Asynchronous MPC with a Strict Honest Majority Using Non-equivocation. In: PODC. pp. 10–19. ACM (2014)
6. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: A composable treatment. In: Advances in Cryptology - CRYPTO 2017. pp. 324–356 (2017). https://doi.org/10.1007/978-3-319-63688-7_11
7. Bandrupalli, A., Ji, X., Kate, A., Liu-Zhang, C.D., Pöllmann, D., Song, Y.: Velox: Scalable fair asynchronous MPC from lightweight cryptography. In: Huang, C.Y., Chen, J.C., Shieh, S.P., Lie, D., Cortier, V. (eds.) ACM CCS 2025: 32nd Conference on Computer and Communications Security. pp. 1799–1813. ACM Press, Taipei, Taiwan (Oct 13–17, 2025). <https://doi.org/10.1145/3719027.3765114>
8. Bandrupalli, A., Ji, X., Kate, A., Liu-Zhang, C., Song, Y.: Computationally efficient asynchronous MPC with linear communication and low additive overhead. In: Advances in Cryptology - CRYPTO. pp. 261–294 (2025)
9. Beaver, D.: Efficient Multiparty Protocols Using Circuit Randomization. In: CRYPTO. Lecture Notes in Computer Science, vol. 576, pp. 420–432. Springer (1991)
10. Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-Secure MPC with Linear Communication Complexity. In: TCC. Lecture Notes in Computer Science, vol. 4948, pp. 213–230. Springer (2008)
11. Beerliová-Trubíniová, Z., Hirt, M., Nielsen, J.B.: On the theoretical gap between synchronous and asynchronous mpc protocols. In: Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing. p. 211–218. PODC '10, Association for Computing Machinery, New York, NY, USA (2010). <https://doi.org/10.1145/1835698.1835746>
12. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th Annual ACM Symposium on Theory of Computing. pp. 1–10. ACM Press, Chicago, IL, USA (May 2–4, 1988). <https://doi.org/10.1145/62212.62213>
13. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Advances in Cryptology - CRYPTO. pp. 421–439 (2014)
14. Bracha, G.: An asynchronous $[(n - 1)/3]$ -resilient consensus protocol. In: Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing. p. 154–162. PODC '84, Association for Computing Machinery, New York, NY, USA (1984). <https://doi.org/10.1145/800222.806743>, <https://doi.org/10.1145/800222.806743>
15. Cachin, C., Guerraoui, R., Rodrigues, L.: Introduction to Reliable and Secure Distributed Programming. Springer Publishing Company, Incorporated, 2nd edn. (2011)

16. Cachin, C., Tessaro, S.: Asynchronous verifiable information dispersal. In: Fraigniaud, P. (ed.) Distributed Computing, 19th International Conference, DISC 2005, Cracow, Poland, September 26-29, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3724, pp. 503–504. Springer (2005). https://doi.org/10.1007/11561927_42
17. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. In: FOCS. pp. 136–145. IEEE Computer Society (2001)
18. Canetti, R.: Universally Composable Security. J. ACM **67**(5), 28:1–28:94 (2020)
19. Canetti, R.: Studies in Secure Multiparty Computation and Applications. PhD thesis, Weizmann Institute of Science (1996)
20. "chainlink 2.0: Next steps in the evolution of decentralized oracle networks". <https://research.chain.link/whitepaper-v2.pdf> (2020)
21. Chakka, P., Joshi, S., Kate, A., Tobkin, J., Yang, D.: Oracle agreement: From an honest super majority to simple majority. In: 43rd IEEE International Conference on Distributed Computing Systems, ICDCS. pp. 714–725 (2023)
22. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1825–1842. ACM (2017)
23. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th Annual ACM Symposium on Theory of Computing. pp. 11–19. ACM Press, Chicago, IL, USA (May 2–4, 1988). <https://doi.org/10.1145/62212.62214>
24. Chopard, A., Hirt, M., Liu-Zhang, C.: On communication-efficient asynchronous MPC with adaptive security. In: Theory of Cryptography - TCC. pp. 35–65 (2021)
25. Choudhury, A., Patra, A.: On the Communication Efficiency of Statistically Secure Asynchronous MPC with Optimal Resilience. J. Cryptol. **36**(2), 13 (2023)
26. Choudhury, A., Patra, A.: An efficient framework for unconditionally secure multiparty computation. IEEE Trans. Inf. Theory **63**(1), 428–468 (2017)
27. Cohen, R.: Asynchronous Secure Multiparty Computation in Constant Time. In: PKC. Lecture Notes in Computer Science, vol. 9615, pp. 183–207. Springer (2016)
28. Coretti, S., Garay, J.A., Hirt, M., Zikas, V.: Constant-Round Asynchronous Multi-Party Computation Based on One-Way Functions. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 10032, pp. 998–1021 (2016)
29. Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley Series in Telecommunications and Signal Processing, Wiley-Interscience, Hoboken, NJ, second edn. (2006)
30. Crites, E.C., Komlo, C., Maller, M.: Fully Adaptive Schnorr Threshold Signatures. In: CRYPTO. Lecture Notes in Computer Science, vol. 14081, pp. 678–709. Springer (2023)
31. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: IEEE Symposium on Security and Privacy (SP). pp. 910–927 (2020)
32. Damgård, I., Nielsen, J.B.: Scalable and Unconditionally Secure Multiparty Computation. In: CRYPTO. Lecture Notes in Computer Science, vol. 4622, pp. 572–590. Springer (2007)
33. Damgård, I.B., Kamp, S.H., Loss, J., Nielsen, J.B.: Asynchronous YOSO a la paillier. Cryptology ePrint Archive, Paper 2025/128 (2025), <https://eprint.iacr.org/2025/128>
34. Danezis, G., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A.: Narwhal and tusk: a dag-based mempool and efficient bft consensus. In: Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys). p. 34–50 (2022)
35. David, B., Magri, B., Matt, C., Nielsen, J.B., Tschudi, D.: GearBox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022: 29th Conference on Computer and Communications Security. pp. 683–696. ACM Press, Los Angeles, CA, USA (Nov 7–11, 2022). <https://doi.org/10.1145/3548606.3559375>
36. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (Apr 1988). <https://doi.org/10.1145/42282.42283>
37. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (Apr 1985)
38. Fitzi, M., Hirt, M.: Optimally efficient multi-valued byzantine agreement. In: PODC. pp. 163–168. ACM (2006)
39. Garay, J.A., Kiayias, A., Leonardos, N.: The Bitcoin Backbone Protocol: Analysis and Applications. J. ACM **71**(4), 25:1–25:49 (2024)
40. Garg, S., Kate, A., Mukherjee, P., Sinha, R., Sridhar, S.: Insta-pok3r: Real-time poker on blockchain. Cryptology ePrint Archive, Paper 2024/1061 (2024), <https://eprint.iacr.org/2024/1061>
41. Gelashvili, R., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A., Xiang, Z.: Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In: Financial Cryptography and Data Security - 26th International Conference, FC. pp. 296–315 (2022)
42. Ghinea, D., Liu-Zhang, C.D., Wattenhofer, R.: Communication-optimal convex agreement. In: ACM PODC. p. 39–49 (2025)

43. Goldreich, O.: The Foundations of Cryptography - Volume 2: Basic Applications. Cambridge University Press (2004)
44. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th Annual ACM Symposium on Theory of Computing. pp. 218–229. ACM Press, New York City, NY, USA (May 25–27, 1987). <https://doi.org/10.1145/28395.28420>
45. Goyal, V., Kothapalli, A., Masserova, E., Parno, B., Song, Y.: Storing and retrieving secrets on a blockchain. In: Public-Key Cryptography – PKC 2022: 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8–11, 2022, Proceedings, Part I. p. 252–282. Springer-Verlag, Berlin, Heidelberg (2022)
46. Goyal, V., Liu-Zhang, C., Song, Y.: Towards achieving asynchronous MPC with linear communication and optimal resilience. In: Advances in Cryptology - CRYPTO. pp. 170–206 (2024)
47. Goyal, V., Masserova, E., Parno, B., Song, Y.: Blockchains enable non-interactive mpc. In: Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part II. p. 162–193. Springer-Verlag, Berlin, Heidelberg (2021)
48. Goyal, V., Raizes, J., Soni, P.: Time-Traveling Simulators Using Blockchains and Their Applications. In: Braverman, M. (ed.) 13th Innovations in Theoretical Computer Science Conference (ITCS 2022). Leibniz International Proceedings in Informatics (LIPIcs), vol. 215, pp. 81:1–81:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2022). <https://doi.org/10.4230/LIPIcs.ITCS.2022.81>, <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITCS.2022.81>
49. Graf, M., Rausch, D., Ronge, V., Egger, C., Küsters, R., Schröder, D.: A security framework for distributed ledgers. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021: 28th Conference on Computer and Communications Security. pp. 1043–1064. ACM Press, Virtual Event, Republic of Korea (Nov 15–19, 2021). <https://doi.org/10.1145/3460120.3485362>
50. Groth, J., Shoup, V.: Design and analysis of a distributed ECDSA signing service. IACR Cryptol. ePrint Arch. p. 506 (2022)
51. Hirt, M., Nielsen, J.B., Przydatek, B.: Asynchronous Multi-Party Computation with Quadratic Communication. In: ICALP. Lecture Notes in Computer Science, vol. 5126, pp. 473–485. Springer (2008)
52. Hofheinz, D., Müller-Quade, J.: Universally composable commitments using random oracles. In: Naor, M. (ed.) Theory of Cryptography. pp. 58–76. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
53. Impagliazzo, R., Rudich, S.: Limits on the provable consequences of one-way permutations. In: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing. p. 44–61. STOC '89, Association for Computing Machinery, New York, NY, USA (1989). <https://doi.org/10.1145/73007.73012>
54. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-Size Commitments to Polynomials and Their Applications. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 6477, pp. 177–194. Springer (2010)
55. Kate, A., Mukherjee, P., Sarkar, P., Saleem, H., Shrestha, N., Yang, D.: Scalable distributed key generation for blockchains. Cryptology ePrint Archive, Paper 2026/072 (2026)
56. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: TCC. Lecture Notes in Computer Science, vol. 7785, pp. 477–498. Springer (2013)
57. Keidar, I., Naor, O., Poupko, O., Shapiro, E.: Cordial Miners: Fast and Efficient Consensus for Every Eventuality. In: 37th International Symposium on Distributed Computing (DISC 2023). pp. 26:1–26:22 (2023)
58. Kokoris-Kogias, E., Malkhi, D., Spiegelman, A.: Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020: 27th Conference on Computer and Communications Security. pp. 1751–1767. ACM Press, Virtual Event, USA (Nov 9–13, 2020). <https://doi.org/10.1145/3372297.3423364>
59. Lab, H.: "a complete history of solana outages: Causes, fixes, and lessons learnt". <https://www.helius.dev/blog/solana-outages-complete-history> (2025)
60. Labs, A.: "roll with move: Secure, instant randomness on aptos". <https://medium.com/aptoslabs/roll-with-move-secure-instant-randomness-on-aptos-c0e219df3fb1> (2024)
61. Lu, D., Yurek, T., Kulshreshtha, S., Govind, R., Kate, A., Miller, A.: Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In: ACM SIGSAC CCS. p. 887–903 (2019)
62. Momose, A.: Practical asynchronous MPC from lightweight cryptography. Cryptology ePrint Archive, Paper 2024/1717 (2024)
63. Momose, A., Ren, L.: Multi-threshold byzantine fault tolerance. In: ACM SIGSAC CCS. pp. 1686–1699 (2021)
64. Nazirkhanova, K., Neu, J., Tse, D.: Information dispersal with provable retrievability for rollups. In: ACM Conference on Advances in Financial Technologies, AFT. pp. 180–197. ACM (2022)
65. Okwatch, L.: "sui blames consensus bug for jan. 14 six-hour network outage". <https://crypto.news/sui-consensus-bug-six-hour-network-outage-2025/> (2026)
66. Pan, H., Tuglu, J., Zhou, N., Wang, T., Shen, Y., Zheng, X., Tassarotti, J., Tseng, L., Palmieri, R.: Rabia: Simplifying state-machine replication through randomization. In: SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles. pp. 472–487 (2021)

67. Pass, R., Seeman, L., Shelat, A.: Analysis of the Blockchain Protocol in Asynchronous Networks. In: EURO-CRYPT. pp. 643–673. Lecture Notes in Computer Science (2017)
68. Patra, A., Choudhury, A., Rangan, C.P.: Efficient asynchronous verifiable secret sharing and multiparty computation. *J. Cryptol.* **28**(1), 49–109 (2015)
69. Rabin, M.O.: How To Exchange Secrets with Oblivious Transfer. Cryptology ePrint Archive, Paper 2005/187 (2005)
70. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: 21st Annual ACM Symposium on Theory of Computing. pp. 73–85. ACM Press, Seattle, WA, USA (May 15–17, 1989). <https://doi.org/10.1145/73007.73014>
71. Rai Choudhuri, A., Goyal, V., Jain, A.: Founding secure computation on blockchains. In: Advances in Cryptology – EUROCRYPT 2019. Lecture Notes in Computer Science, vol. 11476, pp. 351–380. Springer (2019)
72. Shamir, A.: How to Share a Secret. *Commun. ACM* **22**(11), 612–613 (1979)
73. "shelby: Decentralized storage designed to serve". <https://shelby.xyz/whitepaper.pdf> (2025)
74. "solana: The capital market for every asset on earth". <https://solana.com/> (2020)
75. Su, Y., Lu, Y., Li, J., Wang, Y., Dong, C., Tang, Q.: Dumbo-MPC: Efficient fully asynchronous MPC with optimal resilience. In: Bauer, L., Pellegrino, G. (eds.) USENIX Security 2025: 34th USENIX Security Symposium. pp. 1925–1944. USENIX Association, Seattle, WA, USA (Aug 13–15, 2025), <https://www.usenix.org/conference/usenixsecurity25/presentation/su-yuan>
76. "sui | the full stack for a new global economy". <https://www.sui.io/> (2023)
77. Team, I.: "inco | confidentiality layer of web3 ". <https://github.com/Inco-fhevm/confidential-erc20-framework> (2025)
78. Team, Z.: "zama confidential blockchain protocol litepaper". <https://docs.zama.org/protocol/zama-protocol-litepaper> (2025)
79. Tennage, P., Basescu, C., Kokoris-Kogias, L., Syta, E., Jovanovic, P., Estrada-Galinanes, V., Ford, B.: Quepaxa: Escaping the tyranny of timeouts in consensus. In: Proceedings of the 29th Symposium on Operating Systems Principles (SOSP). p. 281–297 (2023)
80. "walrus | enabling data markets for the ai era". <https://www.walrus.xyz/> (2025)
81. Wong, H.W.H., Ma, J.P.K., Chow, S.S.M.: Secure Multiparty Computation of Threshold Signatures Made More Efficient. In: NDSS. The Internet Society (2024)
82. Yao, A.C.C.: Theory and applications of trapdoor functions (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science. pp. 80–91. IEEE Computer Society Press, Chicago, Illinois (Nov 3–5, 1982). <https://doi.org/10.1109/SFCS.1982.45>