# Network-Agnostic Multi-party Computation Revisited (Extended Abstract)

Nidhish Bhimrajka, Ashish Choudhury(✉), and Supreeth Varadarajan

International Institute of Information Technology Bangalore, Bengaluru, India
{nidhish.bhimrajka,ashish.choudhury,supreeth.varadarajan}@iiitb.ac.in

**Abstract.** We study network-agnostic *secure multi-party computation* (MPC) in the presence of *computationally-bounded* adversaries. A network-agnostic protocol provides the best possible security guarantees, irrespective of the type of underlying communication network. Previous MPC protocols in this regime either assume a setup for a common reference string (CRS) and a threshold additively homomorphic encryption (Blum et al. CRYPTO 2020) or a plain public-key infrastructure (PKI) setup (Bacho et al. CRYPTO 2023). Both these MPC protocols perform circuit-evaluation over encrypted data and also deploy different forms of zero-knowledge (ZK) proofs, along with other computationally-expensive cryptographic machinery. We aim to build an MPC protocol based on circuit evaluation on secret-shared data, *avoiding* ZK proofs and other computationally-expensive cryptographic machinery and based on a *plain* PKI setup.

To achieve our goal, we present the *first* network-agnostic *verifiable secret sharing* (VSS) protocol with the *optimal* threshold conditions, which is of independent interest. Previously, network-agnostic VSS is known either with *perfect* security (Appan et al. IEEE IT 2023) where the threshold conditions are *not* known to be optimal or with *statistical security* (Appan et al. TCC 2023) where the threshold conditions are optimal, but the parties need to perform *exponential* amount of computation and communication. Although our proposed MPC protocol incurs higher communication complexity compared to state-of-the-art network-agnostic MPC protocols, it offers valuable insights and motivates alternative directions for designing *computationally inexpensive* MPC protocols, based on a plain PKI setup, which has not been explored in the domain of network-agnostic MPC.

**Keywords:** Cryptographic security · Multiparty Computation · Verifiable Secret Sharing · Network-agnostic · Public Key Infrastructure

This research is an outcome of the R&D work undertaken in the project under the Visvesvaraya PhD Scheme of the Ministry of Electronics & Information Technology, Government of India, being implemented by Digital India Corporation.


Q. Tang and V. Teague (Eds.): PKC 2024, LNCS 14602, pp. 171–204, 2024.
https://doi.org/10.1007/978-3-031-57722-2_6

## 1 Introduction

A *secure multi-party computation* (MPC) protocol [11,32,39,41] allows a set of $n$ mutually distrusting parties $\mathcal{P} = \{P_1, ..., P_n\}$ with private inputs, to securely compute any function of their inputs, without revealing any additional information, even if a subset of the parties are under the control of an adversary who may behave arbitrarily during the protocol execution. In any MPC protocol, the parties exchange messages over an underlying communication network and the network behaviour is assumed to be *known beforehand* to the parties. The more popular synchronous MPC (SMPC) protocols are designed for a *synchronous* network where the local clocks of the parties are assumed to be synchronized and where there is a *publicly-known* upper bound on message delays. Unfortunately, the security of such protocols breaks down completely even if a single expected message fails to get delivered within the expected time-out. To deal with this shortcoming, there is another category of MPC protocols, designed for the *asynchronous* network model [9,10,37], where no assumption is made on the message delays and where the messages can be delayed arbitrarily yet finitely, with the guarantee that every message being sent is delivered *eventually*. Apart from better modelling of real-world networks like the Internet, *asynchronous* MPC (AMPC) protocols also have the advantage of running at the "actual" speed of the underlying network. The downside is that AMPC protocols are more complex, since in the *absence* of any known time-outs, the parties do not know how long to wait for an expected message and waiting for messages from all the parties may turn out to be an endless wait. Consequently, as soon a party receives messages from a "subset" of parties, it has to process them and in the process, messages from a subset of "slow" but potentially honest parties may get ignored. Moreover, the *resilience* (namely the maximum number of allowed corruptions) of AMPC protocols is poor compared to SMPC protocols. A very recent and highly practically-motivated category of MPC protocols is that of *network-agnostic* protocols [3,5,15,27], where the parties *need not* know the behaviour of the underlying network and which provides the best possible security guarantees, irrespective of the behaviour of the network.

**Our Motivation:** In this work, we focus on network-agnostic MPC protocols with *cryptographic* security, where the adversary is assumed to be *computationally bounded*. Let $t_s$ and $t_a$ be the maximum number of parties which can be corrupted by the adversary in the *synchronous* and *asynchronous* network respectively, where $t_s < \frac{n}{2}$ and $t_a < \frac{n}{3}$.[1] The pioneering work of Blum et al. [15] has shown that network-agnostic MPC protocols with cryptographic security is possible only if $t_a + 2t_s < n$ is satisfied. They also present a network-agnostic MPC protocol with the condition $t_a + 2t_s < n$. The work of Blum et al. is followed by the work of Deligios et al. [27] and Bacho et al. [5], who also present

[1] The conditions $t_s < \frac{n}{2}$ and $t_a < \frac{n}{3}$ are necessary and sufficient for cryptographically-secure SMPC [23] and AMPC protocols [33] respectively with full security. By full security, we mean that the honest parties always get the correct output.

network-agnostic MPC protocols with the condition $t_a + 2t_s < n$. All these protocols let the parties jointly perform secure *circuit-evaluation*. Namely, in all these protocols the function to be securely computed is abstracted as an arithmetic circuit over some algebraic structure, which could be a ring or a field and then the parties securely and jointly evaluate each gate in the circuit. The evaluation happens in such a way that the adversary does not learn any additional information about the inputs of the honest parties, beyond what can be inferred from the input and output of the corrupt parties. These protocols can be broadly classified into *two* categories.

- **Threshold-Encryption Based Approach**: This approach was first pioneered for SMPC protocols in [23] and later also used for AMPC protocols in [20,33,34]. The network-agnostic protocols of [5,15] follow this approach. Here the circuit-evaluation happens over encrypted values, where each value during the evaluation remains encrypted under some threshold linearly-homomorphic encryption scheme. Informally, it is a special form of public-key encryption scheme, where the encryption key is *publicly available*, but the decryption key remains secret-shared among the parties, with each party holding a *private* secret-share of the decryption key. Such a setup (of *public* encryption key and *private* decryption-key shares) is assumed to be *already* available to the parties, through some *trusted* entity. Doing circuit-evaluation using this approach is computationally expensive and apart from the expensive setup of threshold encryption, the protocols deploy other heavy cryptographic machinery such as zero-knowledge (ZK) proofs, which are used to prove the "correctness" of each message exchanged by the parties during the circuit evaluation.
  The protocol of [15] deploys threshold homomorphic encryption and also assumes a *trusted* setup for a *common reference string* (CRS). The CRS is utilized to instantiate *non-interactive* ZK (NIZK) proofs for various tasks. The protocol of [5] also deploys threshold homomorphic encryption and NIZK proofs, but gets rid of the CRS setup assumption and instead assumes a *plain* public-key infrastructure (PKI) set-up. The number of communication rounds in these protocols is proportional to the multiplicative depth of the underlying circuit.
- **Garbled-Circuit Based Approach**: This approach was first pioneered for SMPC protocols in [8,25,41] and later for AMPC protocols in [22]. The network-agnostic protocol of [27] uses this approach. This approach yields a *constant* round protocol. To deploy this approach, the protocol of [27] requires a *trusted* setup of a threshold homomorphic encryption scheme. Additionally, the protocol also uses ZK proofs.

Evidently, the known protocols assume strong setup assumptions such as threshold homomorphic encryption and deploy computationally-expensive cryptographic machinery, such as ZK proofs, to name a few. Apart from the above two approaches for secure circuit-evaluation, another well-known approach is that of secret-shared circuit evaluation, used heavily both by SMPC [11,39] and AMPC [9,10] protocols. In this approach, each value during the circuit-evaluation remains verifiably secret-shared among the parties in such a way that

the shares of the corrupt parties do not reveal any additional information to the adversary. Such protocols are conceptually much simpler and typically *do not* require any trusted setup. Moreover, they are computationally inexpensive, based on the properties of polynomials over fields and avoid costly machinery like ZK protocols for proving the correctness during circuit-evaluation. The *unconditionally-secure* network-agnostic protocols of [2,3] are based on the approach of the secret-shared circuit-evaluation. Even though the approach is well studied in the literature, surprisingly, to the best of our knowledge, no one has yet explored the feasibility of network-agnostic MPC protocols with *cryptographic security* based on the approach of secret-shared circuit evaluation. This motivates us to ask the following central question:

*Does there exist an efficient network-agnostic MPC protocol with cryptographic security and optimal threshold conditions, based on secret-shared circuit-evaluation, without deploying any computationally-expensive cryptographic machinery, such as ZK and any expensive setup such as threshold homomorphic encryption?*

**Our Results.** In this work, we make inroads to answer the above question by presenting a network-agnostic MPC with the condition $t_a + 2t_s < n$, based on the approach of secret-shared circuit-evaluation. Our protocol is in the *plain* PKI model and requires the setup of only a linearly-homomorphic commitment scheme. For simplicity, we use Pedersen's commitment scheme, whose security is based on the standard *discrete log* assumption in a cyclic group. To instantiate the scheme, the only setup needed is the *public* knowledge of a generator of a cyclic group, along with the *public* knowledge of a random element of the group. We stress that *unlike* the setup of threshold homomorphic encryption which has to generate both *public* as well as *private* components, the setup for Pedersen's scheme is relatively simpler, since it has to generate only *public* components. Our protocol also *avoids* any kind of ZK proofs (and the associated setup assumptions) for proving the correctness of the messages exchanged among the parties. A central pillar in our protocol is the *first* network-agnostic *Verifiable Secret Sharing* (VSS) protocol [19] with *cryptographic security*, which is of independent interest. VSS is in itself a very important cryptographic primitive and used for a variety of important secure distributed-computing tasks. For example, it can be used to instantiate a *common-coin* primitive [18,29] from scratch *without* any setup, which is a central tool for designing Byzantine agreement (BA) protocols with a constant expected time [29]. VSS also constitutes an important building block for designing *distributed key-generation* (DKG) protocols [35]. Previously, network-agnostic VSS is known either with *perfect-security* but with condition $t_a + 3t_s < n$ [3] or with *statistical-security* and condition $t_a + 2t_s < n$ [2], but where the parties need to perform an exponential amount of computation and communication.

We outline our MPC protocol and compare it with the relevant MPC protocols in Table 1. In the table, **Rounds** denotes the (expected) round complexity,

while $D_M$ and $c_M$ denote the multiplicative depth and the number of multiplication gates in the underlying circuit, representing the function to be securely computed. **Communication Complexity (CC)** denotes the number of bits communicated by the parties in the protocol for evaluating the multiplication gates in the underlying circuit and $\kappa$ denotes the computational security parameter. **Setup** denotes the setup assumptions, which include a plain public-key infrastructure (PKI), a common reference string (CRS), a threshold homomorphic cryptosystem, or a homomorphic commitment.

**Table 1.** Summary of network-agnostic MPC protocols with cryptographic security.

| Reference | Rounds | CC | Setup |
|---|---|---|---|
| [15] | $\mathcal{O}(D_M)$ | $\mathcal{O}(c_M \cdot n^3 \cdot \kappa)$ | CRS, PKI, Threshold Encryption |
| [27] | $\mathcal{O}(\kappa)$ | $\mathcal{O}(c_M \cdot n^3 \cdot \kappa)$ | CRS, PKI, Threshold Encryption |
| [5] | $\mathcal{O}(D_M)$ | $\mathcal{O}(c_M \cdot n^3 \cdot \mathsf{poly}(\kappa))$ | PKI, Threshold Encryption |
| **This work** | $\mathcal{O}(D_M)$ | $\mathcal{O}(c_M \cdot n^7 \cdot \kappa)$ | PKI, Homomorphic Commitment |

Although our protocol incurs higher communication complexity, it demonstrates the feasibility of achieving MPC *without* relying on complex setup assumptions, as well as computationally expensive cryptographic machinery. Moreover, during the design of our protocol, we get an independent and important cryptographic primitive, namely VSS. As explained in the next section, our work shows the feasibility of performing secure circuit-evaluation on secret-shared data with the condition $t_a + 2t_s < n$ and that too without deploying complex cryptographic machinery. Prior secret-shared based protocols were either with condition $t_a + 3t_s < n$ [3] or with the condition $t_a + 2t_s < n$ but requiring exponential computation and communication [2].

### 1.1 Technical Overview

We assume that the function to be securely computed is abstracted as an arithmetic circuit $\mathsf{cir}$ over a finite field, consisting of linear and non-linear (multiplication) gates. The idea is then to securely evaluate $\mathsf{cir}$ in a *secret-shared* manner, based on the paradigm of [11]. In this approach, each value during the evaluation of $\mathsf{cir}$ is (verifiably) $t$-shared according to Shamir's secret-sharing scheme [40], where $t$ denotes the maximum number of corrupt parties. Essentially, this guarantees that an adversary controlling up to $t$ parties gains no additional information throughout the circuit evaluation process, as the shares held by the corrupt parties do not reveal additional details about the underlying shared values. Since the parties will be unaware of the network type, the *degree-of-sharing* $t$ has to be set to *always* $t = t_s$.

The linearity of Shamir's secret-sharing guarantees that the linear gates in $\mathsf{cir}$ can be evaluated non-interactively over secret-shared gate inputs. For eval-

uating the multiplication gates over secret-shared inputs, we deploy the standard Beaver's paradigm [7]. According to this paradigm, multiplication gates are evaluated using randomly generated $t_s$-shared *multiplication-triples*. The secret-shared multiplication-triples are generated leveraging the framework proposed in [21]. This framework demonstrates how to utilize any polynomial-based VSS and a BA protocol to generate shared random multiplication-triples. Importantly, the framework operates in both synchronous and asynchronous networks, assuming that the participating parties possess knowledge of the *exact* network type. The work by [3] introduces techniques for adapting the framework to achieve *perfect* security with condition $t_a + 3t_s < n$ in a network-agnostic context. For our purpose, we adapt the framework with condition $t_a + 2t_s < n$ to achieve *cryptographic* security. This requires network-agnostic BA and VSS protocols with condition $t_a + 2t_s < n$. The BA protocol presented in [14] aligns well with our objectives. The main challenge however is finding a network-agnostic VSS protocol.

Informally, in a (polynomial-based) VSS protocol, there exists a designated *dealer* $\mathsf{D} \in \mathcal{P}$ with a $t$-degree polynomial as input, where $t$ represents the maximum number of *corrupt* parties, possibly including $\mathsf{D}$. The protocol allows $\mathsf{D}$ to distribute points on this polynomial to the parties in a "verifiable" manner, such that the view of the adversary remains independent of $\mathsf{D}$'s polynomial for an *honest* $\mathsf{D}$ (*privacy* property). In a *synchronous* VSS (SVSS) protocol, every party has the correct point on the polynomial after some known time-out, say $T$ (*correctness* property). The *verifiability* ensures that even if $\mathsf{D}$ is corrupt, it is bound to distribute points on some $t$-degree polynomial within time $T$ (*strong-commitment* property). It is well-known that cryptographically-secure SVSS is possible iff $t < \frac{n}{2}$ [39]. In an *asynchronous* VSS (AVSS) protocol, the *correctness* property guarantees that when $\mathsf{D}$ is *honest*, the honest parties will *eventually* receive points on $\mathsf{D}$'s polynomial. However, a corrupt $\mathsf{D}$ may choose not to invoke the protocol, and the parties *cannot* distinguish this situation from when $\mathsf{D}$'s messages are arbitrarily delayed. Therefore, the *strong-commitment* of AVSS ensures that if $\mathsf{D}$ is *corrupt* and if at least one honest party obtains a point on $\mathsf{D}$'s polynomial, then all honest parties will eventually obtain their respective points on this polynomial. It is well-known that cryptographically-secure AVSS is possible iff $t < \frac{n}{3}$ [17].

Existing SVSS protocols become completely insecure in an *asynchronous* network, even if a single anticipated message from an *honest* party encounters a delay. Conversely, existing AVSS protocols work only when at most $t < \frac{n}{3}$ parties are corrupt, and become insecure if the number of corruptions exceeds $\frac{n}{3}$ (which can happen in our context when the network behaves *synchronously* where $t < \frac{n}{2}$). We are currently unaware of any cryptographically-secure VSS protocol that provides the specific guarantees we seek, all while adhering to the minimally intensive computational requirements. We propose a network-agnostic cryptographically-secure VSS protocol that meets the aforementioned requirements, given that the condition $t_a + 2t_s < n$ is met. Our VSS protocol satisfies the *correctness* requirement of SVSS and AVSS in a synchronous and an asyn-

chronous network, respectively. However, it *only* satisfies the *strong-commitment* requirement of AVSS, even in a synchronous network. This is because a potentially *corrupt* $\mathsf{D}$ may choose *not* to invoke the protocol, and the parties will *not* be aware of the exact network type. We stress that this does not hinder us from deploying our VSS protocol in the framework of [21]. Given that our VSS protocol involves technical intricacies, we defer its explanation to Sect. 4 of our paper.

The construction of the aforementioned VSS protocol necessitates the implementation of a Byzantine Broadcast protocol. Informally, a broadcast protocol allows a designated party called a *Sender* to consistently distribute a message among a set of parties which guarantees security in a synchronous network where the adversary can corrupt up to $t < \frac{n}{2}$ parties, and also in an asynchronous network, where the adversary can corrupt up to $t < \frac{n}{3}$ parties. Previous research has presented secure broadcast protocols for both synchronous [28] and asynchronous [16] networks. Additionally, previous studies by [5,15,27] have also leveraged broadcast protocols in the network-agnostic setting. However, their proposed protocols offer comparatively weaker consistency assurances in asynchronous networks. For instance, the protocol introduced by [15] only ensures that even though every honest party outputs some value, only a subset of honest parties output the desired value, while the rest output some default value in an asynchronous network. Consequently, we introduce an additional broadcast protocol designed to ensure security in both synchronous and asynchronous networks, which can be safely integrated into our VSS protocol.

**Other Related Works.** As mentioned earlier, the domain of network-agnostic cryptographic protocols is relatively new. The works of [1,13] present network-agnostic cryptographically-secure atomic broadcast protocol. The work of [36] studies Byzantine fault tolerance and state machine replication protocols for multiple thresholds, including $t_s$ and $t_a$. The works of [30,31] study network-agnostic protocol for the task of approximate agreement. The works of [12,26] have studied the problem of network-agnostic secure message transmission (SMT) over *incomplete* graphs.

**Paper Organization.** The major contribution of the paper is the network-agnostic VSS and so we mostly focus on it; the design of the preprocessing phase protocol for generating the secret-shared multiplication-triples and the MPC protocol mostly follows from [3] by adapting the techniques to the setting of $t_a + 2t_s < n$. Due to space constraints, the detailed formal security proofs are not presented in this extended abstract and are deferred to the full version of the paper.

## 2 Preliminaries and Definitions

We consider a network of $n$ parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, where the distrust in the system is modelled by a *computationally bounded* Byzantine (malicious) adversary $\mathsf{Adv}$, who can corrupt a subset of parties and force them to behave in any

*arbitrary* fashion during the execution of a protocol. We assume a *static* adversary, who decides the set of corrupt parties at the beginning of the protocol execution.

We consider a communication model where the parties have access to *local* clocks and are *not aware* apriori about the network conditions when executing any protocol, where the underlying network could behave either in a *synchronous* fashion or in an *asynchronous* fashion. In a *synchronous* network, every sent message is delivered within some *known* time bound $\Delta$. The protocols in this model can be conveniently described as a sequence of communication *rounds*, where for every $r \in \mathbb{N}$ with $r \geq 1$, any message received in the time slot $[r\Delta, (r+1)\Delta]$ as per the local clock of the receiving party is regarded as a round-$r$ message. Moreover, in this model, it is assumed that the adversary $\mathsf{Adv}$ can control up to $t_s$ parties.

In an *asynchronous* network, the local clocks of the parties are *not* synchronised, and there is *no known* upper bound on message delays. To model the worst-case scenario, the adversary is allowed to schedule the delivery of messages *arbitrarily*, with the restriction that every message being sent is *eventually* delivered. The protocols in this model are described in an *event-based* fashion. That is, upon receiving a message, the receiving party adds the message to a pool of received messages and checks whether a list of conditions specified in the underlying protocol is satisfied to decide its next set of actions. In this model, it is assumed that $\mathsf{Adv}$ can corrupt at most $t_a$ parties.

We assume that $t_a < t_s$ and $t_a + 2t_s < n$ holds. This automatically implies that $t_s < \frac{n}{2}$ and $t_a < \frac{n}{3}$ holds, which are necessary for any cryptographically-secure SMPC and AMPC protocol, respectively. We assume that the function to be securely computed is abstracted as an arithmetic circuit $\mathsf{cir}$ over the prime field $\mathbb{F}_p$, where $p$ is a $\kappa$-bit long prime and where $\kappa$ is the security parameter. Moreover, we assume that $|\mathbb{F}_p| > n$ and each party $P_i$ is publicly associated with the evaluation point $i$ at which all the shares are computed for $P_i$. For simplicity and without loss of generality, we assume that each $P_i$ has a private input $x^{(i)} \in \mathbb{F}_p$, and the parties want to securely compute a function $f : \mathbb{F}_p^n \rightarrow \mathbb{F}_p$. Without loss of generality, $f$ is represented by an arithmetic circuit $\mathsf{cir}$ over $\mathbb{F}_p$, consisting of linear and non-linear (multiplication) gates, where $\mathsf{cir}$ has $c_M$ number of multiplication gates and has a multiplicative depth of $D_M$.

**Termination Guarantees of Our Sub-protocols.** As done in [3], for simplicity, we will not be specifying any termination criteria for our sub-protocols. And the parties will keep on participating in these sub-protocol instances, even after receiving their outputs. The termination criteria of our MPC protocol will ensure that once a party terminates the MPC protocol, it terminates all underlying sub-protocol instances.

### 2.1 Primitives and Definitions

We next present the definitions and primitives used in our protocols.

**Polynomials Over a Field.** A $d$-degree univariate polynomial over $\mathbb{F}_p$ is of the form $f(x) = a_0 + a_1 x + ... + a_d x^d$, where each $a_i \in \mathbb{F}_p$. A $(d, d)$-degree bivariate polynomial over $\mathbb{F}_p$ is of the form

$$F(x, y) = \sum_{i,j=0}^{i=d,j=d} r_{ij} x^i y^j,$$

where each $r_{ij} \in \mathbb{F}_p$. The polynomial is called *symmetric* if $r_{ji} = r_{ij}$ holds for all $i, j$. This automatically implies that $F(i, j) = F(j, i)$ holds for all $i, j \in \{1, \ldots, n\}$. Given $i \in \{1, \ldots, n\}$ and a $d$-degree polynomial $F_i(x)$, we say that $F_i(x)$ *lies* on a $(d, d)$-degree symmetric bivariate polynomial $F(x, y)$, if $F(x, i) = F_i(x)$ holds. The following properties for bivariate polynomials are standard.

**Lemma 1 ([4,24])(Pairwise Consistency Lemma).** *Let $f_{i_1}(x), \ldots, f_{i_q}(x)$ be $d$-degree univariate polynomials over $\mathbb{F}_p$, where $q \geq d + 1$ and $i_1, \ldots, i_q \in \{1, \ldots, n\}$, such that $f_i(j) = f_j(i)$ holds for all $i, j \in \{i_1, \ldots, i_q\}$. Then $f_{i_1}(x), \ldots, f_{i_q}(x)$ lie on a unique $(d, d)$-degree symmetric bivariate polynomial, say $F^\star(x, y)$.*

**Lemma 2 ([4,24]).** *Let $\mathcal{C} \subset \mathcal{P}$ and $q_1(\cdot) \neq q_2(\cdot)$ be $d$-degree polynomials where $d \geq |\mathcal{C}|$ such that $q_1(i) = q_2(i)$ holds for all $P_i \in \mathcal{C}$. Then the probability distributions $\left\{\{F(x, i)\}_{P_i \in \mathcal{C}}\right\}$ and $\left\{\{F'(x, i)\}_{P_i \in \mathcal{C}}\right\}$ are identical, where $F(x, y)$ and $F'(x, y)$ are random $(d, d)$-degree symmetric bivariate polynomials, such that $F(0, y) = q_1(\cdot)$ and $F'(0, y) = q_2(\cdot)$ holds.*

In our protocols, we deploy a homomorphic commitment scheme, which is instantiated with Pedersen's commitment scheme [38]. Informally, the scheme enables a party to *commit* a value such that later it can be *opened* uniquely.

**Pedersen Commitment Scheme [38].** The scheme consists of a two-phase protocol involving a *committer* and a *verifier*. The first phase is known as the "commit" phase, executed through a protocol denoted as $\mathsf{Commit}$, while the second phase is the "opening" phase, implemented via the protocol $\mathsf{Open}$. In the $\mathsf{Commit}$ protocol, the committer possesses a private input value $m \in \mathbb{F}_p$, which it commits to the verifier by publicly disclosing a commitment denoted as $\mathsf{Com}_m$. During the $\mathsf{Open}$ protocol, the committer reveals the actual value $m$ that was committed in $\mathsf{Com}_m$. Subsequently, the verifier verifies whether $m$ was indeed the value committed in $\mathsf{Com}_m$ during the $\mathsf{Commit}$ phase. The verifier's output is either 1 if the commitment is valid or 0 if it is not. Pedersen's commitment scheme offers the *hiding* property, which implies that if the committer is *honest*, the verifier gains no information about the value $m$. The verifier's view remains independent of the specific value $m$ committed in $\mathsf{Com}_m$, and this independence holds even when the verifier is *computationally unbounded*. Additionally, the scheme adheres to the *binding* property, which means that if the committer is *corrupt* and the verifier is *honest*, then except with a negligible probability, it is impossible for the committer to commit a value $m$ in $\mathsf{Com}_m$ during the $\mathsf{Commit}$ phase and subsequently reveal a different value $m^\star \neq m$ during the $\mathsf{Open}$ phase,

leading to the verifier outputting 1. The detailed formal description of the scheme is as follows.

As part of the setup, the parties are equipped with essential information: a generator $g$ of $\mathbb{F}_p$ and a randomly chosen element $h \in \mathbb{F}_p$. To commit a value $m \in \mathbb{F}_p$, the committer first selects a random value $r$ from $\mathbb{F}_p$ and computes the commitment $\mathsf{Com}_m = \mathsf{Commit}(m, r) \stackrel{def}{=} g^m h^r \bmod p$. During the $\mathsf{Open}$ protocol, the committer discloses $m^\star$ and $r^\star$ (which are supposed to be $m$ and $r$ respectively for an *honest* committer), after which the verifier decides to output either 1 or 0, contingent upon the equality check $\mathsf{Com}_m \stackrel{?}{=} g^{m^\star} h^{r^\star} \bmod p$. Both the $\mathsf{Commit}$ and $\mathsf{Open}$ protocols incur a communication complexity of $\mathcal{O}(\kappa)$ bits. It is a well-established fact that if the committer is honest, then the view of a corrupt verifier remains identically distributed for any potential combination of $(m, r)$, even when the verifier possesses unbounded computational capabilities. Conversely, assuming that solving the Discrete logarithm problem in $\mathbb{F}_p$ is computationally challenging, it becomes infeasible for a corrupt committer to reveal $(m^\star, r^\star) \neq (m, r)$ during the $\mathsf{Open}$ phase in such a way that an honest verifier would output 1. Throughout the remainder of this paper, we say that $(m, r)$ is *consistent* with $\mathsf{Com}_m$ if and only if $\mathsf{Com}_m = g^m h^r \bmod p$ holds.

The Pedersen commitment scheme exhibits homomorphic properties. Given commitments $\mathsf{Com}_{m_1} = \mathsf{Commit}(m_1, r_1)$ and $\mathsf{Com}_{m_2} = \mathsf{Commit}(m_2, r_2)$, as well as publicly-known constants $c_1, c_2 \in \mathbb{F}_p$, it is feasible to compute a commitment $\mathsf{Com}_{c_1 m_1 + c_2 m_2}$ for the sum $c_1 m_1 + c_2 m_2$ under the randomness $c_1 r_1 + c_2 r_2$ through local computations performed on $\mathsf{Com}_{m_1}$ and $\mathsf{Com}_{m_2}$. Specifically, we can express this as $(\mathsf{Com}_{m_1})^{c_1} \cdot (\mathsf{Com}_{m_2})^{c_2} = g^{c_1 m_1 + c_2 m_2} \cdot h^{c_1 r_1 + c_2 r_2} = \mathsf{Com}_{c_1 m_1 + c_2 m_2}$. In general, let $q : \mathbb{F}_p^l \to \mathbb{F}_p^m$ be an arbitrary linear function where $q(x_1, \ldots, x_l) = (y_1, \ldots, y_m)$. Then given commitments $\mathsf{Com}_{x_1}, \ldots, \mathsf{Com}_{x_l}$ for $x_1, \ldots, x_l$ respectively, it is possible to locally compute commitments $\mathsf{Com}_{y_1}, \ldots, \mathsf{Com}_{y_m}$ for $y_1, \ldots, y_m$, respectively, using only $\mathsf{Com}_{x_1}, \ldots, \mathsf{Com}_{x_l}$. Throughout the paper, we use the term parties locally compute $\mathsf{Com}_{y_1}, \ldots, \mathsf{Com}_{y_m} = q(\mathsf{Com}_{x_1}, \ldots, \mathsf{Com}_{x_l})$ to refer to this process of leveraging the above linearity properties for such computations.

We will now revisit the definition of $t$-sharing with publicly committed shares as outlined in [6].

**Definition 1 ($t$-Sharing with Publicly Committed Shares [6]).** *A value $s \in \mathbb{F}_p$ is said to be $t$-shared with publicly committed shares, if there exists a $t$-degree polynomial $f(x)$ over $\mathbb{F}_p$ with $f(0) = s$, such that each (honest) $P_i \in \mathcal{P}$ holds $(s_i, r_i)$ and a vector of commitments $(\mathsf{Com}_{s_1}, \ldots, \mathsf{Com}_{s_n})$, where $s_i = f(i)$ and $\mathsf{Com}_{s_j} = \mathsf{Commit}(s_j, r_j)$, for $j = 1, \ldots, n$.*

In our notation, we use $[s]$ to represent a vector of shares and the publicly known commitment of these shares. Additionally, we use $[s]_i$ to represent $P_i$'s share, $(s_i, r_i)$ and the public commitments. We will use the term "a value $s$ is $[\cdot]$-shared among the parties", to describe a scenario where $s$ is $t_s$-shared (unless specified otherwise) with publicly committed shares.

Note that $[\cdot]$-sharing satisfies the linearity property, which arises from the linearity property of Pedersen's commitment scheme. In other words, given shared values $[a]$ and $[b]$ along with public constants $c_1$ and $c_2$, each party $P_i$ can locally compute its information corresponding to $[c_1a + c_2b]$. In a more general context, suppose we have a linear function $q : \mathbb{F}_p^l \rightarrow \mathbb{F}_p^m$ and let $q(x_1, \ldots, x_l) = (y_1, \ldots, y_m)$. Then given the secret-shared values $[x_1], \ldots, [x_l]$, each party can locally compute its information corresponding to $[y_1], \ldots, [y_m]$. Throughout the rest of the paper, we use the phrase parties locally compute $([y_1], \ldots, [y_m]) = q([x_1], \ldots, [x_l])$ to signify this property.

**Digital Signature Scheme.** We assume a Public Key Infrastructure (PKI) setup, where each party $P_i$ within the set $\mathcal{P}$ possesses a pair of keys for a digital signature scheme, specifically a signing key $\mathsf{sk}_i$ and a verification key $\mathsf{vk}_i$. Importantly, the verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$ for all parties are publicly accessible, while the signing key $\mathsf{sk}_i$ is kept private and known only to $P_i$ (malicious parties may choose their keys arbitrarily). We further assume that the digital signature scheme adheres to the standard security notion of *unforgeability*. This means that, except with a negligible probability in $\kappa$, the adversary cannot produce a valid signature of an honest party $P_i$ on any message $m$ that was never signed by $P_i$. Each signature generated using this scheme has a fixed size of $\kappa$ bits, where $\kappa$ represents the security parameter. In the paper, we use the notation $\langle m \rangle_{P_i}$ to indicate party $P_i$'s signature on a message $m$. Additionally, we refer to a signature $\langle m \rangle_{P_i}$ on message $m$ as *valid* if, along with the message $m$, it successfully passes the verification process using the verification key $\mathsf{vk}_i$ corresponding to $P_i$.

**Definition 2 (Byzantine Agreement (BA) [14]).** *Let $\Pi$ be a protocol for $\mathcal{P}$, where every party $P_i$ has an input $b_i \in \{0,1\}$ and a possible output from $\{0,1\} \cup \{\perp\}$. Moreover, let* Adv *be a computationally-bounded adversary, which can corrupt up to $t$ parties in $\mathcal{P}$ during the execution of $\Pi$.*

- *$t$-**liveness:** $\Pi$ has $t$-liveness if all honest parties obtain an output.*
- *$t$-**validity:** $\Pi$ has $t$-validity if the following holds: If all honest parties have input $b$, then every honest party with an output, outputs $b$.*
- *$t$-**weak validity:** $\Pi$ has $t$-weak validity if the following holds: If all honest parties have input $b$, then every honest party with an output, outputs $b$ or $\perp$.*
- *$t$-**consistency:** $\Pi$ has $t$-consistency if all honest parties with an output, output the same value (which can be $\perp$).*
- *$t$-**weak consistency:** $\Pi$ has $t$-weak consistency if all honest parties with an output, output either a common $v \in \{0,1\}$ or $\perp$.*

A protocol $\Pi$ is said to be a *$t$-secure Byzantine Agreement* (BA) protocol if it guarantees *$t$-liveness*, *$t$-validity* and *$t$-consistency*.

**Definition 3 (Byzantine Broadcast [14]).** *Let $\Pi$ be a protocol for $\mathcal{P}$, where a designated sender $\mathsf{S} \in \mathcal{P}$ has input $m \in \{0,1\}$, and parties obtain a possible output from $\{0,1\} \cup \{\perp\}$. Moreover, let* Adv *be a computationally-bounded adversary, which can corrupt up to $t$ parties in $\mathcal{P}$ during the execution of $\Pi$.*

- $t$-**liveness**: *$\Pi$ has $t$-liveness, if all honest parties obtain an output.*
- $t$-**validity**: *$\Pi$ has $t$-validity if the following holds: if $\mathsf{S}$ is honest, then every honest party with an output, outputs $m$.*
- $t$-**weak validity**: *has $t$-weak validity if the following holds: if $\mathsf{S}$ is honest, then every honest party outputs either $m$ or $\perp$.*
- $t$-**consistency**: *$\Pi$ has $t$-consistency if the following holds: if $\mathsf{S}$ is corrupt, then every honest party with an output, has a common output.*
- $t$-**weak consistency**: *$\Pi$ has $t$-weak consistency if the following holds: if $\mathsf{S}$ is corrupt, then every honest party with an output, outputs either a common $m^\star \in \{0,1\}$ or $\perp$.*

A protocol $\Pi$ is said to be a *$t$-secure Broadcast* protocol if it guarantees *$t$-liveness*, *$t$-validity* and *$t$-consistency*.

### 2.2 Existing Building Blocks

*Network-Agnostic Byzantine Agreement.* The work of [14] presents a network-agnostic BA protocol $\Pi_{\mathsf{BA}}$ with $t_a + 2t_s < n$ that achieves $t_s$-*security* when run in a synchronous network and $t_a$-*security* when run in an asynchronous network. The protocol is obtained cleverly by combining a $t_s$-secure *synchronous* BA (SBA) protocol which also provides certain guarantees in an asynchronous network and a $t_a$-secure *asynchronous* BA (ABA) protocol which also provides certain guarantees in a synchronous network (against $t_s$ corruptions). The detailed description is omitted due to brevity, but interested readers can refer to [14] for further detail and clarification.

In our VSS protocol, we use several standard procedures to verify certain properties of univariate polynomials and points lying on a bivariate polynomial, where the bivariate polynomial is publicly committed. All these procedures are based on the homomorphic properties of Pedersen's commitment scheme. We next describe these procedures.

**Verifying Committed Bivariate Polynomial.** This procedure takes input the commitments of the coefficients of $n$ univariate polynomials, supposedly lying on a $(d,d)$-degree symmetric bivariate polynomial, where $d < n$. The procedure outputs 1 iff the committed polynomials lie on a $(d,d)$-degree symmetric bivariate polynomial. In more detail, the procedure $\mathsf{VerifyPoly}(\mathbf{C}, d)$ takes input a matrix $\mathbf{C}$ of commitments of size $n \times (d+1)$, where for $i = 1, \ldots, n$, the $i^{th}$ row consists of the commitments $\{\mathsf{Com}_{ij}\}_{j=0,\ldots,d}$. The output of the procedure is 1 iff there exists $(d,d)$-degree *symmetric* bivariate polynomials $H(x,y)$ and $R(x,y)$ over $\mathbb{F}_p$, such that the following condition is satisfied for $i = 1, \ldots, n$:

- Let $h_i(x) = H(x,i) = h_{i0} + h_{i1} \cdot x + \ldots + h_{id} \cdot x^d$ and $r_i(x) = R(x,i) = r_{i0} + r_{i1} \cdot x + \ldots + r_{id} \cdot x^d$. Then $\mathsf{Com}_{ij} = \mathsf{Commit}(h_{ij}, r_{ij})$ should hold, for $j = 0, \ldots, d$.

The procedure is very simple and *homomorphically* checks if the pairwise consistency lemma (Lemma 1) is satisfied for each $i, j \in \{1, \ldots, n\}$. In more

detail, consider the commitments $\mathsf{Com}_{i0}, \ldots, \mathsf{Com}_{id}$ along the $i^{th}$ row of $\mathbf{C}$. Let $h_i^\star(x) = h_{i0}^\star + h_{i1}^\star \cdot x + \ldots + h_{id}^\star \cdot x^d$ and $r_i^\star(x) = r_{i0}^\star + r_{i1}^\star \cdot x + \ldots + r_{id}^\star \cdot x^d$ be $d$-degree polynomials, such that $\mathsf{Com}_{ij} = \mathsf{Commit}(h_{ij}^\star, r_{ij}^\star)$ holds for $j = 0, \ldots, d$. We wish to check whether $h_i^\star(j) = h_j^\star(i)$ and $r_i^\star(j) = r_j^\star(i)$ holds for each $i, j \in \{1, \ldots, n\}$. For this, we check the above relation over the commitments of $h_i^\star(j)$ and $h_j^\star(i)$, under the randomness $r_i^\star(j)$ and $r_j^\star(i)$ respectively. This is possible since the commitment of $h_i^\star(j)$ and $h_j^\star(i)$ can be homomorphically computed, as these points can be computed as a publicly-known *linear* function of the coefficients of $h_i^\star(x)$ and $h_j^\star(x)$ respectively. In more detail, $h_i^\star(j) = h_{i0}^\star + h_{i1}^\star \cdot j + \ldots + h_{id}^\star \cdot j^d$. and $r_i^\star(j) = r_{i0}^\star + r_{i1}^\star \cdot j + \ldots + r_{id}^\star \cdot j^d$. Consequently, $\mathsf{Commit}(h_i^\star(j), r_i^\star(j)) = \mathsf{Com}_{i0} \cdot (\mathsf{Com}_{i1})^j \cdot (\mathsf{Com}_{i2})^{j^2} \cdot \ldots \cdot (\mathsf{Com}_{id})^{j^d}$ holds and similarly $\mathsf{Commit}(h_j^\star(i), r_j^\star(i)) = \mathsf{Com}_{j0} \cdot (\mathsf{Com}_{j1})^i \cdot (\mathsf{Com}_{j2})^{i^2} \cdot \ldots \cdot (\mathsf{Com}_{jd})^{i^d}$ holds. Hence instead of checking $h_i^\star(j) = h_j^\star(i)$ and $r_i^\star(j) = r_j^\star(i)$, the procedure actually checks if the following holds for each $i, j \in \{1, \ldots, n\}$:

$$\mathsf{Com}_{i0} \cdot (\mathsf{Com}_{i1})^j \cdot \ldots \cdot (\mathsf{Com}_{id})^{j^d} = \mathsf{Com}_{j0} \cdot (\mathsf{Com}_{j1})^i \cdot \ldots \cdot (\mathsf{Com}_{jd})^{i^d}.$$

**Verifying Point on a Committed Polynomial.** The next procedure $\mathsf{VerifyPoint}(\{\mathsf{Com}_0, \ldots, \mathsf{Com}_d\}, (\mathbf{h}, \mathbf{r}), j)$ takes input a set of $d+1$ commitments, an index $j \in \{1, \ldots, n\}$ and a pair of values $(\mathbf{h}, \mathbf{r})$, where the commitments are already *known* to be the commitments of the coefficients of a $d$-degree polynomial. That is, there exist $d$-degree polynomials, say $h(x) = h_0 + h_1 \cdot x + \ldots + h_d \cdot x^d$ and $r(x) = r_0 + r_1 \cdot x + \ldots + r_d \cdot x^d$, where it is *already known* that $\mathsf{Com}_i = \mathsf{Commit}(h_i, r_i)$ holds, for $i = 0, \ldots, d$. The procedure outputs 1 iff $\mathbf{h}$ and $\mathbf{r}$ constitutes the $j^{th}$ point on $h(x)$ and $r(x)$ respectively; i.e. iff $h(j) = \mathbf{h}$ and $r(j) = \mathbf{r}$ holds. Similar to the previous procedure, the verification here happens homomorphically over the commitments. That is, the procedure checks if the following relation holds:

$$\mathsf{Commit}(\mathbf{h}, \mathbf{r}) = \mathsf{Com}_0 \cdot (\mathsf{Com}_1)^j \cdot (\mathsf{Com}_2)^{j^2} \cdot \ldots \cdot (\mathsf{Com}_d)^{j^d}.$$

**Robust Reconstruction of a $[\cdot]$-Shared Value.** Let $s$ be a value which is $[\cdot]$-shared among the parties, where the degree of sharing is $t_s$, with each (honest) $P_i$ having its share $(s_i, r_i)$ and all the parties having the commitments $\mathsf{Com}_{s_1}, \ldots, \mathsf{Com}_{s_n}$, where $\mathsf{Com}_{s_i} = \mathsf{Commit}(s_i, r_i)$. Protocol $\Pi_{\mathsf{RecPub}}([s])$ then allows the parties in $\mathcal{P}$ to publicly reconstruct $s$ irrespective of the network type, provided $t_s < \frac{n}{2}$. The protocol is standard and straightforward. Every $P_i$ provides its share $(s_i, r_i)$ to every party, which is verified with respect to $\mathsf{Com}_{s_i}$. Once $t_s + 1$ correct shares are identified, they are used to interpolate the underlying $t_s$-degree sharing polynomial and its constant term is taken as the output. The protocol requires $\Delta$ time in a synchronous network and irrespective of the network type, incurs a communication of $\mathcal{O}(n^2 \cdot \kappa)$ bits. Since the protocol is standard, we avoid presenting its formal description.

## 3 Network-Agnostic Byzantine Broadcast

We wish to design a cryptographically-secure network-agnostic broadcast protocol which is $t_s$-secure in a synchronous network and $t_a$-secure in an asynchronous network. For doing so, we assume a PKI setup. Our protocol follows the design of the *perfectly-secure* network-agnostic broadcast protocol from [3] which makes use of *two* primitives, which we discuss first.

### 3.1 Asynchronous Broadcast with Weaker Synchronous Guarantees

The first primitive is a protocol $\Pi_{\mathsf{ABC}}$, which is a $t_a$-secure broadcast in an asynchronous network. The primitive also achieves the properties of broadcast in a *synchronous* network against $t_s$ corruptions for an *honest* sender but fails if the sender is *corrupt*. Namely, in the latter case, there is *no* guarantee that the honest parties obtain any output. Moreover, even if the honest parties obtain an output, they *may not* do so at the same (local) time.

The work of [3] uses the famous *perfectly-secure* asynchronous broadcast protocol (or Acast) of [16] as an instantiation of $\Pi_{\mathsf{ABC}}$. The protocol can tolerate up to $t < \frac{n}{3}$ corruptions, irrespective of the network type and hence fits the bill in [3]; this is because for *perfect* security, $t_s < \frac{n}{3}$ holds. However, in our context, $t_s < \frac{n}{2}$ and so we *cannot* use the Acast protocol of [16] as instantiation of $\Pi_{\mathsf{ABC}}$. However, we notice that the reliable broadcast protocol of [36] realises the exact requirements we demand from an instantiation of $\Pi_{\mathsf{ABC}}$. The protocol achieves *cryptographic* security, assuming a PKI setup. For completeness, the protocol is given in Fig. 1.

**Protocol** $\Pi_{\mathsf{ABC}}$

- **Sending the Input** — Sender $\mathsf{S}$, on having input $m$, sends $(\mathsf{propose}, m, \langle m \rangle_S)$ to all the parties.
- **Forwarding the Input** — Each $P_i \in \mathcal{P}$ does the following:
  - Upon receiving the first $(\mathsf{propose}, m, \langle m \rangle_S)$ message with a valid signature from $\mathsf{S}$, send $(\mathsf{propose}, m, \langle m \rangle_S)$ to all $P_j \in \mathcal{P}$, and wait till local time becomes an integer multiple of $\Delta$.
- **Sending Vote** — Each $P_i \in \mathcal{P}$ does the following:
  - If $(\mathsf{propose}, m', \langle m' \rangle_S)$ for $m' \neq m$ is not received, then send $(\mathsf{vote}, m, \langle m \rangle_{P_i})$ to all the parties.
- **Sending Quorum and Output computation** — Each $P_i \in \mathcal{P}$ does the following:
  - Upon receiving a quorum of $n - t_s$ messages $(\mathsf{vote}, m, \langle m \rangle_\star)$ with valid signatures, denote these by $V(m)$, send $V(m)$ to all the parties and output the value $m$.

**Fig. 1.** Asynchronous broadcast with weaker synchronous guarantees

The properties of the protocol $\Pi_{\mathsf{ABC}}$ are stated in Lemma 3.

**Lemma 3.** *Let $t_a < t_s$ and $t_a + 2t_s < n$ and let $\mathsf{S}$ have a message $m$ as input for $\Pi_{\mathsf{ABC}}$. Then, the protocol $\Pi_{\mathsf{ABC}}$ satisfies the following properties.*

- *In a Synchronous Network:*
  - *$t_s$-liveness: If $\mathsf{S}$ is honest, then all honest parties obtain an output within time $3\Delta$.*
  - *$t_s$-validity: If $\mathsf{S}$ is honest, then every honest party outputs $m$.*
  - *$t_s$-consistency: If $\mathsf{S}$ is corrupt and some honest party outputs $m^\star$ at time $T$, then every honest party outputs $m^\star$ at time $T + \Delta$.*
- *In an Asynchronous Network:*
  - *$t_a$-liveness: If $\mathsf{S}$ is honest, then all honest parties eventually obtain an output.*
  - *$t_a$-validity: If $\mathsf{S}$ is honest, then every honest party with an output, outputs $m$.*
  - *$t_a$-consistency: If $\mathsf{S}$ is corrupt and some honest party outputs $m^\star$, then every honest party eventually outputs $m^\star$.*
- *Irrespective of the network type, the protocol incurs a communication of $\mathcal{O}(n^3 \cdot |m|)$ bits from the honest parties, where $|m|$ denotes the size of $m$ in bits.*

In our description, we will say that $\mathsf{S}$ *acasts* $m$ to mean that $\mathsf{S}$ invokes an instance of $\Pi_{\mathsf{ABC}}$ with input $m$ and the parties participate in this instance. Similarly, we will say that the *parties receive $m$ through the acast of* $\mathsf{S}$ to denote that the output of the parties in the $\Pi_{\mathsf{ABC}}$ instance is $m$.

### 3.2 Synchronous Byzantine Agreement

The second component used in [3] is a $t_s$-secure *synchronous* BA (SBA) protocol $\Pi_{\mathsf{SBA}}$, that additionally guarantees *liveness* in an *asynchronous* network against $t_a$ corruptions. The instantiation of $\Pi_{\mathsf{SBA}}$ is with *perfect* security where $t_s < \frac{n}{3}$ and will *not* work for our setting where $t_s < \frac{n}{2}$. Instead, our instantiation of $\Pi_{\mathsf{SBA}}$ is the Dolev-Strong BA protocol [28] based on the PKI setup, which requires $t_s + 1$ rounds (in a *synchronous* network). To achieve liveness in the *asynchronous* network, it suffices to have the parties execute the protocol for local time $T_{\mathsf{SBA}} = (t_s + 1) \cdot \Delta$ time and check if an output is computed at time $T_{\mathsf{SBA}}$. If no output is computed, then the parties output $\perp$, else they take the output as determined by the protocol. The protocol incurs a communication of $\mathcal{O}(n^3 \cdot |m|)$ bits from the honest parties, where $|m|$ is the size of $m$ in bits. The detailed protocol and proofs are omitted due to brevity.

### 3.3 $\Pi_{\mathsf{ABC}} + \Pi_{\mathsf{SBA}} \rightarrow$ Network-Agnostic BC

Once we have instantiations of $\Pi_{\mathsf{ABC}}$ and $\Pi_{\mathsf{SBA}}$, we obtain a network-agnostic broadcast protocol following [3] as follows: first, $\mathsf{S}$ invokes an instance of $\Pi_{\mathsf{ABC}}$ to broadcast its input. If the network is *synchronous* then all honest parties should have the sender's input at the time $3\Delta$. To check the same, the parties run an instance of $\Pi_{\mathsf{SBA}}$ on the outputs obtained from the $\Pi_{\mathsf{ABC}}$ instance at

time $3\Delta$. Finally, at time $3\Delta + T_{\mathsf{SBA}}$, the parties check the output from the $\Pi_{\mathsf{SBA}}$ instance. The parties then decide their output at time $3\Delta + T_{\mathsf{SBA}}$, based on the output they obtained from the instance of $\Pi_{\mathsf{ABC}}$ and $\Pi_{\mathsf{SBA}}$. This will ensure that the resultant protocol achieves $t_s$-security in a *synchronous* network. However, if the network is *asynchronous*, then it might be possible that at the local time $3\Delta + T_{\mathsf{SBA}}$, different honest parties have different outputs. Namely some honest parties may output $\perp$, while others may have an output $m^\star$, *different* from $\perp$. Consequently, as done in [3], we provide a provision for the former set of parties to continue running the protocol, so that they also eventually obtain the output $m^\star$. Following the terminology of [3], we denote the two different methods of computing the outputs as *regular* and *fallback* mode. The protocol is formally presented in Fig. 2.

**Protocol** $\Pi_{\mathsf{BC}}$

Regular Mode

- **Acasting the Input** — Sender $\mathsf{S}$, on having input $m$, acasts it.
- **Agreeing on the Acast Output** — Each party $P_i \in \mathcal{P}$ does the following at time $3\Delta$:
  - Set the input to an instance of $\Pi_{\mathsf{SBA}}$ as follows:
    - If an $m^\star$ is received from the acast of $\mathsf{S}$ then set it as the input;
    - Otherwise, set $\perp$ as the input.
  - Participate in the instance of $\Pi_{\mathsf{SBA}}$ with the set input.
- **Local Output Computation —** At time $3\Delta + T_{\mathsf{SBA}}$ each $P_i \in \mathcal{P}$ computes the output through *regular* mode as follows:
  - Output $m^\star \neq \perp$, if $m^\star$ is received from the acast of $\mathsf{S}$ *and* computed as the output during the instance of $\Pi_{\mathsf{SBA}}$.
  - Else, output $\perp$.

Fallback Mode

Every $P_i \in \mathcal{P}$ who computed the output $\perp$ at time $3\Delta + T_{\mathsf{SBA}}$ does the following: if $m^\star$ received from the acast of $\mathsf{S}$ then change the output from $\perp$ to $m^\star$.

**Fig. 2.** Network-agnostic broadcast protocol

The properties of the protocol $\Pi_{\mathsf{BC}}$ stated in Lemma 4 can be proved in the same way as in [3].

**Lemma 4.** *Let $t_a < t_s$ and $t_a + 2t_s < n$ and let $\mathsf{S}$ have an input $m$ for $\Pi_{\mathsf{BC}}$. Then, the protocol $\Pi_{\mathsf{BC}}$ satisfies the following properties, where $T_{\mathsf{BC}} = 3\Delta + T_{\mathsf{SBA}}$.*

- *In a Synchronous Network:*
  - *$t_s$-liveness: If $\mathsf{S}$ is honest, then all honest parties obtain an output within time $T_{\mathsf{BC}}$.*
  - *$t_s$-validity: If $\mathsf{S}$ is honest, then every honest party outputs $m$.*
  - *$t_s$-consistency: If $\mathsf{S}$ is corrupt, then*
    - *If some honest party outputs $m^\star$ at time $T_{\mathsf{BC}}$, then every honest party outputs $m^\star$ at time $T_{\mathsf{BC}}$.*

  - *If some honest party outputs $m^\star$ at time $T > T_{\mathsf{BC}}$, then every honest party outputs $m^\star$ at time $T + \Delta$.*
- *In an Asynchronous Network:*
  - *$t_a$-liveness: If $\mathsf{S}$ is honest, then all honest parties eventually obtain an output.*
  - *$t_a$-validity: If $\mathsf{S}$ is honest, then every honest party with an output, outputs $m$.*
  - *$t_a$-consistency: If $\mathsf{S}$ is corrupt and some honest party outputs $m^\star$, then every honest party eventually outputs $m^\star$.*
- *Irrespective of the network type, the protocol incurs a communication of $\mathcal{O}(n^3 \cdot |m|)$ bits from the honest parties.*

As in [3], we use the following terminologies for $\Pi_{\mathsf{BC}}$ in the rest of the paper.

**Terminologies for $\Pi_{\mathsf{BC}}$.** When we say that "*$P_i$ broadcasts $m$*", we mean that $P_i$ invokes an instance of $\Pi_{\mathsf{BC}}$ as $\mathsf{S}$ with input $m$ and the parties participate in this instance. Similarly, when we say that "*$P_j$ receives $m$ from the broadcast of $P_i$ through regular mode*", we mean that $P_j$ has the output $m$ at time $T_{\mathsf{BC}}$, during the instance of $\Pi_{\mathsf{BC}}$. Finally, when we say that "*$P_j$ receives $m$ from the broadcast of $P_i$ through fallback mode*", we mean that $P_j$ has the output $m$ after time $T_{\mathsf{BC}}$ during the instance of $\Pi_{\mathsf{BC}}$.

## 4 Network-Agnostic VSS

In this section, we present our network-agnostic VSS protocol $\Pi_{\mathsf{VSS}}$, which allows a designated dealer $\mathsf{D}$ to *verifiably* $[\cdot]$-share its input $s$, where the degree of sharing will be $t_s$, irrespective of the network type. For an *honest* $\mathsf{D}$, the value $s$ will be $[\cdot]$-shared *eventually* in an *asynchronous* network, while $s$ will be $[\cdot]$-shared after a *fixed* known time, if the network behaves *synchronously*. The *verifiability* here ensures that if $\mathsf{D}$ is *corrupt*, then either no honest party obtains any output (if $\mathsf{D}$ does not invoke the protocol), or there exists some value which gets $[\cdot]$-shared among the parties. Note that in the latter case, we *cannot* bound the time within which honest parties will have their shares, even if the network is *synchronous*. This is because a *corrupt* $\mathsf{D}$ may delay sending the designated messages arbitrarily, and the parties will *not* know the exact network type.

The idea behind $\Pi_{\mathsf{VSS}}$ is as follows: $\mathsf{D}$ embeds $s$ in a random $t_s$-degree polynomial $q(\cdot)$ in its constant term, where the polynomial $q(\cdot)$ is further embedded in a random $(t_s, t_s)$-degree symmetric bivariate polynomial, say $F(x, y)$, at $x = 0$. The goal is then to verifiably distribute the point $q(i)$ to each party $P_i$ and make public the commitments of these points. To achieve this, $\mathsf{D}$ further picks a random $(t_s, t_s)$-degree symmetric bivariate polynomial, say $R(x, y)$, and publicly commits the coefficients of the polynomials $f_i(x)$ using the coefficients of the polynomial $r_i(x)$ as randomness, where $f_i(x) = F(x, i)$ and $r_i(x) = R(x, i)$. The matrix of coefficients $\mathbf{C}$ which is of size $n \times (t_s + 1)$ is made public through an instance of our network-agnostic broadcast protocol $\Pi_{\mathsf{BC}}$. Additionally, each party $P_i$ is also provided the pair of points $\{f_i(j), r_i(j)\}_{j=1,\ldots,n}$. Every party

$P_i$ upon receiving the points $\{f_i(j), r_i(j)\}_{j=1,\ldots,n}$ and the matrix $\mathbf{C}$ can check for their "consistency". That is, $P_i$ can check if $\mathbf{C}$ constitutes commitments of the coefficients of some $(t_s, t_s)$-degree symmetric bivariate polynomial (using the procedure VerifyPoly) and if the points $\{f_i(j), r_i(j)\}_{j=1,\ldots,n}$ lie on the $i^{th}$ univariate polynomial committed in $\mathbf{C}$ (using the procedure VerifyPoint). If both the tests are positive, then $P_i$ notifies this publicly, through an instance of the broadcast protocol $\Pi_{\mathsf{BC}}$. The dealer D then looks for a candidate "core" set of parties $\mathcal{CS}$ of size at least $n - t_s$ who responded *positively* and upon finding a $\mathcal{CS}$, D makes it public (again through an instance of $\Pi_{\mathsf{BC}}$).

For simplicity, let us assume that the network behaves *synchronously* and D is *honest*. Then all *honest* parties (which are at least $n - t_s$ in number), should respond positively by time $2T_{\mathsf{BC}}$. This is because each instance of $\Pi_{\mathsf{BC}}$ takes $T_{\mathsf{BC}}$ time to generate an output. Consequently, D should find a candidate $\mathcal{CS}$ and hence all honest parties should have this $\mathcal{CS}$ at time $3T_{\mathsf{BC}}$. Based on this observation, at (local) time $3T_{\mathsf{BC}}$, the parties check if D made public a candidate core set of size at least $n - t_s$ (who have responded positively). Since different parties may have different opinions about the existence of a candidate core set, the parties execute an instance of the network-agnostic BA protocol $\Pi_{\mathsf{BA}}$ to have a common opinion. Based on the output of the $\Pi_{\mathsf{BA}}$ instance, the parties can conclude about the type of the network and behaviour of D.

Let us *first* consider the case when the output of the $\Pi_{\mathsf{BA}}$ instance is *positive*, implying that at least one honest party has seen a candidate $\mathcal{CS}$ at (local) time $3T_{\mathsf{BC}}$. Let $\mathcal{H}$ be the set of *honest* parties and let $\mathcal{H}_{\mathcal{CS}}$ be the set of honest parties in $\mathcal{CS}$. Note that $\mathcal{H}_{\mathcal{CS}} \neq \emptyset$, *irrespective* of the network type, since $|\mathcal{CS}| \geq n - t_s \geq t_s + t_a + 1$. Since the parties in $\mathcal{H}_{\mathcal{CS}}$ have verified $\mathbf{C}$ using the procedure VerifyPoly, it implies that there exists a $(t_s, t_s)$-degree *symmetric* bivariate polynomial, say $F^\star(x, y)$ and a $(t_s, t_s)$-degree bivariate polynomial, say $R^\star(x, y)$, such that coefficients of the polynomials $\{F^\star(x, i)\}_{i=1,\ldots,n}$ are committed by D in $\mathbf{C}$, using the coefficients of the polynomials $\{R^\star(x, i)\}_{i=1,\ldots,n}$ as randomness. We call $F^\star(x, y)$ as D's *committed* bivariate polynomial and note that $F^\star(x, y) = F(x, y)$ holds, if D is *honest*. Let $q^\star(\cdot) \stackrel{def}{=} F^\star(0, y)$ and let $s^\star \stackrel{def}{=} F^\star(0, 0)$. Again note that if D is *honest*, then $s^\star = s$, since $q^\star(\cdot) = q(\cdot)$ holds. Let $F_i^\star(x) = F^\star(x, i)$ and $R_i^\star(x) = R^\star(x, i)$. The next goal will be to ensure that $s^\star$ gets $[\cdot]$-shared, for which it is sufficient to have each $P_i \in \mathcal{H}$ have the pair of points $(F_i^\star(0), R_i^\star(0))$, since the commitment of these points are *already* available through $\mathbf{C}$. We also know that each party in $\mathcal{H}_{\mathcal{CS}}$ *already* has received the designated pair of points $(F_i^\star(0), R_i^\star(0))$ from D, since they responded positively after verifying the pairs of points received from D using the procedure VerifyPoint. So what is left to ensure that the potentially honest parties $P_i$ *outside* $\mathcal{CS}$ get their designated pair of points $(F_i^\star(0), R_i^\star(0))$. We stress that there *might* be such potential honest parties outside $\mathcal{CS}$. While this can always happen in an asynchronous network (even if D is honest) where the designated messages for a subset of honest parties may be delayed, the same can happen in a synchronous network where a *corrupt* D may not send the designated messages to a subset of honest parties.

One option to enable the parties $P_i$ *outside* $\mathcal{CS}$ to get their respective pair of points $(F_i^\star(0), R_i^\star(0))$ is to let the parties *inside* $\mathcal{CS}$ provide $P_i$ the supposedly common points on the polynomials $F_i^\star(x)$ and $R_i^\star(x)$, which $P_i$ can verify using $\mathbf{C}$. And once $P_i$ has at least $t_s + 1$ correct points on these polynomials, it can interpolate them and get $(F_i^\star(0), R_i^\star(0))$. This idea will certainly work, if the network is *guaranteed* to be *asynchronous*, since in this case $|\mathcal{H}_{\mathcal{CS}}| \geq n - t_s - t_a > t_s$. Unfortunately, the network type will be *unknown* and it may so happen that the network is *synchronous* and $\mathsf{D}$ is *corrupt*, in which case we are guaranteed to have $|\mathcal{H}_{\mathcal{CS}}| \geq n - t_s - t_s > t_a$, which is *not* sufficient to implement the above idea. Instead, we follow a different approach, which constitutes the crux of the protocol.

Once $\mathcal{CS}$ is publicly identified, $\mathsf{D}$ next *freshly* secret-shares the points $F_i^\star(0)$ and $R_i^\star(0)$, for every $P_i \notin \mathcal{CS}$. The crucial point here is that the degree of sharing now is *only* $t_a$ and *not* $t_s$. We stress that this *does not* violate the privacy of the points $F_i^\star(0)$ and $R_i^\star(0)$ in any network for an *honest* $\mathsf{D}$. This is because if the network is *synchronous*, then *every* party $P_i \notin \mathcal{CS}$ is *guaranteed* to be *corrupt* for an *honest* $\mathsf{D}$ and so the adversary will already be *knowing* the points $F_i^\star(0)$ and $R_i^\star(0)$. On the other hand, if the network is *asynchronous*, then the fresh sharing will not violate the privacy, since there will be at most $t_a$ corrupt parties and the degree of the new sharing is $t_a$. To secret-share the points $F_i^\star(0)$ and $R_i^\star(0)$, the dealer embeds them in random $t_a$-degree polynomials at their constant term, distribute distinct points on these polynomials to respective parties and also publicly commit the coefficients of these polynomials. The parties then verify the received points with respect to the commitments using the procedure VerifyPoint. Moreover, they also verify whether the points which are freshly secret-shared are the *same* which are committed in the existing matrix $\mathbf{C}$. Every party upon verifying both these conditions positively, notifies it in public. The goal is then to let $\mathsf{D}$ publicly identify a "qualified" subset of parties $\mathcal{Q}$ of size $n - t_s$ (who could be different from $\mathcal{CS}$), who responded positively for the fresh secret-sharings done by $\mathsf{D}$. Note that an *honest* $\mathsf{D}$ will always find such a candidate set $\mathcal{Q}$, irrespective of the network type, since $\mathcal{H}$ will always constitute a candidate $\mathcal{Q}$ set. Once the set $\mathcal{Q}$ is identified, then the parties in $\mathcal{Q}$ enable the parties $P_i \notin \mathcal{CS}$ to interpolate the $t_a$-degree polynomials using which $\mathsf{D}$ has freshly shared $F_i^\star(0)$ and $R_i^\star(0)$. For this, the parties in $\mathcal{Q}$ provide their respective points on these fresh polynomials to $P_i$, who can identify the correct points by using the procedure VerifyPoint. Note that $P_i$ will need only $t_a + 1$ correct points now, which are bound to arrive in *any* network, since $\mathcal{Q}$ is bound to have at least $t_a + 1$ honest party in any network. This completes the description of the protocol for the case when a candidate $\mathcal{CS}$ is identified within the designated time of $3T_{\mathsf{BC}}$.

Let us now consider the *second* case when *no* candidate $\mathcal{CS}$ is identified within the designated time of $3T_{\mathsf{BC}}$. This case is relatively simpler. In this case we already know that either $\mathsf{D}$ is *corrupt* or the network is *asynchronous*. Consequently, $\mathsf{D}$ is now asked to look for a candidate $\mathcal{CS}$ of size at least $n - t_a$, who responded positively for the matrix $\mathbf{C}$ and the points received from $\mathsf{D}$. Note that an *honest* $\mathsf{D}$ is guaranteed to get such a $\mathcal{CS}$ because in this case the network

is *asynchronous* and so $|\mathcal{H}| \geq n - t_a$ and hence $\mathcal{H}$ will eventually constitute a candidate $\mathcal{CS}$. Once $\mathcal{CS}$ (of size at least $n - t_a$) is identified, then the parties *inside* $\mathcal{CS}$ can help the parties $P_i$ *outside* $\mathcal{CS}$ to get their respective pair of points $(F_i^\star(0), R_i^\star(0))$ by supplying them the supposedly common points on the polynomials $F_i^\star(x)$ and $R_i^\star(x)$. Party $P_i$ can then identify the correct points (using $\mathbf{C}$ and procedure $\mathsf{VerifyPoint}$) and once $P_i$ has $t_s + 1$ correct points, it can interpolate them and get $(F_i^\star(0), R_i^\star(0))$. Interestingly, the availability of at least $t_s + 1$ correct points from the parties in the *new* $\mathcal{CS}$ is *always* guaranteed, even if the network is *synchronous*. This is because now $|\mathcal{CS}| \geq n - t_a$ and hence has $n - t_a - t_s > t_s$ honest parties even in a *synchronous* network.

This completes the description of the protocol $\Pi_{\mathsf{VSS}}$, which is presented in Fig. 3. There are *two* cases in the protocol, depending upon whether the parties identify a candidate core set of size at least $n - t_s$ within the designated time $3T_{\mathsf{BC}}$. For the *first* case, $\mathsf{D}$ has to secret-share values *two* times, first while distributing points on bivariate polynomials and second while again secret-sharing the shares of the parties who are outside the candidate core set. To distinguish between these two types of sharing, we use the terms *primary* and *secondary* sharing respectively.

The properties of the protocol $\Pi_{\mathsf{VSS}}$ are stated in Theorem 1.

**Theorem 1.** *Let $t_a < t_s$ and $t_a + 2t_s < n$ and let $\mathsf{D}$ has an input $s \in \mathbb{F}_p$ for $\Pi_{\mathsf{VSS}}$. Moreover, let $T_{\mathsf{VSS}} = 6T_{\mathsf{BC}} + T_{\mathsf{BA}} + \Delta$. Then, the protocol $\Pi_{\mathsf{VSS}}$ achieves the following properties.*

- *If $\mathsf{D}$ is honest, then the following hold.*
  - *$t_s$-correctness: In a synchronous network, $s$ is $[\cdot]$-shared at time $T_{\mathsf{VSS}}$.*
  - *$t_a$-correctness: In an asynchronous network, $s$ is eventually $[\cdot]$-shared.*
  - *$t_s$-privacy: Irrespective of the network type, the view of the adversary remains independent of $s$.*
- *If $\mathsf{D}$ is corrupt, then either no honest party computes any output, or there exists some $s^\star \in \mathbb{F}_p$ such that the following holds.*
  - *$t_s$-strong commitment: In a synchronous network, $s^\star$ is $[\cdot]$-shared, such that one of the following holds.*
    - *If any honest party computes its output at time $T_{\mathsf{VSS}}$, then all honest parties compute its output at time $T_{\mathsf{VSS}}$.*
    - *If any honest party computes its output at time $T > T_{\mathsf{VSS}}$, then every honest party computes its output by time $T + \Delta$.*
  - *$t_a$-strong commitment: In an asynchronous network, $s^\star$ is eventually $[\cdot]$-shared.*
- *Irrespective of the network type, the protocol incurs a communication of $\mathcal{O}(n^5 \cdot \kappa)$ bits from the honest parties and invokes 1 instance of $\Pi_{\mathsf{BA}}$.*

**Protocol $\Pi_{\mathsf{VSS}}$ for $L$ Inputs.** Protocol $\Pi_{\mathsf{VSS}}$ can be easily modified to handle the case when $\mathsf{D}$ has $L$ inputs where $L \geq 1$, such that the number of $\Pi_{\mathsf{BA}}$ instances in the protocol is *only* one and remains *independent* of $L$. The idea is to execute the steps of the protocol $\Pi_{\mathsf{VSS}}$ $L$ times, once on behalf of each input of $\mathsf{D}$. However, *instead* of finding $L$ candidate $\mathcal{CS}$ or $\mathcal{Q}$ sets, the dealer $\mathsf{D}$ finds a

**Protocol $\Pi_{\mathsf{VSS}}$**

- **Phase I: Primary-Share Distribution** — D on having the input $s$, does the following:
  - Pick a random $t_s$-degree polynomial $q(\cdot)$, such that $q(0) = s$.
  - Pick two random $(t_s, t_s)$-degree symmetric bivariate polynomials $F(x, y)$ and $R(x, y)$, such that $F(0, y) = q(\cdot)$.
  - For $i = 1, \ldots, n$, let $f_i(x) = F(x, i) = a_{i0} + a_{i1}x + \ldots + a_{it_s}x^{t_s}$ and let $r_i(x) = R(x, i) = b_{i0} + b_{i1}x + \ldots + b_{it_s}x^{t_s}$. For $j = 0, \ldots, t_s$, compute the commitment $\mathsf{Com}_{ij} = \mathsf{Commit}(a_{ij}, b_{ij})$.
  - For $i = 1, \ldots, n$, send the vector of values $(f_{i1}, \ldots, f_{in})$ and $(r_{i1}, \ldots, r_{in})$ to $P_i$, where for $j = 1, \ldots, n$, the condition $f_{ij} = f_i(j)$ and $r_{ij} = r_i(j)$ holds.
  - Let $\mathbf{C}$ be the $n \times (t_s + 1)$ matrix, where for $i = 1, \ldots, n$ the $i^{th}$ row consists of $(\mathsf{Com}_{i0}, \ldots, \mathsf{Com}_{it_s})$. Broadcast the matrix $\mathbf{C}$.
- **Phase II: Publicly Declaring the Consistency of Primary-Shares** — Each $P_i \in \mathcal{P}$ (including D) upon receiving the vector of values $(f_{i1}, \ldots, f_{in})$ and $(r_{i1}, \ldots, r_{in})$ from D, and the matrix $\mathbf{C}$ from D's broadcast through *regular* mode, wait till the current local time becomes an integer multiple of $\Delta$. It then broadcasts $(\mathsf{ReceivedPrimary}, i)$ if *all* the following holds:
  - $\mathbf{C}$ is an $n \times (t_s + 1)$ matrix.
  - $\mathsf{VerifyPoly}(\mathbf{C}, t_s)$ outputs 1.
  - For $i = 1, \ldots, n$, let $(\mathsf{Com}_{i0}, \ldots, \mathsf{Com}_{it_s})$ be the $i^{th}$ row of $\mathbf{C}$. Then $\mathsf{VerifyPoint}(\{\mathsf{Com}_{i0}, \ldots, \mathsf{Com}_{it_s}\}, (f_{ij}, r_{ij}), j)$ outputs 1, for $j = 1, \ldots, n$.
- **Phase III: Publicly Declaring the Core Set** — D does the following at time $2T_{\mathsf{BC}}$:
  - Initialize a set $\mathcal{CS} = \emptyset$.
  - Include $P_i$ to $\mathcal{CS}$, if $(\mathsf{ReceivedPrimary}, i)$ is received from the broadcast of $P_i$ through *regular* mode.
  - If $|\mathcal{CS}| \geq n - t_s$, then broadcast $\mathcal{CS}$.
- **Local Computation: Verifying and Accepting the Core Set** — Each $P_i \in \mathcal{P}$ (including D) does the following at time $3T_{\mathsf{BC}}$:
  - If a set $\mathcal{CS}$ is received from D's broadcast through *regular* mode, such that $|\mathcal{CS}| \geq n - t_s$, then *accept* it if $(\mathsf{ReceivedPrimary}, j)$ *was* received from the broadcast of every $P_j \in \mathcal{CS}$, through *regular* mode at time $2T_{\mathsf{BC}}$.
- **Phase IV: Deciding Whether to Go for a Core Set of Size** $(n - t_a)$ — At time $3T_{\mathsf{BC}}$, each $P_i \in \mathcal{P}$ participates in an instance of $\Pi_{\mathsf{BA}}$ with input $b_i = 1$ if a $\mathcal{CS}$ is accepted, else with input $b_i = 0$, and waits for time $T_{\mathsf{BA}}$.

**Case I (If a Core Set of Size $n - t_s$ is Found)**

This part is executed if the output of $\Pi_{\mathsf{BA}}$ is 1.

- **Phase V: Distributing Secondary-Shares of Non-Core Parties by** D —For each $P_i \notin \mathcal{CS}$, D does the following:
  - Select a random $t_a$-degree polynomial $m_i(x)$ such that $m_i(0) = f_i(0) = a_{i0}$. Let $m_i(x) = a_{i0} + e_{i1}x + \ldots + e_{it_a}x^{t_a}$.
  - Select a random $t_a$-degree polynomial $c_i(x)$ such that $c_i(0) = r_i(0) = b_{i0}$. Let $c_i(x) = b_{i0} + u_{i1}x + \ldots + u_{it_a}x^{t_a}$.
  - For $k = 1, \ldots, t_a$, compute the commitment $\mathsf{Sec\text{-}Com}_{ik} = \mathsf{Commit}(e_{ik}, u_{ik})$.
  - For $j = 1, \ldots, n$, send $m_{ij}$ and $c_{ij}$ to party $P_j$, where $m_{ij} = m_i(j)$ and $c_{ij} = c_i(j)$.

  D then broadcasts the commitments $\{\mathsf{Sec\text{-}Com}_{ik}\}_{P_i \notin \mathcal{CS}, k=1,\ldots,t_a}$.
- **Phase VI: Publicly Declaring the Consistency of Secondary-Shares** — Each $P_j \in \mathcal{P}$ does the following:
  - If $\mathcal{CS}$ and $\mathbf{C}$ is not yet received from D's broadcast, then wait to receive them from D's broadcast through *fallback* mode.
  - Upon receiving the points $\{m_{ij}, c_{ij}\}_{P_i \notin \mathcal{CS}}$ from D and the commitments $\{\mathsf{Sec\text{-}Com}_{ik}\}_{P_i \notin \mathcal{CS}, k=1,\ldots,t_a}$ from D's broadcast (through *any* mode), wait till the current local time becomes an integer multiple of $\Delta$ and then broadcast $(\mathsf{ReceivedSecondary}, j)$ if the following holds for *every* $P_i \notin \mathcal{CS}$:
    - $\mathsf{VerifyPoint}(\{\mathsf{Com}_{i0}, \mathsf{Sec\text{-}Com}_{i1}, \ldots, \mathsf{Sec\text{-}Com}_{it_a}\}, (m_{ij}, c_{ij}), j)$ outputs 1. Here $(\mathsf{Com}_{i0}, \ldots, \mathsf{Com}_{it_s})$ denotes the $i^{th}$ row of $\mathbf{C}$.

**Fig. 3.** Network-agnostic VSS protocol

- **Phase VII: Publicly Declaring the** Qualified **set** — D does the following after (local) time $5T_{\mathsf{BC}} + T_{\mathsf{BA}}$:
  - Initialize a set $\mathcal{Q} = \emptyset$.
  - Include $P_j$ to $\mathcal{Q}$, if $(\mathsf{ReceivedSecondary}, j)$ is received from the broadcast of $P_j$ (through *any* mode).
  - If $|\mathcal{Q}| \geq n - t_s$, then broadcast $\mathcal{Q}$.
- **Local Computation: Verifying and Accepting the** Qualified **Set** — Every party in $\mathcal{P}$ (including D) does the following at time $6T_{\mathsf{BC}} + T_{\mathsf{BA}}$:
  - If a set $\mathcal{Q}$ is received from D's broadcast (through *any* mode) such that $|\mathcal{Q}| \geq n - t_s$, then *accept* it if $(\mathsf{ReceivedSecondary}, j)$ is received from the broadcast of every $P_j \in \mathcal{Q}$ (through *any* mode).
- **Phase VIII: Sending Secondary-Shares to the Non-Core Parties** — Each $P_j \in \mathcal{P}$ (including D) does the following:
  - If $P_j \in \mathcal{Q}$ and $P_j$ has *accepted* $\mathcal{Q}$, then send $(m_{ij}, c_{ij})$ to every $P_i \notin \mathcal{CS}$.
- **Output Computation** — Each $P_i \in \mathcal{P}$ waits till the current local time becomes an integer multiple of $\Delta$ and then does the following:
  - If $P_i \in \mathcal{CS}$, then output $(f_i(0), r_i(0))$ and the commitments $\{\mathsf{Com}_{k0}\}_{k=1,\ldots,n}$.
  - Else if $P_i \notin \mathcal{CS}$ and if $P_i$ has *accepted* $\mathcal{Q}$, then do the following:
    - Initialize a set $\mathcal{SS}_i = \emptyset$.
    - Upon receiving $(m_{ij}, c_{ij})$ from $P_j \in \mathcal{Q}$, include $P_j$ to $\mathcal{SS}_i$ if $\mathsf{VerifyPoint}(\{\mathsf{Com}_{i0}, \mathsf{Sec\text{-}Com}_{i1}, \ldots, \mathsf{Sec\text{-}Com}_{it_a}\}, (m_{ij}, c_{ij}), j)$ outputs 1.
    - If $|\mathcal{SS}_i| > t_a$, interpolate the points $\{(j, m_{ij})\}_{P_j \in \mathcal{SS}_i}$ and $\{(j, c_{ij})\}_{P_j \in \mathcal{SS}_i}$ to obtain $t_a$-degree polynomials $m_i(x)$ and $c_i(x)$ respectively. Set $f_i(0) = m_i(0)$ and $r_i(0) = c_i(0)$, then output $(f_i(0), r_i(0))$ and $\{\mathsf{Com}_{k0}\}_{k=1,\ldots,n}$.

**Case II (If a Core Set of Size $n - t_s$ is Not Found)**

This part is executed if the output of $\Pi_{\mathsf{BA}}$ is 0

- **Phase V: Broadcasting a Core Set of Size** $(n - t_a)$ —D does the following:
  - Upon receiving $(\mathsf{ReceivedPrimary}, i)$ from the broadcast of $P_i$ (through *any* mode) include $P_i$ to $\mathcal{CS}$.
  - If $|\mathcal{CS}| \geq n - t_a$ , then broadcast $\mathcal{CS}$.
- **Local Computation: Verifying and Accepting the Core Set of Size** $(n - t_a)$ — Each $P_i \in \mathcal{P}$ does the following:
  - Participate in any instance of $\Pi_{\mathsf{BC}}$ invoked by D for broadcasting $\mathcal{CS}$ *only* after time $6T_{\mathsf{BC}} + T_{\mathsf{BA}} + \Delta$. Wait till some $\mathcal{CS}$ is obtained from D's broadcast (through *any* mode) and *accept* it if the following holds:
    - $|\mathcal{CS}| \geq n - t_a$.
    - $(\mathsf{ReceivedPrimary}, j)$ is received from the broadcast of all $P_j \in \mathcal{CS}$ (through *any* mode).
- **Phase VI: Sending Common Points to the Parties Outside** $\mathcal{CS}$ — Each $P_i \in \mathcal{P}$ does the following:
  - If $P_i \in \mathcal{CS}$ and $P_i$ has *accepted* $\mathcal{CS}$, then send $(f_{ij}, r_{ij})$ to each $P_j \notin \mathcal{CS}$.
- **Output Computation** — Each $P_i \in \mathcal{P}$ waits till the current local time becomes an integer multiple of $\Delta$ and then does the following:
  - If $P_i \in \mathcal{CS}$, then output $(f_i(0), r_i(0))$ and the commitments $\{\mathsf{Com}_{k0}\}_{k=1,\ldots,n}$.
  - Else if $P_i$ has *accepted* $\mathcal{CS}$, then do the following:
    - Initialize a set $\mathcal{SS}_i = \emptyset$.
    - Upon receiving $(f_{ji}, r_{ji})$ from $P_j \in \mathcal{CS}$, include $P_j$ to $\mathcal{SS}_i$ if the following holds:
      - $\mathsf{VerifyPoint}(\{\mathsf{Com}_{j0}, \mathsf{Com}_{j1}, \ldots, \mathsf{Com}_{jt_s}\}, (f_{ji}, r_{ji}), i)$ outputs 1.
    - If $|\mathcal{SS}_i| \geq t_s + 1$, interpolate the points $\{(j, f_{ji})\}_{P_j \in \mathcal{SS}_i}$ and $\{(j, r_{ji})\}_{P_j \in \mathcal{SS}_i}$ to obtain $t_s$-degree polynomials $f_i(x)$ and $r_i(x)$ respectively. Then output $(f_i(0), r_i(0))$ and $\{\mathsf{Com}_{k0}\}_{k=1,\ldots,n}$.

**Fig. 3.** (*continued*)

*single* $\mathcal{CS}$ or $\mathcal{Q}$ set. For this, each party broadcasts a *single* $(\mathsf{ReceivedPrimary}, \star)$ message or $(\mathsf{ReceivedSecondary}, \star)$ message (instead of $L$ such messages), if the conditions for broadcasting these messages are satisfied on behalf of *all* the $L$ inputs of $\mathsf{D}$. As the modifications are straightforward, we avoid the details and note that the protocol incurs a communication of $\mathcal{O}(n^5 \cdot L \cdot \kappa)$ bits from the honest parties and invokes 1 instance of $\Pi_{\mathsf{BA}}$.

## 5 Agreement on a Common Subset (ACS)

In this section, we adapt the ACS protocol proposed by [3] to our specific setting by incorporating our network-agnostic VSS protocol $\Pi_{\mathsf{VSS}}$ and the network-agnostic BA protocol $\Pi_{\mathsf{BA}}$ of [14]. The ACS protocol, denoted as $\Pi_{\mathsf{ACS}}$, will be utilized in both our preprocessing phase protocol and the circuit-evaluation protocol. In the protocol, each party is supposed to $[\cdot]$-share $L$ input values using instances of $\Pi_{\mathsf{VSS}}$.[2] The goal of $\Pi_{\mathsf{ACS}}$ is to enable the parties to agree upon a common subset of parties $\mathsf{CS}$ of size at least $n - t_s$, such that the inputs of all the parties in $\mathsf{CS}$ are $[\cdot]$-shared. Additionally, in a synchronous network, all honest parties are present in $\mathsf{CS}$.

The underlying idea of $\Pi_{\mathsf{ACS}}$ is as follows: Each party $P_i \in \mathcal{P}$ acts as a dealer and invokes an instance $\Pi_{\mathsf{VSS}}^{(i)}$ of the protocol, $\Pi_{\mathsf{VSS}}$, to verifiably $[\cdot]$-share its inputs. In a *synchronous* network, after time $T_{\mathsf{VSS}}$, the inputs of all *honest* parties should be $[\cdot]$-shared. Consequently, after (local) time $T_{\mathsf{VSS}}$, the parties examine the instances of $\Pi_{\mathsf{VSS}}$ in which they computed their respective outputs. Based on this information, the parties engage in $n$ instances of the protocol $\Pi_{\mathsf{BA}}$, where the $j^{th}$ instance is to determine whether $P_j$ should be included in $\mathsf{CS}$. The input criteria for these $\Pi_{\mathsf{BA}}$ instances are as follows: if a party has computed output during the instance $\Pi_{\mathsf{VSS}}^{(j)}$, then it participates with input 1 in the $j^{th}$ instance $\Pi_{\mathsf{BA}}^{(j)}$ of $\Pi_{\mathsf{BA}}$. Once at least $n - t_s$ instances of $\Pi_{\mathsf{BA}}$ yield an output of 1, the parties participate with input 0 in any remaining $\Pi_{\mathsf{BA}}$ instances for which they have not yet provided any input. Finally, after obtaining outputs from all $n$ instances of $\Pi_{\mathsf{BA}}$, party $P_j$ is included in $\mathsf{CS}$ iff the output of the $\Pi_{\mathsf{BA}}^{(j)}$ instance is 1. Since the parties wait for time $T_{\mathsf{VSS}}$ before initiating the $\Pi_{\mathsf{BA}}$ instances, it is guaranteed that all honest dealers are included in $\mathsf{CS}$ in a synchronous network.

The properties of the protocol $\Pi_{\mathsf{ACS}}$ stated in Lemma 5 follows from [3].

**Lemma 5.** *Let $t_a < t_s$ and $t_a + 2t_s < n$. Then, the protocol $\Pi_{\mathsf{ACS}}$ achieves the following properties, where every party $P_i$ has $L$ values $(s_{(i,1)}, \ldots, s_{(i,L)})$ from $\mathbb{F}_p$ as input.*

- *$t_s$-correctness: If the network is synchronous, then at time $T_{\mathsf{ACS}} = T_{\mathsf{VSS}} + 2T_{\mathsf{BA}}$, the honest parties have a common subset $\mathsf{CS}$ of size at least $n - t_s$, such that all the following holds:*
  - *All honest parties will be present in $\mathsf{CS}$.*

[2] The exact inputs depend upon the underlying context.

  - *Corresponding to every $P_i \in \mathsf{CS}$, there exist $L$ values $(s^{\star}_{(i,1)}, \ldots, s^{\star}_{(i,L)})$, which are same as $(s_{(i,1)}, \ldots, s_{(i,L)})$ for an honest $P_i$, such that $s^{\star}_{(i,1)}, \ldots, s^{\star}_{(i,L)}$ are $[\cdot]$-shared among the parties at time $T_{\mathsf{ACS}}$.*
- *$t_a$-correctness: If the network is asynchronous, then the honest parties eventually output a common subset $\mathsf{CS}$ of size at least $n - t_s$. Moreover, corresponding to every $P_i \in \mathsf{CS}$, there exist $L$ values $(s^{\star}_{(i,1)}, \ldots, s^{\star}_{(i,L)})$, which are same as $(s_{(i,1)}, \ldots, s_{(i,L)})$ for an honest $P_i$, such that $s^{\star}_{(i,1)}, \ldots, s^{\star}_{(i,L)}$ are eventually $[\cdot]$-shared among the parties.*
- *$t_s$-privacy: Irrespective of the network type, the view of the adversary remains independent of the inputs of the honest parties.*
- *Irrespective of the network type, the protocol incurs a communication of $\mathcal{O}(n^6 \cdot L \cdot \kappa)$ bits from the honest parties and invokes $\mathcal{O}(n)$ instances of $\Pi_{\mathsf{BA}}$.*

## 6 The Preprocessing Phase Protocol

We now present our network-agnostic preprocessing protocol in this section. The protocol aims to produce $c_M$ number of $[\cdot]$-shared multiplication-triples that are random from the adversary's point of view. The protocol is obtained by adapting a similar protocol from [3] to the setting where $t_a + 2t_s < n$.[3] We begin by describing the various building blocks used in the protocol. The current description is mostly taken from [3] and we refer to [3] for the complete proofs.

### 6.1 Network-Agnostic Beaver's Multiplication Protocol

Protocol $\Pi_{\mathsf{Beaver}}(([x],[y]),([a],[b],[c]))$ takes as inputs $[\cdot]$-shared $x, y$ and a $[\cdot]$-shared triple $(a, b, c)$ and outputs a $[\cdot]$-shared $z$, where $z = x \cdot y$, if and only if $c = a \cdot b$. During the protocol, the parties first locally compute $[e] = [x] - [a]$ and $[d] = [y] - [b]$ and then publicly reconstruct $e$ and $d$, using the reconstruction protocol $\Pi_{\mathsf{RecPub}}$. Using these values, a $[\cdot]$-sharing of $z$ is then computed locally as $[z] = e \cdot d + e \cdot [b] + d \cdot [a] + [c]$. One can see that if $(a, b, c)$ is random from the adversary's point of view, then $x$ and $y$ will remain private from the point of view of the adversary. The protocol takes $\Delta$ time to output $[z]$ in a synchronous network, while it outputs $[z]$ eventually in an asynchronous network. Irrespective of the network type, the protocol incurs a communication of $\mathcal{O}(n^2 \cdot \kappa)$ bits from the honest parties.

### 6.2 Network-Agnostic Triple-Transformation Protocol

Protocol $\Pi_{\mathsf{TripTrans}}$ takes as input a set of $2d + 1$ $[\cdot]$-shared triples $\{([x^{(i)}], [y^{(i)}], [z^{(i)}])\}_{i=1,\ldots,2d+1}$, where the triples may not be "related". The protocol outputs "co-related" $[\cdot]$-shared triples $\{([\mathsf{x}^{(i)}], [\mathsf{y}^{(i)}], [\mathsf{z}^{(i)}])\}_{i=1,\ldots,2d+1}$, satisfying the following properties *regardless* of the network type:

[3] In [3], the protocol was presented for the condition $t_a + 3t_s < n$.

- There exist $d$-degree polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and a $2d$-degree polynomial $\mathsf{Z}(\cdot)$, such that $\mathsf{X}(i) = \mathsf{x}^{(i)}$, $\mathsf{Y}(i) = \mathsf{y}^{(i)}$ and $\mathsf{Z}(i) = \mathsf{z}^{(i)}$ holds for $i = 1, \ldots, 2d+1$.
- The triple $(\mathsf{x}^{(i)}, \mathsf{y}^{(i)}, \mathsf{z}^{(i)})$ is a multiplication-triple if and only if $(x^{(i)}, y^{(i)}, z^{(i)})$ is a multiplication-triple. Consequently, $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot) \cdot \mathsf{Y}(\cdot)$ holds iff all the $2d+1$ input triples are multiplication-triples.
- The adversary learns the triple $(\mathsf{x}^{(i)}, \mathsf{y}^{(i)}, \mathsf{z}^{(i)})$ iff it knows the input triple $(x^{(i)}, y^{(i)}, z^{(i)})$.

The protocol is *identical* to a same protocol with *perfect security* proposed in [3]; the only difference is that they use a *perfectly-secure* version of the Beaver's multiplication protocol (since $t_s < \frac{n}{3}$ holds for them), while we use a cryptographically-secure version of the Beaver's multiplication protocol (since $t_s < \frac{n}{2}$ holds for our setting). In a synchronous network, the protocol outputs the transformed triples by time $\Delta$, while in an asynchronous network, the parties eventually output the transformed triples. The protocol incurs a communication of $\mathcal{O}(n^2 \cdot d \cdot \kappa)$ bits.

The protocol proceeds as follows: the first and second components of the first $d+1$ input triples are used to define the $d$-degree polynomials $\mathsf{X}(\cdot)$ and $\mathsf{Y}(\cdot)$. These points define the first $d+1$ points on these polynomials. Then, using the shares of the first $d+1$ triples, the parties locally compute $[\cdot]$-sharings of $d$ new points on these polynomials. The remaining $d$ input triples are then utilized to calculate the $[\cdot]$-sharing of the product of these new points using the Beaver's multiplication protocol. Consequently, the $\mathsf{Z}(\cdot)$ polynomial is defined by the $d$ computed products and the third component of the first $d+1$ input triples.

### 6.3 Network-Agnostic Protocol for Generating a Random Value

Protocol $\Pi_{\mathsf{Rand}}$ is a cryptographically-secure network-agnostic protocol which enables the parties to generate a random value $r \in \mathbb{F}_p$, which will be known to all the parties at the end of the protocol. The instantiation of $\Pi_{\mathsf{Rand}}$ is based on a random value generation protocol described in [21]. The protocol proceeds as follows: the parties invoke an instance of the protocol $\Pi_{\mathsf{ACS}}$, where the input of each party is a random element from $\mathbb{F}_p$. Protocol $\Pi_{\mathsf{ACS}}$ ensures that a common subset $\mathsf{CS}$ of at least $n - t_s$ parties is agreed upon, such that all the parties in $\mathsf{CS}$ have $[\cdot]$-shared their (random) values. The value $r$ is then set to the sum of the values shared by the parties in $\mathsf{CS}$, which will be available in a $[\cdot]$-shared fashion. The parties then publicly reconstruct $r$ using an instance of $\Pi_{\mathsf{RecPub}}$. Since at least one honest party is guaranteed in $\mathsf{CS}$ (irrespective of the network type), whose secret-shared input will be random and unknown to the adversary, it follows that $r$ will be indeed random. The protocol takes $T_{\mathsf{Rand}} = T_{\mathsf{ACS}} + \Delta$ time in a *synchronous* network to generate the output, while in an *asynchronous* network, the honest parties eventually get their output. Irrespective of the network type, the protocol incurs a communication of $\mathcal{O}(n^6 \cdot \kappa)$ bits from the honest parties and invokes $\mathcal{O}(n)$ instances of $\Pi_{\mathsf{BA}}$.

### 6.4 Network-Agnostic Polynomial-Verification Protocol

We next describe a network-agnostic polynomial-verification protocol, $\Pi_{\mathsf{PolyVer}}$. In the protocol, there exists a triplet of polynomials $(\mathsf{X}(\cdot), \mathsf{Y}(\cdot), \mathsf{Z}(\cdot))$. Polynomials $\mathsf{X}(\cdot)$ and $\mathsf{Y}(\cdot)$ are $d$-degree polynomials, while $\mathsf{Z}(\cdot)$ is a $2d$-degree polynomial. There will be $2d+1$ distinct points on these polynomials, which will be $[\cdot]$-shared among the parties. The goal of $\Pi_{\mathsf{PolyVer}}$ is to probabilistically verify if $\mathsf{Z}(\cdot) \stackrel{?}{=} \mathsf{X}(\cdot) \cdot \mathsf{Y}(\cdot)$ holds. The instantiation of $\Pi_{\mathsf{PolyVer}}$ is based on a polynomial verification protocol presented in [21].

The basic idea of the protocol is as follows: first, a random value $\alpha$ is generated using an instance of $\Pi_{\mathsf{Rand}}$. The parties then check if $\mathsf{Z}(\alpha) \stackrel{?}{=} \mathsf{X}(\alpha) \cdot \mathsf{Y}(\alpha)$ holds. For this, the parties compute $\mathsf{X}(\alpha), \mathsf{Y}(\alpha)$ and $\mathsf{Z}(\alpha)$ in a $[\cdot]$-shared fashion, followed by publicly reconstructing these values using instances of $\Pi_{\mathsf{RecPub}}$. Since $\alpha$ is selected randomly, if the polynomials $(\mathsf{X}(\cdot), \mathsf{Y}(\cdot), \mathsf{Z}(\cdot))$ *do not* satisfy the multiplicative relationship, then the above test will fail, except with probability at most $\frac{2d}{|\mathbb{F}_p|}$. During the verification process, the only information learnt by the adversary are the points $\mathsf{X}(\alpha)$, $\mathsf{Y}(\alpha)$ and $\mathsf{Z}(\alpha)$. In a *synchronous* network, the protocol will generate output after time $T_{\mathsf{PolyVer}} = T_{\mathsf{Rand}} + \Delta$, while in an *asynchronous* network, the parties get output eventually. The protocol incurs a communication of $\mathcal{O}(n^6 \cdot \kappa)$ bits from the honest parties.

### 6.5 Network-Agnostic Triple-Sharing Protocol

The network-agnostic triple-sharing protocol $\Pi_{\mathsf{TripSh}}$ allows a designated *dealer* $\mathsf{D}$ to verifiably $[\cdot]$-share $t_s$ multiplication-triples. The protocol ensures that if the dealer is *honest* then the triples remain random from the adversary's view and all honest parties output the shares of the dealer's multiplication-triples. The "verifiability" here guarantees that if $\mathsf{D}$ is *corrupt*, then either no honest party computes any output (if $\mathsf{D}$ *does not* invoke the protocol) or there exist $t_s$ multiplication-triples, which are $[\cdot]$-shared among the parties. The protocol is borrowed from [3,21].

The protocol begins with the dealer $[\cdot]$-sharing $2t_s+1$ random multiplication-triples, denoted as $(x^{(j)}, y^{(j)}, z^{(j)})_{j=1,\ldots,2t_s+1}$, using an instance of $\Pi_{\mathsf{VSS}}$. The verifiability property of $\Pi_{\mathsf{VSS}}$ ensures that the shared triples are $[\cdot]$-shared. However, there is no guarantee that these shared triples are actually multiplication-triples. To verify if the $[\cdot]$-shared triples are indeed multiplication-triples, the $[\cdot]$-sharing of these triples is transformed to $[\cdot]$-sharing of triples $(\mathsf{x}^{(j)}, \mathsf{y}^{(j)}, \mathsf{z}^{(j)})_{j=1,\ldots,2t_s+1}$ using an instance of $\Pi_{\mathsf{TripTrans}}$. The transformed triples have associated polynomials $\mathsf{X}(\cdot)$, $\mathsf{Y}(\cdot)$, and $\mathsf{Z}(\cdot)$, with degrees $t_s$, $t_s$, and $2t_s$, respectively. If the dealer is honest, the adversary learns no information about $\mathsf{X}(\cdot)$, $\mathsf{Y}(\cdot)$, and $\mathsf{Z}(\cdot)$.

Next, the relationship $\mathsf{Z}(\cdot) \stackrel{?}{=} \mathsf{X}(\cdot) \cdot \mathsf{Y}(\cdot)$ is verified by executing the $\Pi_{\mathsf{PolyVer}}$ protocol. It follows from the properties of $\Pi_{\mathsf{TripTrans}}$ and $\Pi_{\mathsf{PolyVer}}$ that $\Pi_{\mathsf{PolyVer}}$ outputs 1 iff all the input triples $(x^{(j)}, y^{(j)}, z^{(j)})_{j=1,\ldots,2t_s+1}$ are multiplication-triples, except with an error probability of $\frac{2t_s}{|\mathbb{F}_p|}$. This is because a corrupt dealer

will only learn the random verification point used for verifying the multiplicative relationship during $\Pi_{\mathsf{PolyVer}}$ after sharing the triples.

If $\mathsf{D}$ is *honest* then the adversary learns only one point on $\mathsf{X}(\cdot)$, $\mathsf{Y}(\cdot)$, and $\mathsf{Z}(\cdot)$ during the verification process. This leaves $t_s + 1 - 1 = t_s$ degrees of freedom in these polynomials. If $\Pi_{\mathsf{PolyVer}}$ outputs 1, the parties output $t_s$ $[\cdot]$-shared triples $([a^{(i)}], [b^{(i)}], [c^{(i)}])$ on behalf of the dealer $\mathsf{D}$, where $a^{(i)} = \mathsf{X}(\beta^{(i)})$, $b^{(i)} = \mathsf{Y}(\beta^{(i)})$, and $c^{(i)} = \mathsf{Z}(\beta^{(i)})$. Here $\beta^{(1)}, \ldots, \beta^{(t_s)}$ are distinct elements from $\mathbb{F}_p$, which are *different* from the random verification point, used during $\Pi_{\mathsf{PolyVer}}$ and also from the evaluation points of the parties. The shared triples $\{([a^{(i)}], [b^{(i)}], [c^{(i)}])\}_{i=1,\ldots,t_s}$ constitute the actual multiplication-triples that are $[\cdot]$-shared on behalf of the dealer. If $\Pi_{\mathsf{PolyVer}}$ outputs 0, then the parties output a default $[\cdot]$-sharing of $(0, 0, 0)$ $t_s$ times on behalf of the dealer. In a *synchronous* network, the protocol generates output after time $T_{\mathsf{TripSh}} = T_{\mathsf{VSS}} + T_{\mathsf{PolyVer}} + \Delta$, while in an *asynchronous* network, the parties get their output eventually. The protocol incurs a communication of $\mathcal{O}(n^6 \cdot \kappa)$ bits and invokes 1 instance of $\Pi_{\mathsf{BA}}$.

### 6.6 Network-Agnostic Triple-Extraction Protocol

The next protocol $\Pi_{\mathsf{TripExt}}$ takes as input a publicly known set of $2d + 1$ parties denoted as $\mathcal{CS}$, where $d \geq t_s$, and where each party in $\mathcal{CS}$ has $[\cdot]$-shared a multiplication-triple. Notably, the multiplication-triples shared by honest parties in $\mathcal{CS}$ are random for the adversary. The output of the protocol consists of $d+1-t_s$ $[\cdot]$-shared multiplication-triples, which remain random from the view of the adversary. The protocol is *identical* to a similar protocol with *perfect security* proposed in [3]; the only difference is that they used a *perfectly-secure* version of the triple-transformation protocol, while we use our cryptographically-secure instantiation of the same.

The underlying idea of the protocol is as follows: the parties invoke an instance of $\Pi_{\mathsf{TripTrans}}$ to "transform" the input triples into a set of correlated triples. Since the input for the protocol consists of multiplication-triples, this transformation process ensures that the output triples are also multiplication-triples. Let $\mathsf{X}(\cdot)$, $\mathsf{Y}(\cdot)$, and $\mathsf{Z}(\cdot)$ represent the polynomials associated with the transformed triples. The properties of $\Pi_{\mathsf{TripTrans}}$ guarantee that the adversary has knowledge of at most $t_s$ points on these polynomials, thereby implying that a minimum of $d+1-t_s$ points on these polynomials are random from the adversary's view. Consequently, the parties locally compute and generate $d + 1 - t_s$ "new" points on these polynomials, which are guaranteed to appear random to the adversary.

In a *synchronous* network, the protocol takes $\Delta$ time to generate the output, while in *asynchronous* network, the honest parties eventually get their output. The protocol incurs a communication of $\mathcal{O}(n^2 \cdot d \cdot \kappa)$ bits from the honest parties.

### 6.7 The Network-Agnostic Preprocessing Phase Protocol

We finally present our cryptographically-secure network-agnostic preprocessing phase protocol, which generates $[\cdot]$-sharing of $c_M$ multiplication-triples, which

are random from the point of view of the adversary. The protocol is similar to the preprocessing phase protocol of [3] with *perfect security*; the only difference is that we now use various components which are cryptographically-secure. Moreover, we also use a trick to get a *better* communication complexity, compared to [3].

The protocol proceeds as follows: each party $P_i$ acts as a dealer and invokes an instance $\Pi^{(i)}_{\mathsf{TripSh}}$ of $\Pi_{\mathsf{TripSh}}$ to share $c_M$ random multiplication-triples on its behalf. Corrupt parties $P_j$ may choose not to invoke their $\Pi^{(j)}_{\mathsf{TripSh}}$ instances. Consequently, the parties employ a similar approach as in the protocol $\Pi_{\mathsf{ACS}}$ to agree on a common subset of parties $\mathcal{CS}$ of size $n-t_s$, whose multiplication-triples will be $[\cdot]$-shared. For this, the parties invoke instances of the network-agnostic BA protocol $\Pi_{\mathsf{BA}}$. Since the adversary will not know the multiplication-triples shared by the honest parties in $\mathcal{CS}$, the parties execute $c_M$ instances of $\Pi_{\mathsf{TripExt}}$ on the multiplication-triples shared by the parties in $\mathcal{CS}$ to extract random $[\cdot]$-shared multiplication-triples.

For simplicity and without loss of generality, let $|\mathcal{CS}| = 2d + 1$. Note that $d \geq t_s$ *need not* hold here, since $n-t_s \geq t_s+t_a+1$ and $t_a < t_s$.[4] Consequently, by applying the procedure $\Pi_{\mathsf{TripExt}}$, it is *not* guaranteed that the resultant extracted secret-shared multiplication-triples will be indeed random for the adversary.[5] To get rid of this, we deploy the following trick: if $|\mathcal{CS}| < 2t_s+1$, then we add *dummy* parties in $\mathcal{CS}$ and consider a default $[\cdot]$-sharing of $(0, 0, 0)$ on their behalf, followed by applying the procedure $\Pi_{\mathsf{TripExt}}$ on the multiplication-triples of the "extended" $\mathcal{CS}$. This will *always* result in $[\cdot]$-sharing of $c_M$ random multiplication-triples. For instance, if the network is *synchronous*, then we know that *all honest* parties are guaranteed to be in $\mathcal{CS}$, since the parties start executing the instances of $\Pi_{\mathsf{BA}}$ *only after* time $T_{\mathsf{TripSh}}$, ensuring that the multiplication-triples of all honest parties are $[\cdot]$-shared through their respective $\Pi_{\mathsf{TripSh}}$ instances. Consequently, the dummy parties added in a *synchronous* network are *guaranteed* to be corrupt. On the other hand, if the network is *asynchronous*, then the dummy parties added to $\mathcal{CS}$ might be honest; however, in this case, there will be at least $n - t_s - t_a > t_s$ honest parties *already* present in the "non-extended" $\mathcal{CS}$. Hence, irrespective of the network type, it is always guaranteed that even after adding dummy parties to $\mathcal{CS}$, there are at least $t_s + 1$ honest parties in $\mathcal{CS}$, whose secret-shared multiplication-triples are random for the adversary. Consequently, the parties will now be able to extract $c_M$ random secret-shared multiplication-triples. The preprocessing phase protocol is presented in Fig. 4.

The properties of the protocol $\Pi_{\mathsf{PreProcessing}}$ stated in Lemma 6 follows from [3].

**Lemma 6.** *Let $t_a < t_s$ and $t_a+2t_s < n$. Then, the protocol $\Pi_{\mathsf{PreProcessing}}$ achieves the following properties. In a synchronous network, by time $T_{\mathsf{TripGen}} = T_{\mathsf{TripSh}} +$*

[4] Recall that for $\Pi_{\mathsf{TripExt}}$, it is *necessary* that $d \geq t_s$ holds.

[5] Note that in [3], the condition $t_a+3t_s < n$ holds and consequently, $n-t_s \geq 2t_s+t_a+1$ holds; hence the triple-extraction on the multiplication-triples shared by the parties in $\mathcal{CS}$ is *guaranteed* to result in random secret-shared multiplication-triples in their case.

**Protocol** $\Pi_{\mathsf{PreProcessing}}$

- **Phase I - Sharing Random Multiplication-Triples**: Each $P_i \in \mathcal{P}$ does the following:
  - Act as a dealer $\mathsf{D}$ and invoke an instance $\Pi^{(i)}_{\mathsf{TripSh}}$ of $\Pi_{\mathsf{TripSh}}$, so that $c_M$ multiplication-triples are shared on its behalf.
  - For $j = 1, \ldots, n$, participate in the instance $\Pi^{(j)}_{\mathsf{TripSh}}$ invoked by $P_j$ and wait for time $T_{\mathsf{TripSh}}$.
  - Initialize a set $\mathcal{C}_i = \emptyset$ and include $P_j$ in $\mathcal{C}_i$, if an output is computed during the instance $\Pi^{(j)}_{\mathsf{TripSh}}$.
- **Phase II - Agreement on a Common Subset of Triple-Providers**: Each $P_i \in \mathcal{P}$ does the following:
  - For $j = 1, \ldots, n$, participate in an instance $\Pi^{(j)}_{\mathsf{BA}}$ of $\Pi_{\mathsf{BA}}$ with input 1, if $P_j \in \mathcal{C}_i$.
  - Once $n - t_s$ instances of $\Pi_{\mathsf{BA}}$ have produced an output 1, then participate with input 0 in all the $\Pi_{\mathsf{BA}}$ instances $\Pi^{(j)}_{\mathsf{BA}}$, such that $P_j \notin \mathcal{C}_i$.
  - Once an output is computed in all the $n$ instances of $\Pi_{\mathsf{BA}}$, set $\mathcal{CS}$ to be the set of parties $P_j$, such that 1 is obtained as the output in the instance of $\Pi^{(j)}_{\mathsf{BA}}$.
  - If $|\mathcal{CS}| \geq 2t_s + 1$, then select the first $2t_s + 1$ least-indexed parties in $\mathcal{CS}$ and discard the rest. Else if $|\mathcal{CS}| < 2t_s + 1$, then include sufficiently many parties $P_j$ to $\mathcal{CS}$ (least indices parties not in $\mathcal{CS}$), such that $|\mathcal{CS}| = 2t_s + 1$, and assume a default-sharing of $c_M$ number of $[\cdot]$-shared triples $([0], [0], [0])$ on the behalf of $P_j$.
- **Phase III - Extracting Random Multiplication-Triples**: The parties do the following:
  - For every $P_j \in \mathcal{CS}$, let $\{([x^{(j,l)}], [y^{(j,l)}], [z^{(j,l)}])\}_{l=1,\ldots,c_M}$ be the $[\cdot]$-shared multiplication-triples, shared on $P_j$'s behalf.
  - For $l = 1, \ldots, c_M$, the parties participate in the instance $\Pi_{\mathsf{TripExt}}(\mathcal{CS}, \{([x^{(j,l)}], [y^{(j,l)}], [z^{(j,l)}])\}_{P_j \in \mathcal{CS}})$ of $\Pi_{\mathsf{TripExt}}$ and compute the output $([\mathsf{a}^{(l)}], [\mathsf{b}^{(l)}], [\mathsf{c}^{(l)}])$.
  - Output the shared triples $\{([\mathsf{a}^{(l)}], [\mathsf{b}^{(l)}], [\mathsf{c}^{(l)}])\}_{l=1,\ldots,c_M}$.

**Fig. 4.** Network-agnostic preprocessing phase protocol

*$2T_{\mathsf{BA}} + \Delta$, the honest parties output a $[\cdot]$-sharing of $c_M$ multiplication-triples, while in an asynchronous network, the honest parties eventually output a $[\cdot]$-sharing of $c_M$ multiplication-triples. Irrespective of the network type, the view of the adversary remains independent of the output multiplication-triples. The protocol incurs a communication of $\mathcal{O}(c_M \cdot n^6 \cdot \kappa)$ bits from the honest parties and invokes $\mathcal{O}(n)$ instances of $\Pi_{\mathsf{BA}}$.*

## 7 The Network-Agnostic Circuit-Evaluation Protocol

The network-agnostic circuit evaluation protocol $\Pi_{\mathsf{CirEval}}$ (Fig. 5) is standard and is similar to the circuit-evaluation protocol of [3], except that we now use cryptographically-secure building blocks. The protocol consists of *four* phases. In the first phase, the parties generate $c_M$ random $[\cdot]$-shared multiplication-triples using an instance of the $\Pi_{\mathsf{PreProcessing}}$ protocol. Simultaneously, the parties execute an instance of the $\Pi_{\mathsf{ACS}}$ protocol to generate $[\cdot]$-sharing of their respective inputs for the function $f$. The instance of $\Pi_{\mathsf{ACS}}$ will output a common subset $\mathsf{CS}$

**Protocol** $\Pi_{\mathsf{CirEval}}$

- **Preprocessing and Input-Sharing**: The parties do the following:
  - Each $P_i \in \mathcal{P}$ on having the input $x^{(i)}$ for $f$, participates in an instance of $\Pi_{\mathsf{ACS}}$ with input $x^{(i)}$. Let $\mathsf{CS}$ be the common subset of parties, computed as an output during the instance of $\Pi_{\mathsf{ACS}}$, where $|\mathsf{CS}| \geq n - t_s$. Corresponding to every $P_j \notin \mathsf{CS}$, set $x^{(j)} = 0$ and set $[x^{(j)}]$ to a default $[\cdot]$-sharing of 0.
  - In parallel, participate in an instance of $\Pi_{\mathsf{PreProcessing}}$. Let $\{([\mathsf{a}^{(j)}], [\mathsf{b}^{(j)}], [\mathsf{c}^{(j)}])\}_{j=1,\ldots,c_M}$ be the $[\cdot]$-shared multiplication-triples, computed as an output during the instance of $\Pi_{\mathsf{PreProcessing}}$.
- **Circuit Evaluation**:
  - Let $G_1, ..., G_m$ be a publicly-known topological ordering of the gates of $\mathsf{cir}$. For $k = 1, ..., m$, the parties do the following for gate $G_k$:
    - *If $G_k$ is an addition gate:* the parties locally compute $[w] = [u] + [v]$, where $u$ and $v$ are gate-inputs and $w$ is the gate-output.
    - *If $G_k$ is a multiplication-with-a-constant gate with constant $c$:* the parties locally compute $[v] = c \cdot [u]$, where $u$ is the gate-input and $v$ is the gate-output.
    - *If $G_k$ is an addition-with-a-constant gate with constant $c$:* the parties locally compute $[v] = c + [u]$, where $u$ is the gate-input and $v$ is the gate-output.
    - *If $G_k$ is a multiplication gate:* let $G_k$ be the $l^{th}$ multiplication gate in $\mathsf{cir}$ where $l \in \{1, ..., c_M\}$ and let $([\mathsf{a}^{(l)}], [\mathsf{b}^{(l)}], [\mathsf{c}^{(l)}])$ be the $l^{th}$ $[\cdot]$-shared multiplication-triple, generated from $\Pi_{\mathsf{PreProcessing}}$. Moreover, let $[u]$ and $[v]$ be the shared gate-inputs of $G_k$. Then the parties participate in an instance $\Pi_{\mathsf{Beaver}}(([u], [v]), ([\mathsf{a}^{(l)}], [\mathsf{b}^{(l)}], [\mathsf{c}^{(l)}]))$ of $\Pi_{\mathsf{Beaver}}$ and compute the output $[w]$.
- **Output Computation**:
  - Let $[y]$ be the $[\cdot]$-shared circuit-output. The parties participate in an instance $\Pi_{\mathsf{RecPub}}([y])$ of $\Pi_{\mathsf{RecPub}}$ to reconstruct $y$.
- **Termination**: Each $P_i \in \mathcal{P}$ concurrently executes the following steps during the protocol:
  - Upon computing circuit output $y$, send the message $(\mathsf{ready}, y)$ to all the parties in $\mathcal{P}$.
  - Upon receiving the message $(\mathsf{ready}, y)$ from at least $t_s + 1$ parties, send the message $(\mathsf{ready}, y)$ to all the parties in $\mathcal{P}$, provided no $(\mathsf{ready}, \star)$ message has been sent yet.
  - Upon receiving the message $(\mathsf{ready}, y)$ from at least $n - t_s$ parties, output $y$ and terminate.

**Fig. 5.** Network-agnostic circuit-evaluation protocol

of at least $n - t_s$ parties, whose inputs are $[\cdot]$-shared. For the parties outside of $\mathsf{CS}$, a default $[\cdot]$-sharing of 0 is considered as their input. Note that the properties of $\Pi_{\mathsf{ACS}}$ would guarantee that all honest parties are included in $\mathsf{CS}$, ensuring the consideration of inputs from *all honest* parties (namely *input provision*). The second phase involves joint secret-shared evaluation of each gate in the circuit $\mathsf{cir}$, with the resulting output being publicly reconstructed during the *third* phase. Note that once an honest party reconstructs the circuit output, it *cannot* afford to immediately terminate the protocol if the network is *asynchronous* since its participation might be required in various subprotocols to generate output for the other honest parties. Consequently, the last phase is the termination phase, whereupon reconstructing the circuit output, the parties circulate it and check whether it is "safe" to terminate the protocol. The steps for this phase are sim-

ilar to the Bracha's Acast protocol [16]. Once it is confirmed that it is safe to terminate the protocol, the parties terminate the protocol and all underlying subprotocols.

The properties of the protocol $\Pi_{\mathsf{CirEval}}$ stated in Theorem 2 follows from [2,3].

**Theorem 2.** *Let $t_a < t_s$, such that $t_a + 2t_s < n$. Moreover, let $f : \mathbb{F}_p^n \rightarrow \mathbb{F}_p$ be a publicly-known function represented by an arithmetic circuit* cir *over $\mathbb{F}_p$ consisting of $c_M$ number of multiplication gates, and whose multiplicative depth is $D_M$. Moreover, let party $P_i$ has input $x^{(i)}$ for $f$. Then, $\Pi_{\mathsf{CirEval}}$ achieves the following.*

- *In a synchronous network, all honest parties output $y = f(x^{(1)}, ..., x^{(n)})$ at time $6T_{\mathsf{BA}} + (12n + 56 + D_M)\Delta$, where $x^{(j)} = 0$ for every $P_j \notin \mathsf{CS}$, such that $|\mathsf{CS}| \geq n - t_s$ and every honest party $P_j \in \mathcal{P}$ is present in $\mathsf{CS}$.*
- *In an asynchronous network, the honest parties eventually output $y = f(x^{(1)}, ..., x^{(n)})$, where $x^{(j)} = 0$ for every $P_j \notin \mathsf{CS}$, such that $|\mathsf{CS}| \geq n - t_s$.*
- *Irrespective of the network type, the view of the adversary will be independent of the inputs of the honest parties in $\mathsf{CS}$.*
- *The protocol incurs a communication of $\mathcal{O}(c_M \cdot n^7 \cdot \kappa)$ bits from the honest parties and invokes $\mathcal{O}(n)$ instances of $\Pi_{\mathsf{BA}}$.*

## 8 Conclusion and Open Problems

In this paper, we presented a network-agnostic MPC protocol with *optimal* threshold conditions within the plain PKI model. Our protocol is designed by introducing a network-agnostic VSS protocol, resulting in a computationally simpler MPC protocol compared to existing protocols relying on zero-knowledge proofs, threshold homomorphic encryption, and other setups. There are several interesting research directions to pursue in this domain. We outline a few of them below.

- The communication complexity of our MPC protocol does not currently match that of the state-of-the-art network-agnostic protocol of [5]. It would be interesting to develop MPC protocols based on VSS that achieve the same level of communication complexity as the state-of-the-art.
- In this work we have considered the plain PKI model. It will be interesting to apply the methodologies presented in this paper to develop a network-agnostic MPC protocol that incorporates trusted setups.
- Obtaining a packed version of our network-agnostic VSS protocol would be of great interest, as it would subsequently contribute to reducing the communication complexity of the MPC protocol.

**Acknowledgement.** We would sincerely like to thank the anonymous reviewers whose comments and feedback helped to tremendously improve the overall paper. Additionally, we would like to thank PKC 2024, ACM IARCS, and Silence Laboratories for providing conference and travel support.

## References


1. Alexandru, A.B., Blum, E., Katz, J., Loss, J.: State machine replication under changing network conditions. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022. LNCS, vol. 13791, pp. 681–710. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22963-3_23
2. Appan, A., Choudhury, A.: Network agnostic MPC with statistical security. In: Rothblum, G., Wee, H. (eds.) TCC 2023. LNCS, vol. 14370, pp. 63–93. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-48618-0_3
3. Appan, A., Chandramouli, A., Choudhury, A.: Perfectly-secure synchronous MPC with asynchronous fallback guarantees. In: Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing. PODC'22, pp. 92–102. Association for Computing Machinery (2022). https://doi.org/10.1145/3519270.3538417
4. Asharov, G., Lindell, Y.: A full proof of the BGW protocol for perfectly secure multiparty computation. J. Cryptol. **30**(1), 58–151 (2017).https://doi.org/10.1007/s00145-015-9214-4
5. Bacho, R., Collins, D., Liu-Zhang, C.D., Loss, J.: Network-agnostic security comes (almost) for free in DKG and MPC. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023. LNCS, vol. 14081, pp. 71–106. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38557-5_3
6. Bangalore, L., Choudhury, A., Garimella, G.: Round efficient computationally secure multi-party computation revisited. In: Proceedings of the 20th International Conference on Distributed Computing and Networking. ICDCN '19, pp. 292–301. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3288599.3288600
7. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_34
8. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols. In: Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing. STOC '90, pp. 503–513. Association for Computing Machinery (1990). https://doi.org/10.1145/100216.100287
9. Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous secure computation. In: STOC, pp. 52–61. ACM (1993). https://doi.org/10.1145/167088.167109
10. Ben-Or, M., Kelmer, B., Rabin, T.: Asynchronous secure computations with optimal resilience (extended abstract). In: PODC, pp. 183–192. ACM (1994). https://doi.org/10.1145/197917.198088
11. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract), pp. 1–10 (1988). https://doi.org/10.1145/62212.62213
12. Bhimrajka, N., Choudhury, A., Varadarajan, S.: Network-agnostic perfectly secure synchronous message transmission revisited. In: INDOCRYPT (2023). https://doi.org/10.1007/978-3-031-56235-8_2
13. Blum, E., Katz, J., Loss, J.: Tardigrade: an atomic broadcast protocol for arbitrary network conditions. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13091, pp. 547–572. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92075-3_19
14. Blum, E., Katz, J., Loss, J.: Synchronous consensus with optimal asynchronous fallback guarantees. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11891, pp. 131–150. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_6

15. Blum, E., Liu-Zhang, C.D., Loss, J.: Always have a backup plan: fully secure synchronous MPC with asynchronous fallback. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12171, pp. 707–731. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56880-1_25
16. Bracha, G.: An asynchronous [(n - 1)/3]-resilient consensus protocol. In: Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing. PODC '84, pp. 154–162. Association for Computing Machinery (1984). https://doi.org/10.1145/800222.806743
17. Cachin, C., Kursawe, K., Lysyanskaya, A., Strobl, R.: Asynchronous verifiable secret sharing and proactive cryptosystems. In: CCS, pp. 88–97. ACM (2002). https://doi.org/10.1145/586110.586124
18. Canetti, R., Rabin, T.: Fast asynchronous byzantine agreement with optimal resilience. In: STOC, pp. 42–51 (1993). https://doi.org/10.1145/167088.167105
19. Chor, B., Goldwasser, S., Micali, S., Awerbuch, B.: Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In: FOCS, pp. 383–395. IEEE Computer Society (1985). https://doi.org/10.1109/SFCS.1985.64
20. Choudhury, A., Patra, A.: Optimally resilient asynchronous MPC with linear communication complexity. In: ICDCN. ACM (2015). https://doi.org/10.1145/2684464.2684470
21. Choudhury, A., Patra, A.: An efficient framework for unconditionally secure multiparty computation. IEEE Trans. Inf. Theory **63**(1), 428–468 (2017). https://doi.org/10.1109/TIT.2016.2614685
22. Coretti, S., Garay, J., Hirt, M., Zikas, V.: Constant-round asynchronous multiparty computation based on one-way functions. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 998–1021. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_33
23. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–300. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_18
24. Cramer, R., Damgård, I.: Multiparty Computation, an Introduction, pp. 41–87. Birkhäuser Basel (2006). https://doi.org/10.1007/3-7643-7394-6_2
25. Damgård, I., Ishai, Y.: Scalable secure multiparty computation. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 501–520. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_30
26. Deligios, G., Liu-Zhang, C.: Synchronous perfectly secure message transmission with optimal asynchronous fallback guarantees. In: Baldimtsi, F., Cachin, C. (eds.) FC 2023, Part I. Lecture Notes in Computer Science, vol. 13950, pp. 77–93. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-47754-6_5
27. Deligios, G., Hirt, M., Liu-Zhang, C.-D.: Round-efficient byzantine agreement and multi-party computation with asynchronous fallback. In: Nissim, K., Waters, B. (eds.) TCC 2021. LNCS, vol. 13042, pp. 623–653. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90459-3_21
28. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983). https://doi.org/10.1137/0212045
29. Feldman, P., Micali, S.: Optimal algorithms for byzantine agreement. In: STOC, pp. 148–161. ACM (1988). https://doi.org/10.1145/62212.62225
30. Ghinea, D., Liu-Zhang, C., Wattenhofer, R.: Optimal synchronous approximate agreement with asynchronous fallback. In: PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, 25–29 July 2022, pp. 70–80. ACM (2022). https://doi.org/10.1145/3519270.3538442

31. Ghinea, D., Liu-Zhang, C., Wattenhofer, R.: Multidimensional approximate agreement with asynchronous fallback. In: Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2023, Orlando, FL, USA, 17–19 June 2023, pp. 141–151. ACM (2023). https://doi.org/10.1145/3558481.3591105
32. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A.V. (ed.) Proceedings of the 19th Annual ACM Symposium on Theory of Computing, New York, New York, USA, pp. 218–229. ACM (1987). https://doi.org/10.1145/28395.28420
33. Hirt, M., Nielsen, J.B., Przydatek, B.: Cryptographic asynchronous multi-party computation with optimal resilience. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 322–340. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_19
34. Hirt, M., Nielsen, J.B., Przydatek, B.: Asynchronous multi-party computation with quadratic communication. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 473–485. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70583-3_39
35. Kate, A., Goldberg, I.: Distributed key generation for the internet. In: 29th IEEE International Conference on Distributed Computing Systems (ICDCS 2009), 22–26 June 2009, Montreal, Québec, Canada, pp. 119–128. IEEE Computer Society (2009). https://doi.org/10.1109/ICDCS.2009.21
36. Momose, A., Ren, L.: Multi-threshold byzantine fault tolerance. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. CCS '21, pp. 1686–1699. Association for Computing Machinery (2021).https://doi.org/10.1145/3460120.3484554
37. Patra, A., Choudhury, A., Pandu Rangan, C.: Efficient asynchronous verifiable secret sharing and multiparty computation. J. Cryptol. **28**(1), 49–109 (2015). https://doi.org/10.1007/s00145-013-9172-7
38. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_9
39. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: Johnson, D.S. (ed.) Proceedings of the 21st Annual ACM Symposium on Theory of Computing, 14–17 May 1989, Seattle, Washington, USA, pp. 73–85. ACM (1989). https://doi.org/10.1145/73007.73014
40. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979). https://doi.org/10.1145/359168.359176
41. Yao, A.C.: Protocols for secure computations (extended abstract). In: FOCS, pp. 160–164. IEEE Computer Society (1982). https://doi.org/10.1109/SFCS.1982.38