Noname manuscript No.
(will be inserted by the editor)

Efficient Asynchronous Byzantine Agreement with Optimal Resilience

Arpita Patra · Ashish Choudhury · C. Pandu Rangan

Received: date / Accepted: date

Abstract We present a simple, efficient and *optimally resilient Asynchronous Byzantine Agreement* (ABA) protocol involving $n = 3t + 1$ parties over a completely asynchronous network, tolerating a *computationally unbounded Byzantine* adversary, who can control t parties. The *amortized* communication complexity of our ABA protocol is $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits for attaining agreement on a *single* bit. Here ϵ (where $\epsilon > 0$) denotes the probability of non-termination. We compare our protocol with most recent optimally resilient, ABA protocols proposed in [15] and [1] and show that our protocol gains by a factor of $\mathcal{O}(n^7 (\log \frac{1}{\epsilon})^3)$ over the ABA of [15] and by a factor of $\mathcal{O}(n^4 \frac{\log n}{\log \frac{1}{\epsilon}})$ over the ABA of [1].

To design our protocol, we first present a novel, simple and *optimally resilient* statistical *asynchronous verifiable secret sharing* (AVSS) protocol with $n = 3t + 1$, which significantly improves the communication complexity of the only known optimally resilient statistical AVSS protocol of [15]. Our AVSS shares multiple secrets *concurrently* and is far better than multiple parallel executions of AVSS sharing single secret. We believe that our AVSS can be used in many other applications for improving communication complexity and hence is of independent interest.

The common coin primitive is one of the most important building blocks for the construction of ABA protocol. The only known efficient common coin protocol [28, 14] uses multiple executions of AVSS sharing a single secret as a black-box. Unfortunately, this common coin protocol does not achieve its goal when multiple invocations of AVSS sharing single secret are replaced by single invocation of AVSS sharing multiple secrets. Hence in this paper, we extend the existing common coin protocol to make it compatible with our new AVSS. As a byproduct, our new common coin protocol is much more communication efficient than the existing common coin protocol.

A preliminary version of this paper appeared in PODC 2009. The work was done when the first two authors were PhD students at Department of Computer Science and Engineering, IIT Madras

Arpita Patra
Dept. of Computer Science
Aarhus University, Denmark
E-mail: arpitapatra_10@yahoo.co.in, arpitapatra10@gmail.com, arpita@cs.au.dk

Ashish Choudhury
Applied Statistics Unit
Indian Statistical Institute Kolkata
E-mail: partho_31@yahoo.co.in, partho31@gmail.com

C. Pandu Rangan
Dept. of Computer Science and Engineering
IIT Madras, Chennai India 600036
Tel.: +91-44-22574358
E-mail: prangan55@yahoo.com, prangan55@gmail.com

1 Introduction

The problem of Byzantine Agreement (BA) was introduced in [48] and since then it has emerged as the most fundamental problem in distributed computing [44]. Roughly speaking, the BA problem is as follows: there are n parties, each having an input binary value; the goal is for all honest parties to agree on a consensus value. The challenge lies in reaching agreement despite the presence of faulty parties, who may deviate from the protocol arbitrarily. The BA problem has been investigated extensively in various models [29, 4, 10, 15, 14, 44, 32, 41, 2, 7–9, 11–13, 16, 21, 20, 22–25, 34, 30, 31, 26, 28, 36–38, 42, 43, 49, 50, 56, 54, 55]. An interesting variant of BA is *Asynchronous* BA (ABA) tolerating a *computationally* *unbounded* malicious adversary. This problem has got relatively less attention in comparison to the BA problem in synchronous network. In this paper, we make significant inroad towards this direction by designing simple and communication efficient ABA protocol.

1.1 The Model and Definition

We follow the network model of [15,14]. Specifically, there is a set of n parties, say $\mathcal{P} = \{P_1, \ldots, P_n\}$, where every two parties are directly connected by a secure and authentic channel and t out of the n parties can be under the influence of a *computationally unbounded Byzantine (active) adversary*, denoted as $\mathcal{A}_t$. The adversary $\mathcal{A}_t$, completely dictates the parties under its control and can force them to deviate from the protocol in any arbitrary manner. The parties not under the influence of $\mathcal{A}_t$ are called *honest or uncorrupted*.

The underlying network is asynchronous, where the communication channels between the parties have arbitrary, yet finite delay (i.e the messages are guaranteed to reach eventually). To model the worst case scenario, $\mathcal{A}_t$ is given the power to schedule the delivery of *all* messages in the network. However, $\mathcal{A}_t$ can *only schedule* the messages communicated between honest parties, without having any access to them. In asynchronous network, the inherent difficulty in designing a protocol comes from the fact that when *cannot distinguish between a slow sender and a corrupted sender*. Due to this the protocols in asynchronous network are generally involved in nature and require new set of primitives. We now formally define ABA.

Definition 1 (ABA [15]) : Let Π be an asynchronous protocol executed among the set of parties $\mathcal{P}$, with each party having a private binary input. We say that Π is an ABA protocol tolerating $\mathcal{A}_t$ if the following hold:

1. **Termination**: If all honest parties participate in the protocol then all honest parties eventually terminate the protocol.
2. **Agreement**: All honest parties who have terminated the protocol hold identical outputs.
3. **Validity**: If all honest parties had same input ρ, then all honest parties upon termination output ρ.

We now define (ϵ, δ)-ABA protocol for a given ϵ and δ, where $\epsilon, \delta > 0$.

Definition 2 ((ϵ, δ)-ABA) : An ABA protocol Π is called (ϵ, δ)-ABA if Π satisfies **Termination** property except with an error probability of ϵ and **Agreement/ Validity** property except with error probability of δ.

The important parameters of any ABA protocol are:

1. **Resilience**: maximum number of corrupted parties (t) that the protocol can tolerate;
2. **Communication Complexity**: total number of bits communicated by *honest* parties;
3. **Computational Complexity:** computational resources required by the honest parties. An ABA protocol is called computationally efficient if the computational resources required by each honest party are polynomial in n, $\log \frac{1}{\epsilon}$ and $\log \frac{1}{\delta}$; and
4. **Running Time**: We present an informal definition of the running time of an asynchronous protocol, taken from [15,14] (for more details, see [44]): Consider a virtual 'global clock' measuring time in the network. Note that the parties cannot read this clock. Let the *delay* of a message be the time elapsed from its sending to its receipt. Let the *period* of a finite execution of a protocol be the longest delay of a message in the execution. The *duration* of a finite execution is the total time measured by the global clock divided by the period of the execution.
   The *expected running time* of a protocol, *conditioned on an event*, is the maximum over all inputs and applicable adversaries, of the average over the random inputs of the parties, of the duration of executions of the protocol in which this event occurs.

1.2 Existing Results for ABA

From [48], BA (and hence ABA) tolerating $\mathcal{A}_t$ is possible *if and only if* $n \geq 3t+1$. Thus, any ABA protocol designed with $n = 3t+1$ is called as *optimally resilient*. By the seminal result of [31], any ABA protocol, irrespective of the value of n, must have some *non-terminating* runs, where some honest party(ies) may not output any value and thus may not terminate at all. So in any (ϵ, δ)-ABA protocol with non-zero ϵ, the probability of the occurrence of a non-terminating execution is at most ϵ (these type of protocols are called $(1-\epsilon)$-terminating [15,14]). On the other hand in any $(0, \delta)$-ABA protocol, the *probability* of occurrence of a non-terminating execution is *asymptotically zero* (these type of protocols are called *almost-surely terminating* [1]). In Table 1, we summarize the best known ABA protocols.

1.3 Common Technique Used in ABA Protocols

Over a period of time, the techniques and the design approaches of ABA has evolved spectacularly. Rabin [50] designed an ABA protocol *assuming* that the parties have access to a 'common coin protocol', which allows the honest parties to output a common random

Table 1 Summary of Best Known Existing ABA Protocols. In the table, $poly(x)$ stands for polynomial in x

Ref.	Type	Resilience	Communication Complexity (CC)	Expected Running Time (ERT)
[10]	$(0,0)$	$t < n/3$	$\mathcal{O}(2^n)$	$\mathcal{O}(2^n)$
[27,28]	$(0,0)$	$t < n/4$	$poly(n)$	$\mathcal{O}(1)$
[15,14]	$(\epsilon,0)$	$t < n/3$	$poly(n, \frac{1}{\epsilon})$	$\mathcal{O}(1)$
[1]	$(0,0)$	$t < n/3$	$poly(n)$	$\mathcal{O}(n^2)$

Table 2 Comparison of Our Optimally Resilient ABA with Best Known Optimally Resilient ABA Protocols

Ref.	Type	Communication Complexity (CC)	ERT
[15]	$(\epsilon,0)$	Private– $\mathcal{O}(\mathcal{C}n^{11}(\log \frac{1}{\epsilon})^4)$ A-cast– $\mathcal{O}(\mathcal{C}n^{11}(\log \frac{1}{\epsilon})^2 \log n)$	$\mathcal{C} = \mathcal{O}(1)$
[1]	$(0,0)$	Private– $\mathcal{O}(\mathcal{C}n^6 \log n)$ A-cast– $\mathcal{O}(\mathcal{C}n^6 \log n)$	$\mathcal{C} = \mathcal{O}(n^2)$
This Article	$(\epsilon,0)$	Private– $\mathcal{O}(\mathcal{C}n^4(\log \frac{1}{\epsilon}))$ A-cast– $\mathcal{O}(\mathcal{C}n^4(\log \frac{1}{\epsilon}))$	$\mathcal{C} = \mathcal{O}(1)$

bit with some probability (called as the success probability). Bracha [10] proposed a simple implementation of common coin protocol, whose success probability is $\Theta(2^{-n})$. Feldman and Micali [27,28], were the first to come up with a common coin protocol that has constant success probability. The essence of [27] is the reduction of the common coin to that of implementing an *Asynchronous Verifiable Secret Sharing* (AVSS) protocol. Here AVSS is a two phase protocol (Sharing and Reconstruction) carried out among the parties in $\mathcal{P}$ in the presence of $\mathcal{A}_t$. Informally, the goal of the AVSS protocol is to allow a special party in $\mathcal{P}$ called *dealer* to share a secret s among the parties in $\mathcal{P}$ during the sharing phase in a way that would later allow for a unique reconstruction of this secret in the reconstruction phase, while preserving the secrecy of s until the reconstruction phase. Following [27,28], researchers almost followed the same approach of reducing the design of ABA to that of designing AVSS. The same approach is followed by the authors in [15,1] to design their optimally resilient ABA protocols [1]

1.4 Our Motivation and Contribution

In literature, a lot of attention has peen paid to design communication efficient BA protocols in synchronous settings (see [9,16,22,49,35]). Unfortunately, not too much attention has been paid to design communication efficient ABA protocols with optimal resilience. Naturally, designing optimally resilient, communication efficient, fast ABA protocol which runs in constant expected time is an important and interesting problem. Our result in this paper marks a significant progress in this direction.

We present an optimally resilient, $(\epsilon,0)$-ABA protocol. Our ABA protocol requires private communication of $\mathcal{O}(\mathcal{C}n^5 \log \frac{1}{\epsilon})$ bits, as well as A-cast [2] of $\mathcal{O}(\mathcal{C}n^5 \log \frac{1}{\epsilon})$ bits for reaching agreement on $t+1 = \Theta(n)$ bits *concurrently.* So *amortized* communication complexity for agreeing on a *single* bit is $\mathcal{O}(\mathcal{C}n^4 \log \frac{1}{\epsilon})$ bits of private, as well as A-cast communication. Moreover, conditioned on the event that our ABA protocol terminates, it does so in constant expected time; i.e., $\mathcal{C} = \mathcal{O}(1)$. In Table 2, we compare our ABA protocol with the optimally resilient ABA protocols of [15,1]. From the table, we find that our ABA protcol achieves a huge gain in communication complexity over the ABA of [15], while keeping all other properties in place. On the other hand, our ABA enjoys the following merits over the ABA of [1]:

1. Our ABA is better in terms of communication complexity when $(\log \frac{1}{\epsilon}) < n^4 \log n$.
2. Our ABA runs in constant expected time. However, we stress that our ABA is of type $(\epsilon,0)$ whereas ABA of [1] is of type $(0,0)$.

1.5 A Brief Discussion on the Approaches Used in the ABA Protocols of [15,1] and Current Article

We now briefly discuss the approach used in the ABA protocols of [15], [1] and the current article.

1. The ABA protocol of Canetti et.al [15,14] uses the reduction from AVSS to ABA. Hence they have first designed an AVSS with $n = 3t+1$. There are well known inherent difficulties in designing AVSS with $n = 3t+1$ (see [15,14]). To overcome these difficulties, the authors in [15] used the following route to design their AVSS scheme: *ICP* $\rightarrow$ *A-RS* $\rightarrow$ *AWSS* $\rightarrow$ *Two & Sum AWSS* $\rightarrow$ *AVSS*, where $X \rightarrow Y$ means that protocol Y is designed using protocol X as a black-box. Since the final AVSS scheme is designed on the top of so many sub-protocols, it is highly communication intensive as well as very much involved. The protocol privately communicates $\mathcal{O}(n^9(\log \frac{1}{\epsilon})^4)$ bits, A-cast $\mathcal{O}(n^9(\log \frac{1}{\epsilon})^2 \log(n))$ bits during *sharing phase* and privately communicates $\mathcal{O}(n^6(\log \frac{1}{\epsilon})^3)$ bits, A-cast $\mathcal{O}(n^6(\log \frac{1}{\epsilon}) \log(n))$ bits

[1] The authors in [1] followed a slightly different approach. For details, see Section 1.5.

[2] A-cast is the parallel notion of broadcast in synchronous world. A-cast allows a party to send a value to all other parties identically.

during *reconstruction phase* [3] for sharing a single secret s, where all the honest parties terminate the protocol with probability at least $1-\epsilon$.

2. The ABA protocol of [1] used the same reduction from AVSS to ABA as in [15], except that the use of AVSS is replaced by a variant of AVSS that the authors called *shunning* (asynchronous) VSS (SVSS), where each party is guaranteed to terminate *almost-surely*. SVSS is a slightly weaker notion of AVSS in the sense that if all the parties behave correctly, then SVSS satisfies all the properties of AVSS without any error. Otherwise it does not satisfy the properties of AVSS but enables some honest party to identify at least one corrupted party, whom the honest party shuns from then onwards. The use of SVSS instead of AVSS in generating common coin causes the ABA of [1] to run for $\mathcal{O}(n^2)$ expected time. The SVSS protocol requires private communication of $\mathcal{O}(n^4 \log(n))$ bits and A-cast of $\mathcal{O}(n^4 \log(n))$ bits.
3. Our ABA protocol also follows the same reduction from AVSS to ABA as in [15]. We first design a communication efficient AVSS protocol with $n = 3t+1$. Instead of following a fairly complex route taken by [15] to design an AVSS scheme, we follow a shorter route: *ICP* $\rightarrow$ *AWSS* $\rightarrow$ *AVSS*. Beside this, we significantly improve each of these building blocks by employing new design approaches. Also each of the building blocks deals with multiple secrets concurrently and thus lead to significant gain in communication complexity. Specifically, our AVSS scheme requires private communication *as well as* A-cast communication of $\mathcal{O}((\ell n^3 + n^4)\log\frac{1}{\epsilon})$ bits to share ℓ secret(s) concurrently, where $\ell \geq 1$. Moreover, it requires A-cast communication of $\mathcal{O}((\ell n^3 + n^4)\log\frac{1}{\epsilon})$ bits to reconstruct the ℓ secret(s).
   As discussed earlier in subsection 1.2, the *common-coin* protocol is a very important building block of ABA protocol. Previously, AVSS sharing single secret was used to design the only known *common-coin* protocol with polynomial communication complexity [28,14]. Informally, in the common coin protocol of [28], each party P_i in $\mathcal{P}$ is asked to act as a dealer and share n random secrets using AVSS. For this P_i invokes n parallel instances of AVSS as a dealer to parallely share n secrets. It is obvious that we can do better if P_i invokes *single* instance of AVSS, which shares n secrets *concurrently*. However, our detailed analysis of the existing common coin protocol shows that the above modification does not lead to a correct solution for common coin protocol. Hence we bring several new modifications to the existing *common-coin* protocol so that it can use our new AVSS (that can share multiple secrets concurrently). As a result, our new common coin protocol is more communication efficient than the existing common coin protocol of [14,15]. Together, this lead to our efficient ABA protocol.

[3] The exact communication complexity analysis of the AVSS (and ABA) scheme of [15] was not done earlier. For the sake of completeness, we carry out the same in **APPENDIX A**.

1.6 Primitives To be Used

We now present the definition of the primitives which are used in this paper. Our ABA protocol has error probability of ϵ in **Termination**, where $\epsilon > 0$ and is called the error parameter. To bound the error probability by ϵ, all our protocols work over a finite field $\mathbb{F}$ where $\mathbb{F} = GF(2^\kappa)$ and $\epsilon = 2^{-\Omega(\kappa)}$, for some non-zero κ. Thus each field element can be represented by $\mathcal{O}(\kappa) = \mathcal{O}(\log\frac{1}{\epsilon})$ bits. Moreover, without loss of generality, we assume $n = poly(\kappa)$. That is, n is polynomial in κ. Thus $n = poly(\log\frac{1}{\epsilon})$.

Definition 3 (**Statistical Asynchronous Weak Secret Sharing (AWSS)** [15]) Let (Sh, Rec) be a pair of protocols in which a dealer $D \in \mathcal{P}$ shares a secret s. We say that (Sh, Rec) is a t-resilient statistical AWSS scheme for n parties if all the following hold for every possible behavior of $\mathcal{A}_t$:

- **Termination**: With probability at least $1-\epsilon$, the following requirements hold:
  1. If D is *honest* and all honest parties participate in the protocol, then each honest party will eventually terminate protocol Sh.
  2. If some honest party has terminated protocol Sh, then irrespective of the behavior of D, each honest party will eventually terminate Sh.
  3. If all honest parties have terminated Sh and invoked Rec, then each honest party will eventually terminate Rec.
- **Correctness**: With probability at least $1-\epsilon$, the following requirements hold:
  1. If D is *honest* then each honest party upon terminating Rec, outputs the shared secret s.
  2. If D is *faulty* and some honest party has terminated Sh, then there exists a *unique* $s' \in \mathbb{F} \cup \{NULL\}$, such that each honest party upon terminating Rec will output *either* s' *or* $NULL$. This property is also called as weak-commitment.
- **Secrecy**: If D is *honest* and no honest party has begun executing protocol Rec, then $\mathcal{A}_t$ has no information about s.

Definition 4 (**Statistical Asynchronous Verifiable Secret Sharing (AVSS)** [15]) The **Termination** and **Secrecy** conditions for AVSS are same as in AWSS. The only difference is in the second **Correctness** property, which is *strengthened* as follows:

- **Correctness 2**: If D is *faulty* and some honest party has terminated Sh, then there exists a *unique* $s' \in \mathbb{F} \cup \{NULL\}$, such that with probability at least $1-\epsilon$, each honest party upon terminating Rec will output *only* s'. This property is also called as strong-commitment.

Remark 1 In the literature, there are stronger definitions of VSS which requires that D's committed secret $s' \in \mathbb{F}$, instead of $\mathbb{F} \cup \{NULL\}$ [40]. Such stronger definition is required if VSS is used for multi-party computation MPC [5]. However, such strong definition is not required if we want to use VSS to design BA protocol. In fact, such a weak definition of VSS is used in [45] to study the round complexity of VSS.

The above definition of AWSS and AVSS can be extended for secret S containing ℓ element(s) from $\mathbb{F}$.

Definition 5 (A-cast [15]) Let Π be an asynchronous protocol initiated by a special party (called the sender), having input m (the message to be broadcast). We say that Π is a t-resilient A-cast protocol if the following hold:

- **Termination**:
  1. If the sender is honest and all the honest parties participate in the protocol, then each honest party will eventually terminate the protocol.
  2. Irrespective of the behavior of the sender, if any honest party terminates the protocol then each honest party will eventually do the same.
- **Correctness**: If the honest parties terminate the protocol then they do so with a common output m^*. Furthermore, if the sender is honest then $m^* = m$.

Bracha [10] gave an elegant implementation of A-cast with $n = 3t+1$. For details, see [14]. The following theorem states the communication complexity of Bracha's A-cast protocol.

Theorem 1 *Bracha's A-cast protocol privately communicates $\mathcal{O}(\ell n^2)$ bits to A-cast an ℓ bit message.*

Notation 1 *In the rest of the paper, we use the following convention: we say that P_j receives m from the A-cast of P_i, if P_j (as a receiver) terminates the execution of P_i's A-cast (as a sender), with m as the output.*

2 Organization of the Paper

For the ease of presentation, we divide the paper into two parts. In the first part, our focus is to describe the main ideas used in our AWSS and AVSS protocols. Hence for ease of understanding, we present our AWSS and AVSS scheme sharing *single* secret. By incorporating this AVSS into the existing common coin protocol [28,14], we devise an ABA scheme which allows the parties to agree on a *single* bit and requires private communication as well as A-cast of $\mathcal{O}(n^6(\log \frac{1}{\epsilon}))$ bits. In fact, this ABA scheme was reported in [47].

In the second part of the paper, we extend our AWSS and AVSS scheme to share *multiple* secrets concurrently. We then show how to modify the common coin protocol of [28,14] and present a new common coin protocol that use our AVSS sharing multiple secrets concurrently. Finally, using this common coin protocol, we present our new ABA scheme whose *amortized* communication cost of reaching agreement on a *single* bit is $\mathcal{O}(n^4(\log \frac{1}{\epsilon}))$ bits of private as well as A-cast communication. We then conclude our article with conclusion and open problems.

3 AVSS Scheme for Sharing a Single Secret

In this section, we first present a new Information Checking Protocol (ICP). Then using ICP, we design an AWSS scheme. Finally, a new AVSS scheme is constructed using our AWSS scheme. So the next three subsections are dedicated to ICP, AWSS and AVSS respectively.

3.1 Information Checking Protocol (ICP)

The Information Checking Protocol (ICP) is a tool for authenticating messages in the presence of computationally unbounded corrupted parties. The notion of ICP was first introduced by Rabin et.al [51]. As described in [51,15,18], an ICP is executed among three parties: a *dealer* $D \in \mathcal{P}$, an *intermediary* $INT \in \mathcal{P}$ and a *verifier* $R \in \mathcal{P}$. The dealer D gives a secret value $s \in \mathcal{F}$ to INT. At a later stage, INT is required to reveal s to R and *convince* R, that s is indeed the value which INT received from D. In order to facilitate INT and R to achieve their goal, D sends some *authentication information* to INT (along with secret s) and at the same time, D sends some *verification information* to R. This can be viewed as if D is giving his *signature* on s to INT, which INT can later reveal to R. In [51], the authors called this signature as *IC Signature*.

The basic definition of ICP involves only a *single* verifier R [51,18,15]. We extend this notion to *multiple* verifiers, where all the n parties in $\mathcal{P}$ act as verifiers simultaneously. This will be later helpful in using ICP as a tool in our AWSS protocol. Moreover, our ICP can deal with *multiple* secrets *concurrently* and thus achieves better communication complexity than multiple execution of ICP dealing with single secret. Our ICP is executed in asynchronous settings and thus we refer it as AICP. We now formally define AICP.

Definition 6 (Asynchronous Information Checking Protocol (AICP)) Let $D \in \mathcal{P}$ and D has a secret $S = (s^1, \ldots, s^\ell)$, containing ℓ element(s) from $\mathbb{F}$. D wants to give S to $INT \in \mathcal{P}$, such that later when INT reveals S, the entire set $\mathcal{P}$ can act as verifiers and verify that S was indeed received by INT from D. Then any AICP protocol to achieve this task has the following three phases:

1. **Generation Phase**: initiated by D, where D privately sends S, along with some *authentication information* to INT and some *verification information* to individual verifiers.
2. **Verification Phase**: initiated by INT where INT interacts with D and the verifiers in $\mathcal{P}$ to ensure that INT possesses an S obtained from D, which will be later accepted by each (honest) verifier in $\mathcal{P}$. The secret S, along with the *authentication information*, which is finally possessed by INT at the end of **Verification Phase** is called as D's *IC signature* on S, denoted by $ICSig(D, INT, \mathcal{P}, S)$.
3. **Revelation Phase**: carried out by INT and the verifiers in $\mathcal{P}$. Here INT reveals $ICSig(D, INT, \mathcal{P}, S)$, that is INT reveals the secret S along with the authentication information. The verifiers then publish their responses after verifying S with respect to their verification information. Depending upon the responses by the verifiers, every individual verifier $P_i \in \mathcal{P}$ either accepts S (indicating that P_i is convinced that S was indeed obtained by INT from D) or rejects it (indicating that P_i is not convinced that S was indeed obtained by INT from D). Upon acceptance (resp., rejection), verifier P_i sets $\mathsf{Reveal}_i = S$ (resp., $\mathsf{Reveal}_i = NULL$).

Any AICP should satisfy the following properties:

1. **AICP-Correctness1:** If D and INT are *honest*, then S will be accepted in **Revelation Phase** by each *honest* verifier.
2. **AICP-Correctness2:** At the end of **Verification Phase**, an *honest* INT will possess an S which will be accepted in **Revelation Phase** by each honest verifier, except with probability ϵ.
3. **AICP-Correctness3:** If D is *honest*, then during **Revelation Phase**, with probability at least $(1-\epsilon)$, every $S' \neq S$ revealed by a *corrupted* INT will not be accepted by an *honest* verifier.
4. **AICP-Secrecy:** If D and INT are *honest* and INT has not started **Revelation Phase**, then $\mathcal{A}_t$ will have no information about S.

We now present an informal idea of our novel AICP called Multi-Verifier-AICP. The protocol operates over field $\mathbb{F} = GF(2^\kappa)$, where $\epsilon = 2^{-\Omega(k)}$.

The Intuition: In Multi-Verifier-AICP, D selects a random polynomial $F(x)$ of degree $\ell+t$, whose first ℓ coefficients are the elements of S and delivers $F(x)$ to INT. In addition, to each verifier P_i, D gives the value of $F(x)$ at a random *evaluation point* α_i. During the revelation phase, INT will A-cast $F(x)$ and each verifier P_i will check if $F(x)$ satisfies α_i. Notice that this distribution helps to achieve **AICP-Correctness3**. Specifically, if D is *honest*, then a *corrupted* INT will not know α_i of an honest P_i and so with very high probability, $F'(x) \neq F(x)$ produced by INT will be caught by honest P_i. The above distribution also maintains **AICP-Secrecy**, as degree of $F(x)$ is $\ell + t$, but only t points on $F(x)$ will be disclosed to $\mathcal{A}_t$. So $\mathcal{A}_t$ will lack ℓ points to *uniquely* interpolate $F(x)$.

But the above distribution alone is not enough to achieve **AICP-Correctness2**. A *corrupted* D might distribute $F(x)$ to INT and value of $F'(x) \neq F(x)$ to each honest verifier. To avoid this situation, INT and the verifiers interact to check the consistency of $F(x)$ held by INT and the values held by verifiers. However, we have to also ensure that secrecy of S is maintained during this consistency checking if INT is *honest*. At the same time, we have to also ensure that a *corrupted* INT should not be able to find α_i's of honest verifiers during the consistency checking. In order to facilitate this checking, D also gives to INT another random polynomial $R(x)$ of degree $\ell + t$ (in addition to $F(x)$). Parallely, to each individual verifier P_i, D gives the value of $R(x)$ at α_i. The specific details of the consistency checking, along with other formal steps of protocol Multi-Verifier-AICP are given in Fig. 1.

Remark 2 We stress that in protocol Multi-Verifier-AICP, $D, INT \in \mathcal{P}$. Hence they also act as verifiers and receive verification information during **Gen**. Moreover, they perform all other steps (in addition to what they are supposed to perform as D and INT) of the protocol as verifiers, which are performed by other verifiers.

We now prove the properties of the protocol.

Claim If D and INT are honest then D will A-cast `OK` and not $F(x)$ during **Ver**.

PROOF: Follows from the fact that if D is honest then $F(\alpha_i) = v_i$ and $R(\alpha_i) = r_i$ for all $P_i \in ReceivedSet$. □

Lemma 1 (AICP-Correctness1) *If D and INT are honest, then S revealed by INT during* **Revelation Phase** *will be accepted by each honest verifier.*

Proof: From previous claim, if D and INT are honest, then D will A-cast OK during **Ver**. Moreover, $v_i = F(\alpha_i)$ and $r_i = R(\alpha_i)$ for each *honest* $P_i \in ReceivedSet$ and there are at least $t+1$ such honest P_i's in $ReceivedSet$. So during **Reveal-Public**, each honest $P_i \in ReceivedSet$ will A-cast Accept, as condition **C1** i.e $v_i = F(\alpha_i)$ will hold for each of them. So each honest P_i will set $\mathsf{Reveal}_i = S$. □

Fig. 1 AICP with $n = 3t + 1$

Protocol Multi-Verifier-AICP$(D, INT, \mathcal{P}, S = (s^1, \ldots, s^\ell), \epsilon)$

Generation Phase: Gen$(D, INT, \mathcal{P}, S, \epsilon)$

1. D picks and sends the following to INT:
   (a) A random degree-$(\ell+t)$ polynomial $F(x) = s^1 + s^2x + \ldots + s^\ell x^{\ell-1} + r_1x^\ell + r_2x^{\ell+1} + \ldots + r_{t+1}x^{\ell+t}$, where r_i's are random elements of $\mathbb{F}$, for $i = 1, \ldots t+1$.
   (b) A random degree-$(\ell+t)$ polynomial $R(x)$ over $\mathbb{F}$.
2. D privately sends the following to every verifier P_i:
   (a) (α_i, v_i, r_i), where $\alpha_i \in \mathbb{F} - \{0\}$ is random and all α_i's are distinct.
   (b) $v_i = F(\alpha_i)$ and $r_i = R(\alpha_i)$.

$F(x)$ here forms **authentication information**, while (α_i, v_i, r_i)'s form **verification information**.

Verification Phase: Ver$(D, INT, \mathcal{P}, S, \epsilon)$

1. For $i = 1, \ldots, n$, verifier P_i sends a Received-From-D signal to INT after receiving (α_i, v_i, r_i) from D.
2. Upon receiving Received-From-D from $2t+1$ verifiers, INT creates a set $ReceivedSet = \{P_i \mid INT$ received Received-From-D signal from $P_i\}$. INT then chooses a random $d \in \mathbb{F} \setminus \{0\}$ and A-casts $(d, B(x), ReceivedSet)$, where $B(x) = dF(x) + R(x)$.
3. D checks $dv_i + r_i \stackrel{?}{=} B(\alpha_i)$ for every $P_i \in ReceivedSet$. If not then he A-casts $F(x)$. Otherwise D A-casts OK.
4. The verifiers and INT do the following:
   (a) If OK is received from the A-cast of D then do nothing.
   (b) If $F(x)$ is received from the A-cast of D, then INT *replaces* the $F(x)$ *privately* received from D during **Gen** with the $F(x)$ now obtained from D's A-cast. Parallely, each verifier P_i re-sets $v_i = F(\alpha_i)$ so that v_i now satisfies the $F(x)$ A-casted by D.

$F(x)$ which is now finally possessed by INT is called D's IC signature on S and we denote this by $ICSig(D, INT, \mathcal{P}, S)$.

Revelation Phase: Reveal-Public$(D, INT, \mathcal{P}, S, \epsilon)$

1. INT A-casts $F(x)$.
2. On receiving $F(x)$ from the A-cast of INT, verifier $P_i \in ReceivedSet$ A-cast Accept in the following conditions.
   (a) $v_i = F(\alpha_i)$ — we call this as condition **C1**; OR
   (b) $B(\alpha_i) \neq dv_i + r_i$ and D A-casted OK during **Ver** — we call this as condition **C2**.
   Otherwise, P_i A-cast Reject.

Local Computation (By Every Verifier in $\mathcal{P}$): If $(t+1)$ verifiers from $ReceivedSet$ have A-casted Accept then accept $F(x)$ and set $\mathsf{Reveal}_i = S$, where S consists of lower order ℓ coefficients of $F(x)$. Else reject $F(x)$ and set $\mathsf{Reveal}_i = NULL$.

Claim Let INT be *honest* and D be *corrupted*. Moreover, during protocol **Gen**, let D has distributed $(F(x), R(x))$ to INT and (α_i, v_i, r_i) to an honest verifier $P_i \in ReceivedSet$ such that $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$. Then except with probability ϵ, $B(\alpha_i) \neq dv_i + r_i$.

Proof: We first argue that there is *only one* non-zero d for which $B(\alpha_i) = dv_i + r_i$ will hold, even though $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$. For otherwise, assume there exists another non-zero $e \neq d$, for which $B(\alpha_i) = ev_i + r_i$ is true, even if $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$. This implies that $(d-e)F(\alpha_i) = (d-e)v_i$ or $F(\alpha_i) = v_i$, which is a contradiction. Now since d is randomly chosen by honest INT *only after* D handed over $(F(x), R(x))$ to INT and (α_i, v_i, r_i) to every *honest* $P_i \in ReceivedSet$, a corrupted D has to guess d in advance during **Gen** to make sure that $B(\alpha_i) = dv_i + r_i$ holds. However, D can guess d with probability at most $\frac{1}{|\mathbb{F}|-1} \approx \epsilon$. □

Lemma 2 (AICP-Correctness2) *At the end of protocol* **Ver**, *$F(x)$ (and hence S) possessed by an honest INT will be accepted in* **Revelation Phase** *by each honest verifier, except with probability ϵ.*

Proof: If D is honest, then lemma follows from Lemma 1. So we consider a *corrupted* D. We claim that in this case, each *honest* $P_i \in ReceivedSet$ will A-cast Accept during **Reveal-Public**, except with probability ϵ. Since there are at least $t+1$ honest verifiers in $ReceivedSet$, it implies that each honest party will accept $F(x)$ and hence S. We have to consider following two cases:

1. *D A-cast $F(x)$ during* **Ver**: In this case, the above claim holds without any error, as honest INT will replace the $F(x)$ which it obtained from D during **Gen**, with the $F(x)$ now A-casted by D. Moreover, each *honest* $P_i \in ReceivedSet$ will *re-set* their v_i, such that $v_i = F(\alpha_i)$. So during **Revelation Phase**, condition **C1**, namely $F(\alpha_i) = v_i$ will hold.
2. *D A-cast OK during* **Ver**: Here, we have the following cases depending on the relation that holds between $(F(x), R(x))$ and (α_i, v_i, r_i)):
   (a) $F(\alpha_i) = v_i$: Here P_i will A-cast Accept without any error as **C1** (i.e $F(\alpha_i) = v_i$) will hold.
   (b) $F(\alpha_i) \neq v_i$ *and* $R(\alpha_i) = r_i$: Here P_i will A-cast Accept without any error probability, as **C2** (i.e $B(\alpha_i) \neq dv_i + r_i$) will hold.
   (c) $F(\alpha_i) \neq v_i$ *and* $R(\alpha_i) \neq r_i$: Here P_i will A-cast Accept except with probability ϵ, as **C2** will hold from the previous claim. □

Lemma 3 (AICP-Correctness3) *If D is honest, then during* **Revelation Phase**, *with probability at least* $(1-\epsilon)$, *every $S' \neq S$ revealed by a corrupted INT will not be accepted by an honest verifier.*

Proof: To reveal $S' \neq S$ during **Reveal-Public**, INT must A-cast $F'(x) \neq F(x)$, such that lower order ℓ coefficients of $F'(x)$ are S'. We now claim that if INT does so, then except with probability ϵ, every honest verifier P_i in $ReceivedSet$ will A-cast `Reject` during **Reveal-Public**. This further implies that S' will be rejected as there are at least $t+1$ honest parties in $ReceivedSet$. We consider the following two cases:

1. *D A-cast $F(x)$ during* **Ver**: In this case, the condition **C2** will be never satisfied during **Reveal-Public**. So the only condition in which an honest $P_i \in ReceivedSet$ will A-cast `Accept` for $F'(x)$ is that $F'(\alpha_i) = v_i = F(\alpha_i)$ holds. But the corrupted INT will have no information about α_i, as D and P_i are honest. Hence the probability that INT can ensure $F'(\alpha_i) = v_i = F(\alpha_i)$ is same as INT correctly guesses α_i, which is at most $\frac{\ell+t}{|\mathbb{F}-1|} \approx 2^{-\Omega(\kappa)} \approx \epsilon$ (since $F(x)$ and $F'(x)$ can have same value at most at $\ell+t$ values of x).
2. *D A-cast `OK` during* **Ver**: In this case, we show that the conditions for which an *honest* verifier P_i in $ReceivedSet$ would A-cast `Accept` for $F'(x)$ are either impossible or may happen with probability ϵ:
   (a) $F'(\alpha_i) = v_i = F(\alpha_i)$: As discussed above, this can happen with probability at most ϵ.
   (b) $B(\alpha_i) \neq dv_i + r_i$ *and D A-casted `OK` during* **Ver**: This case is never possible because if $B(\alpha_i) \neq dv_i+r_i$, then honest D would have A-casted $F(x)$ during **Ver**. □

Lemma 4 (AICP-Secrecy) *If D and INT are honest and INT has not started* **Revelation Phase**, *then $\mathcal{A}_t$ will have no information about S.*

Proof: Follows from the fact that if D and INT are honest then D will A-cast `OK` during **Ver** and $\mathcal{A}_t$ will get at most t points on degree-$(\ell+t)$ polynomial $F(x)$ during **Gen** and **Ver**. □

Theorem 2 *Protocol Multi-Verifier-AICP is an efficient AICP. Protocol* **Gen** *privately communicates $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ bits. Protocol* **Ver** *requires A-cast of $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ and private communication of $\mathcal{O}(n\log n)$ bits.* **Reveal-Public** *A-casts $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ bits.*

Proof: The first part of the theorem follows from Lemma 1, Lemma 2, Lemma 3 and Lemma 4. In protocol **Gen**, D privately gives $\ell+t$ field elements to INT and three field elements to each verifier. Since each field element can be represented by $\mathcal{O}(\kappa) = \mathcal{O}(\log\frac{1}{\epsilon})$ bits, **Gen** incurs a private communication of $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ bits. In protocol **Ver**, every verifier privately sends `Received-From-D` signal to INT, thus incurring a private communication of $\mathcal{O}(n)$ bits. In addition, INT A-casts $B(x)$ containing $\ell+t$ field elements, thus incurring A-cast of $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ bits. In protocol **Reveal-Public**, INT A-casts $F(x)$, consisting of $\ell+t$ field elements, while each verifier A-casts `Accept`/`Reject` signal. So **Reveal-Public** involves A-cast of $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ bits. □

Remark 3 (Note on Finding Communication Complexity) In **Reveal-Public** there are $\Theta(n)$ instances of A-cast: one by INT, while one by each verifier in $ReceivedSet$. However, for communication complexity analysis, we did not focussed on the number of instances of A-cast but rather on the total number of field elements which are A-casted. We will follow this strategy to do the communication complexity analysis of all our protocols and also for the ABA protocol of [15,1]. This will not affect the overall communication complexity analysis.

Notation 2 *We will use following notations while using our protocol Multi-Verifier-AICP in our AWSS scheme. Recall that D and INT can be any party from $\mathcal{P}$. We say that:*

1. *"P_i gives $ICSig(P_i, P_j, \mathcal{P}, S)$ to P_j" to mean that P_i as a dealer executes* **Gen**$(P_i, P_j, \mathcal{P}, S, \epsilon)$*, considering P_j as INT to give his IC signature on S to P_j.*
2. *"P_i receives $ICSig(P_j, P_i, \mathcal{P}, S)$ from P_j" to mean that P_i as INT has completed* **Ver**$(P_j, P_i, \mathcal{P}, S, \epsilon)$ *with the help of the verifiers in $\mathcal{P}$ and finally possess $ICSig(P_j, P_i, \mathcal{P}, S)$, where P_j is the dealer.*
3. *"P_i reveals $ICSig(P_j, P_i, \mathcal{P}, S)$" to means P_i as INT executes* **Reveal-Public**$(P_j, P_i, \mathcal{P}, S, \epsilon)$ *along with the participation of the verifiers in $\mathcal{P}$ to reveal S.*
4. *"P_k completes revelation of $ICSig(P_j, P_i, \mathcal{P}, S)$ with $\text{Reveal}_k = \overline{S}$ (resp. $\text{Reveal}_k = NULL$)" to mean that P_k as a verifier has completed* **Reveal-Public**$(P_j, P_i, \mathcal{P}, S, \epsilon)$ *with $\text{Reveal}_k = \overline{S}$ (resp. $\text{Reveal}_k = NULL$).*

3.2 AWSS Scheme for Sharing a Single Secret

We now present a novel AWSS scheme with $n = 3t+1$, consisting of sub-protocols AWSS-Share and AWSS-Rec. While AWSS-Share allows D to share a secret s, AWSS-Rec enables public reconstruction of either D's shared secret or $NULL$. Moreover, if D is *corrupted*, then s can be either from $\mathbb{F}$ or it can be $NULL$ (in a sense explained in the sequel).

Before beginning the protocol, lets discuss a simple WSS protocol in synchronous settings with $n = 2t+1$. The reason behind discussing the protocol is that our overall AWSS scheme is based on this idea with several other ammendments to deal with the asynchrony of the network. The protocol is as follows:

1. **Sharing Phase**: D takes a random degree-t polynomial $f(x)$, such that $f(0) = s$ and computes the shares $s_i = f(i)$, for $i = 1, \ldots, n$. Then to every

party P_i, D gives $ICSig(D, P_i, \mathcal{P}, s_i)$. The sharing phase terminates, once every P_i as INT, has received $ICSig(D, P_i, \mathcal{P}, s_i)$ from D.

2. **Reconstruction Phase**: Each P_i is asked to reveal $ICSig(D, P_i, \mathcal{P}, s_i)$. Let $WCORE$ be the set of all such P_i's, who are successfully able to reveal the signatures. Now we take the shares of all the parties in $WCORE$ and see whether they lie on a degree-t polynomial. If yes, then the constant term of the polynomial is taken as the secret, otherwise $NULL$ is reconstructed.

It is easy to see that the above protocol satisfies **secrecy** and **correctness** property. For **weak-commitment**, we say that D's committed secret is defined by the shares of the honest parties during sharing phase. Specifically, if the shares of the honest parties lie on a degree-t polynomial, say $f^\star(x)$, then we say that D has committed $s^\star = f^\star(0)$. Otherwise, we say that D has committed $s^\star = NULL$. However, *notice that in this protocol, we cannot ensure that a corrupted D has committed $s^\star \neq NULL$ because we are not checking whether D is giving shares on a degree-t polynomial to honest parties during sharing phase.*

Now if we try to adapt the above protocol in asynchronous settings with $n = 3t+1$, then we have to terminate the sharing phase, as soon as $2t+1$ P_i's, denoted by $WCORE$, have received $ICSig(D, P_i, \mathcal{P}, s_i)$ from D, instead of waiting for *all* $3t+1$ parties to receive IC signatures. We can now say that D's committed secret is defined by the shares of the honest parties in $WCORE$ and there are at least $t+1$ honest parties in $WCORE$. But now we have to terminate the reconstruction phase, as soon as $t+1$ P_i's in $WCORE$ correctly reveals $ICSig(D, P_i, \mathcal{P}, s_i)$, instead of waiting for *all* parties in $WCORE$ to reveal the IC signatures. And since we are terminating with $t+1$ revealed shares, we are bound to get a degree-t polynomial and hence a secret. Now if D is *honest*, then still all the properties will be satisfied because with very high probability, the revealed shares are indeed the correct shares. However, if D is *corrupted*, then in the worst case, there can be t corrupted P_i's in $WCORE$, who can reveal *any* $ICSig(D, P_i, \mathcal{P}, \overline{s_i})$. Moreover, adversary can schedule the messages in such a way that t corrupted parties in $WCORE$ are the first to reveal their shares. Now depending upon which honest party's share from $WCORE$ is correctly revealed next, any degree-t polynomial and hence secret can be reconstructed. This strictly violates the weak commitment property.

The problem with the above adaptation is that we cannot ensure that the shares of *all* honest parties in $WCORE$ will be available in the reconstruction phase. To deal with this problem, we share s using two level of sharing, where each s_i is further committed by P_i using *IC-commitment*, which is defined in the sequel. We now give the high level description of AWSS-Share.

High Level Description of AWSS-Share: First D selects a random, symmetric bivariate polynomial $F(x, y)$ of degree-t in x and y such that $F(0,0) = s$. D then gives $ICSig(D, INT, \mathcal{P}, f_i(j))$ for every $j = 1, \ldots, n$ to P_i. This step implicitly implies that P_i will receive $f_i(x) = F(x, i)$ from D. After receiving these IC signatures from D, every pair of parties (P_i, P_j) *exchange* their own IC signature on their common value, namely $f_i(j) = f_j(i) = F(i, j)$. Then D, in conjunction with all other parties, perform a sequences of communications and computations. As a result of this, at the end of AWSS-Share, every party agrees on a set of $2t+1$ parties, called $WCORE$, such that every party $P_j \in WCORE$ has *IC-committed* $f_j(0)$ using $f_j(x)$ to a set of $2t+1$ parties, called as OKP_j, where *IC-commitment* is defined as follows:

Definition 7 (IC-commitment) In protocol AWSS-Share, we say that P_j has *IC-committed* $f_j(0)$ to the parties in OKP_j, using the degree-t polynomial $f_j(x)$ received from D, if all the following holds for every $P_k \in OKP_j$:

1. P_k has received $ICSig(D, P_k, \mathcal{P}, f_k(j))$ from D;
2. P_k has received $ICSig(P_j, P_k, \mathcal{P}, f_j(k))$ from P_j;
3. $f_k(j) = f_j(k)$.

In some sense, we may view as if every $P_j \in WCORE$ has commited his received (from D) polynomial $f_j(x)$ to the parties in OKP_j (by giving his *IC Signature* on one point of $f_j(x)$ to each party) and the parties in OKP_j allowed him to do so after verifying that they have got D's IC signature on the same value of $f_j(x)$. We will show that later in reconstruction phase, *IC-commitment* $f_j(0)$ of every *honest* $P_j \in WCORE$ will be reconstructed correctly irrespective of whether D is honest or corrupted. Moreover, a corrupted P_j's *IC-commitment* will be reconstructed correctly when D is honest. But on the other hand, any value can be reconstructed as P_j's *IC-commitment*, if *both* D and P_j are corrupted. These properties are at the heart of our AWSS protocol.

Achieving the agreement (among the parties) on $WCORE$ and corresponding OKP_js is a bit tricky in asynchronous network. Even though these sets are constructed on the basis of information that are A-casted, parties may end up with different versions of $WCORE$ and OKP_j's while attempting to generate them locally, due to the asynchronous nature of the network. We solve this problem by asking D first to construct $WCORE$ and OKP_js based on A-casted information and then ask

D to A-cast $WCORE$. After receiving $WCORE$ and OKP_js from the A-cast of D, individual parties ensure the validity of these sets by waiting to receive the same A-cast using which D would have formed these sets. A similar approach was used in the protocols of [1].

Notice that if D is honest, then each honest party will always satisfy all the properties for being in $WCORE$. Hence, if D is honest, then all honest parties will always eventually agree on a $WCORE$ of size $2t+1$ and will terminate AWSS-Share. However, if D is corrupted, then there may exist no $WCORE$ of size $2t+1$, in which case no honest party will terminate AWSS-Share. Protocol AWSS-Share is formally presented in Fig. 2.

Fig. 2 Sharing Phase of AWSS Scheme

Protocol AWSS-Share($D, \mathcal{P}, s, \epsilon$)

DISTRIBUTION: CODE FOR D – Only D executes this code.

1. Select a random, symmetric bivariate polynomial $F(x,y)$ of degree-t in x and y, such that $F(0,0)=s$. For $i=1,\ldots,n$, let $f_i(x)=F(x,i)$.
2. For $i=1,\ldots,n$, give $ICSig(D,P_i,\mathcal{P},f_i(j))$ to P_i for each $j=1,\ldots,n$. For this, D initiates n^2 instances of Gen, each with an error parameter of $\epsilon'=\frac{\epsilon}{n^2}$.

VERIFICATION: CODE FOR P_i – Every party including D executes this code.

1. Wait to receive $ICSig(D,P_i,\mathcal{P},f_i(j))$ for each $j=1,\ldots,n$ from D.
2. Check if $(f_i(1),\ldots,f_i(n))$ defines degree-t polynomial. If yes then give $ICSig(P_i,P_j,\mathcal{P},f_i(j))$ to P_j for $j=1,\ldots,n$.
3. If $ICSig(P_j,P_i,\mathcal{P},f_j(i))$ is received from P_j and if $f_i(j)=f_j(i)$, then A-cast $\texttt{OK}(P_i,P_j)$.

WCORE CONSTRUCTION : CODE FOR D – Only D executes this code.

1. For each P_j, build a set $OKP_j = \{P_k | D$ receives $\texttt{OK}(P_k,P_j)$ from the A-cast of $P_k\}$. When $|OKP_j|=2t+1$, then conclude that P_j's *IC-commitment* on $f_j(0)$ is over and add P_j in $WCORE$ (which was initially empty).
2. Wait until $|WCORE|=2t+1$. Then A-cast $WCORE$ and OKP_j for all $P_j \in WCORE$.

WCORE VERIFICATION & AGREEMENT ON WCORE : CODE FOR P_i — Every party executes this code

1. Wait to receive $WCORE$ and OKP_j for all $P_j \in WCORE$ from D's A-cast, such that $|WCORE|=2t+1$ and $|OKP_j|=2t+1$ for each $P_j \in WCORE$.
2. Wait to receive $\texttt{OK}(P_k,P_j)$ for all $P_k \in OKP_j$ and $P_j \in WCORE$. Only after receiving all these OKs, consider the $WCORE$ and OKP_j's received from D as valid, accept them and terminate AWSS-Share.

Before proceeding further, we now define what we call as D's AWSS-commitment during AWSS-Share.

Definition 8 (D's AWSS-commitment) We say that D has *AWSS-committed* a secret $s \in \mathbb{F}$ during AWSS-Share if there is a unique degree-t univariate polynomial, say $f(x)$, such that $f(0)=s$ and every *honest* P_i in $WCORE$ receives $f(i)$ from D. Otherwise, we say that D has *AWSS-committed* $NULL$.

An honest D always *AWSS-commit* $s \in \mathbb{F}$, as in this case $f(x)=f_0(x)=F(x,0)$. Moreover, every honest party P_i in $WCORE$ receives $f(i)=f_0(i)=f_i(0)$ (this can be obtained from $f_i(x)$). But AWSS-Share can *not* ensure that corrupted D has also *AWSS-commit*ed $s \in \mathbb{F}$. This means that a corrupted D may distribute information to the parties such that, polynomial $f_0(x)$ defined by the $f_0(i)=f_i(0)$ values possessed by honest P_i's in $WCORE$ may not be a degree-t polynomial. In this case we say D has *AWSS-committed* $NULL$.

High Level Idea of AWSS-Rec: In AWSS-Rec, we try to reconstruct D's AWSS-committed secret. For this, we reconstruct *IC-commitment* $f_j(0)=f_0(j)$ of every $P_j \in WCORE$ and check whether the reconstructed $f_j(0)$'s of all $P_j \in WCORE$ lies on a unique degree-t polynomial. If yes, then the constant term of the polynomial is considered as the reconstructed secret, else $NULL$ is taken as the reconstructed secret.

To reconstruct the *IC-commitment* $f_j(0)$ for $P_j \in WCORE$, it is enough to have $t+1$ points on the degree-t polynomial $f_j(x)$, used by P_j during AWSS-Share to do the *IC-commitment*. We ask the parties in OKP_j to reveal these points. Specifically, every party $P_k \in OKP_j$ is asked to reveal $ICSig(D,P_k,\mathcal{P},f_k(j))$ and $ICSig(P_j,P_k,\mathcal{P},f_j(k))$ such that $f_k(j)=f_j(k)$ holds. Every such $f_j(k)=f_k(j)$ which is revealed successfully by $P_k \in OKP_j$ is considered as a *valid* point on $f_j(x)$. Since there are at least $t+1$ honest parties in OKP_j, eventually at least $t+1$ $f_j(k)$'s and $f_k(j)$'s, satisfying $f_j(k)=f_k(j)$ will be revealed correctly with which $f_j(x)$ and thus $f_j(0)$ will be reconstructed. Notice that due to asynchrony of the network, we cannot wait for *every* $P_k \in OKP_k$ to reveal $ICSig(D,P_k,\mathcal{P},f_k(j))$ and $ICSig(P_j,P_k,\mathcal{P},f_j(k))$ and so as soon as $t+1$ P_k's from OKP_j reveal valid points on $f_j(x)$, we have to reconstruct $f_j(x)$ and hence $f_j(0)$.

Asking every $P_k \in OKP_j$ to reveal IC signature of D as well as of P_j on the same value is required to ensure **Correctness 1** and **Correctness 2** property of AWSS. Specifically, we will see that when at least one of D and P_j is honest, then P_j's *IC-commitment* (i.e $f_j(0)$) will be reconstructed correctly. But when both D and P_j are corrupted, P_j's *IC-Commitment* can be reconstructed as any $\overline{f_j(0)}$ which may or not be equal to $f_j(0)$. It is this later property that makes our protocol to qualify as a AWSS protocol rather than a AVSS protocol. Now if we recall the adaptation of the synchronous WSS protocol to asynchronous setting, we can see that how using *IC-commitment*, we can now get back the shares (namely $f_j(0)$) of *all* honest P_j's in $WCORE$, which helps us to achieve weak commitment. Protocol AWSS-Rec is given in Fig. 3.

Fig. 3 Reconstruction Phase of AWSS Scheme

AWSS-Rec$(D, \mathcal{P}, s, \epsilon)$

SIGNATURE REVELATION: CODE FOR P_i — Every party executes this code

1. If P_i belongs to OKP_j for some $P_j \in WCORE$, then reveal $ICSig(D, P_i, \mathcal{P}, f_i(j))$ and $ICSig(P_j, P_i, \mathcal{P}, f_j(i))$.

LOCAL COMPUTATION: CODE FOR P_i — Every party executes this code

1. For every $P_j \in WCORE$, reconstruct P_j's *IC-commitment*, say $\overline{f_j(0)}$ as follows:
   (a) Construct a set $ValidP_j = \emptyset$.
   (b) Add $P_k \in OKP_j$ to $ValidP_j$ if the following conditions hold:
      i. Revelation of $ICSig(D, P_k, \mathcal{P}, f_k(j))$ and $ICSig(P_j, P_k, \mathcal{P}, f_j(k))$ are completed with $\mathsf{Reveal}_i = \overline{f_k(j)}$ and $\mathsf{Reveal}_i = \overline{f_j(k)}$ respectively; and
      ii. $\overline{f_k(j)} = \overline{f_j(k)}$.
   (c) Wait until $|ValidP_j| = t+1$. Construct a degree-t polynomial $\overline{f_j(x)}$ passing through $(k, \overline{f_j(k)})$ where $P_k \in ValidP_j$. Associate $\overline{f_j(0)}$ with $P_j \in WCORE$.
2. Wait for $\overline{f_j(0)}$ to be reconstructed for all $P_j \in WCORE$.
3. Check whether the points $(j, \overline{f_j(0)})$ for $P_j \in WCORE$ lie on a unique degree-t polynomial $\overline{f_0(x)}$. If yes, then output $\overline{s} = \overline{f_0(0)}$ and terminate AWSS-Rec. Else output $\overline{s} = NULL$ and terminate AWSS-Rec.

We now prove the properties of our AWSS scheme.

Lemma 5 (AWSS-Termination) *AWSS-Share, AWSS-Rec satisfy termination property of Definition 3.*

PROOF:

- **Termination 1:** If D is honest and all honest parties participate during AWSS-Share, then eventually all honest parties will A-cast `OK` for each other. So D will eventually include $2t+1$ parties in $WCORE$ and A-cast the same, along with OKP_j's for every $P_j \in WCORE$. Since honest D has included P_j in $WCORE$ after receiving the `OK` signals from the A-cast of the parties in OKP_j's, each honest party will also eventually receive the same `OK`, will consider the $WCORE$ and OKP_j's as valid and will accept them and will eventually terminate AWSS-Share.
- **Termination 2:** If an honest P_i has terminated AWSS-Share, then he must have received $WCORE$ and OKP_j's from the A-cast of D and verified their validity by receiving the `OK` signals from the A-cast of the parties in OKP_j's for every $P_j \in WCORE$. By property of A-cast, each honest party will also eventually receive the same, will consider $WCORE$ and OKP_j's as valid and will terminate AWSS-Share.
- **Termination 3:** For every $P_j \in WCORE$, there are at least $t+1$ honest P_k's in OKP_j, who will be able to successfully reveal $ICSig(D, P_k, f_k(j), \mathcal{P})$ and $ICSig(P_j, P_k, f_j(k), \mathcal{P})$ with $f_j(k) = f_k(j)$ during Reveal-Public, except with error probability ϵ' (as each instance of AICP is executed with an error parameter ϵ'). So each honest $P_k \in OKP_j$ will be present in $ValidP_j$ except with probability ϵ'. Thus except with probability $n^2\epsilon' = \epsilon$, P_j's *IC-commitment* will be reconstructed for all $P_j \in WCORE$ and hence except with probability ϵ, all honest parties will terminate AWSS-Rec. □

Lemma 6 (AWSS-Secrecy) *AWSS-Share satisfies secrecy property of Definition 3.*

PROOF: The proof follows from the secrecy of our AICP protocol and properties of symmetric bivariate polynomial of degree-t in x and y [17]. Specifically, let $P_1, \ldots, P_t$ be under the control of $\mathcal{A}_t$. So during AWSS-Share, $\mathcal{A}_t$ will know $f_1(x), \ldots, f_t(x)$ and t points on $f_{t+1}(x), \ldots, f_n(x)$. However, $\mathcal{A}_t$ still lacks one more point to uniquely interpolate $F(x, y)$ and get $s = F(0, 0)$. □

Lemma 7 (AWSS-Correctness) *AWSS-Share, AWSS-Rec satisfy correctness property of Definition 3.*

PROOF:

- **Correctness 1:** Here we have to consider the case when D is *honest*. We show that D's *AWSS-commitment* will be reconstructed correctly except with probability ϵ. For this, we show that P_j's *IC-commitment* $f_j(0)$ will be correctly reconstructed with probability at least $(1 - \frac{\epsilon}{n})$ for *every* $P_j \in WCORE$. Consequently, as $|WCORE| = 2t+1$, all the honest parties will reconstruct $f_0(x) = F(x, 0)$ and hence $s = f_0(0)$ with probability at least $(1 - (2t+1)\frac{\epsilon}{n}) \approx (1 - \epsilon)$. So we consider the following two cases:
  1. $P_j \in WCORE$ is *honest*: From Lemma 3 (i.e., **AICP-Correctness3**), a corrupted $P_k \in \overline{OKP_j}$ can reveal $ICSig(P_j, P_k, \mathcal{P}, \overline{f_j(k)})$ where $\overline{f_j(k)} \neq f_j(k)$, with probability at most ϵ'. As there can be at most t corrupted parties in $ValidP_j$, except with probability $t\epsilon' = \frac{\epsilon}{n}$, the value $\overline{f_j(k)} = f_j(k)$ for all $P_k \in ValidP_j$. Hence honest P_j's *IC-commitment* $f_j(0)$ will be correctly reconstructed with probability at least $(1 - \frac{\epsilon}{n})$.
  2. $P_j \in WCORE$ is *corrupted*: From Lemma 3 (i.e., **AICP-Correctness3**), a corrupted $P_k \in \overline{OKP_j}$ can reveal $ICSig(D, P_k, \mathcal{P}, \overline{f_k(j)})$ where $\overline{f_k(j)} \neq f_k(j)$, with probability at most ϵ'. Thus

except with probability $t\epsilon' = \frac{\epsilon}{n}$, the value $\overline{f_k(j)} = f_k(j) = f_j(k)$ for all $P_k \in ValidP_j$. So corrupted P_j's *IC-commitment* $f_j(0)$ will be correctly reconstructed with probability at least $(1 - \frac{\epsilon}{n})$.

– **Correctness 2:** Here we have to consider a *corrupted* D. Now there are following two cases:
  1. D's AWSS-commitment $s \in \mathbb{F}$: This implies that the $f_j(0)$ values received by the *honest* parties in $WCORE$ during AWSS-Share lies on a degree-t polynomial $f_0(x)$. Now using similar arguments as in **Correctness 1**, it follows that $f_j(0)$ will be reconstructed correctly with probability at least $(1-(t+1)\epsilon') \approx (1-\frac{\epsilon}{n})$ for every *honest* $P_j \in WCORE$. As there are at least $t+1$ honest parties in $WCORE$, *IC-commitment* of all honest parties in $WCORE$ will be reconstructed correctly with probability at least $(1-\epsilon)$.
  But for a *corrupted* P_j in $WCORE$, P_j's *IC-commitment* can be reconstructed as *any* value $\overline{f_j}(0)$. This is because a corrupted $P_k \in OKP_j$ can reveal $ICSig(D, P_k, \mathcal{P}, \overline{f_k(j)})$, as well as $ICSig(P_j, P_k, \mathcal{P}, \overline{f_j(k)})$, for *any* $\overline{f_k(j)} = \overline{f_j(k)}$ of adversary's choice. Also the adversary can schedule the signature revelation in such a way that signature revelation by corrupted P_k's in OKP_j are completed before the signature revelation by honest P_k's in OKP_j. Now if reconstructed $\overline{f_j(0)} = f_j(0)$ for all corrupted $P_j \in WCORE$, then s will be reconstructed. Otherwise, $NULL$ will be reconstructed. However, since for all the honest P_j's in $WCORE$, *IC-commitment* $f_j(0)$ (which in turn define $f_0(x)$) will be reconstructed correctly with probability at least $(1-\epsilon)$, no other secret (other than s) can be reconstructed.
  2. D's AWSS-commitment is $NULL$: This implies that $f_j(0)$'s corresponding to honest P_j's in $WCORE$ do not define a degree-t polynomial. In this case $NULL$ will be reconstructed. This is because $f_j(0)$ corresponding to each honest $P_j \in WCORE$ will be reconstructed correctly except with probability ϵ (following the argument given in previous case).

□

Lemma 8 (AWSS-Communication Complexity) *Protocol AWSS-Share incurs a private communication of $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits. Protocol AWSS-Rec involves A-cast of $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits.*

PROOF: In AWSS-Share, there are $\mathcal{O}(n^2)$ instances of Gen and Ver (of Multi-Verifier-AICP), each dealing with $\ell = 1$ value and executed with an error parameter of $\epsilon' = \frac{\epsilon}{n^2}$. From Theorem 2, this requires a private communication, as well as A-cast of $\mathcal{O}(n^3 \log \frac{n^2}{\epsilon}) = \mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits, as $n = \text{poly}(\frac{1}{\epsilon})$. Moreover, there are A-cast of $\mathcal{O}(n^2)$ OK signals. In addition, there is A-cast of $WCORE$ containing the identity of $2t+1$ parties and OKP_j's corresponding to each $P_j \in WCORE$, where each OKP_j contains the identity of $2t+1$ parties. Now the identity of a party can be represented by $\mathcal{O}(\log n)$ bits. So in total, AWSS-Share incurs a private communication of $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^2 \log n + n^3 \log \frac{1}{\epsilon}) = \mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits. In AWSS-Rec, there are $\mathcal{O}(n^2)$ instances of Reveal-Public of our Multi-Verifier-AICP, each dealing with $\ell = 1$ value. This requires A-cast of $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits.

Theorem 3 *Protocols (AWSS-Share, AWSS-Rec) constitutes a valid statistical AWSS scheme with $n = 3t+1$.*

PROOF: Follows from Lemma 5, 6 and 7.

Notation 3 (AWSS Sharing of a Polynomial) *If we closely look into the computations of AWSS-Share, then we observe that the shares of AWSS-shared secret s are nothing but the points on degree-t polynomial $f_0(x) = F(x, 0)$, where $f_0(0) = s$. Due to asynchrony of the network, instead of all $3t+1$ parties, only a set of $2t+1$ parties $WCORE$ will hold the shares of s. Similarly, to reconstruct s we try to reconstruct the degree-t polynomial $f_0(x)$ using the shares (IC-commitments) of the parties in $WCORE$. So we now abuse the notion of AWSS-sharing of a secret and say that:*
1. *D executes AWSS-Share($D, \mathcal{P}, f(x), \epsilon$) to mean that D AWSS-shares degree-t polynomial $f(x)$ during AWSS-Share. To do so, D will choose a symmetric bivariate polynomial $F(x, y)$ of degree-t in x and y, where $F(x, 0) = f(x)$ holds and will execute the steps of protocol AWSS-Share.*
2. *Parties execute AWSS-Rec($D, \mathcal{P}, f(x), \epsilon$), which allows the (honest) parties to reconstruct either the AWSS-Shared polynomial $f(x)$ or $NULL$, except with an error probability of ϵ.* □

Remark 4 The above notation of abusing the notion of sharing (reconstructing) a secret to sharing (reconstructing) a degree-t polynomial $f(x)$ is very well known and commonly used in WSS protocols in synchronous settings [45, 33, 40]. This does not break the interface when WSS is further used as a black-box in VSS because internally, to share a degree-t polynomial $f(x)$, D has to follow the same steps as in the WSS protocol, with the condition that now the selected bivariate polynomial $F(x, y)$ should satisfy $F(x, 0) = f(x)$. □

3.3 Our AVSS Scheme for Sharing a Single Secret

In this section, we present our novel AVSS scheme consisting of sub-protocols AVSS-Share and AVSS-Rec. Before presenting the protocol, lets recall why the protocol in the previous section fails to qualify as an AVSS scheme. In the previous protocol, if D is *corrupted* and $P_i \in WCORE$ is also *corrupted*, then *any* value can be reconstructed as P_i's IC-commitment. This is because during reconstruction phase, *any* degree-t polynomial can be reconstructed on behalf of $P_i \in WCORE$. If we can ensure that this reconstructed degree-t polynomial is either the same as received by P_i from D during the sharing phase or $NULL$, then we can achieve strong commitment. We now see how we achieve this property by using AWSS as a black-box.

High Level Idea of AVSS-Share: D selects a symmetric bivariate polynomial $F(x, y)$ of degree-t in x and y, such that $F(0, 0) = s$ and sends $f_i(x) = F(x, i)$ to party P_i. Now each party P_i is asked to act as a dealer and WSS-share his received polynomial $f_i(x)$. Then the parties agree on a set of $2t + 1$ parties, say $VCORE$, such that each $P_i \in VCORE$ has WSS-shared $f_i(x)$. However, we have to ensure that even a *corrupted* $P_i \in VCORE$ has indeed AWSS-shared $f_i(x)$. This is done as follows: during the instance of WSS initiated by P_i, the party P_i selects a degree-t symmetric bivariate polynomial $Q^{P_i}(x, y)$, such that $Q^{P_i}(x, 0) = f_i(x)$. Since every party P_j receives $q_j^{P_i}(x) = Q^{P_i}(x, j)$ from P_i as part of AWSS-Share, P_j can check whether $q_j^{P_i}(0) \stackrel{?}{=} f_j(i)$, as ideally $q_j^{P_i}(0) = f_i(j) = f_j(i)$ should hold in case of honest D, P_i and P_j. A party P_j participates in the remaining steps of the instance of AWSS-Share where P_i is the dealer, only if $q_j^{P_i}(0) = f_j(i)$ holds. Moreover, we also ensure that each party $P_j \in VCORE$ has AWSS-shared $f_j(x)$ to at least $2t + 1$ parties in $VCORE$. The agreement on $VCORE$ and $WCORE$ sets corresponding to each $P_j \in VCORE$ is achieved using a mechanism, similar to the one used in AWSS-Share for achieving agreement on $WCORE$ and corresponding OK sets. Protocol AVSS-Share is given in Fig. 4. Before proceeding further, we define what we call as D's commitment during AVSS-Share.

Definition 9 **(D's AVSS-commitment)** We say that D has AVSS-committed $s \in \mathbb{F}$ in AVSS-Share if there is a unique symmetric bivariate polynomial $F(x, y)$ of degree-t in x and y, such that $F(0, 0) = s$ and every *honest* P_i in $VCORE$ receives $f_i(x) = F(x, i)$ from D. Otherwise, we say that D has committed $NULL$ and D's AVSS-committed secret is not *meaningful*.

If a *corrupted* D has committed $NULL$, then it implies that the $f_i(x)$'s of honest parties in $VCORE$ do not define a symmetric bivariate polynomial of degree-t in x and y. This further implies that there is an honest pair (P_γ, P_δ) in $VCORE$ such that $f_\gamma(\delta) \neq f_\delta(\gamma)$. Also notice that we can *not* ensure that a corrupted D has committed $s \in \mathbb{F}$. This is because we are not checking whether $f_i(x), f_j(x)$ of every $P_i, P_j \in VCORE$ satisfies $f_i(j) = f_j(i)$. Performing such a check will require extra communication and computation in asynchronous settings. However, it is enough in our context that D is committed to a value (including $NULL$), which will be reconstructed uniquely during reconstruction phase.

Fig. 4 Sharing Phase of AVSS Scheme

AVSS-Share($D, \mathcal{P}, s, \epsilon$)

DISTRIBUTION: CODE FOR D — Only D executes this code

1. Select a random symmetric bivariate polynomial $F(x, y)$ of degree-t in x and y such that $F(0, 0) = s$ and send $f_i(x) = F(x, i)$ to party P_i, for $i = 1, \ldots, n$.

WSS SHARING OF $f_i(x)$: CODE FOR P_i — Every party, including D executes this code

1. Wait to obtain $f_i(x)$ from D.
2. If $f_i(x)$ is a degree-t polynomial then invoke AWSS-Share($P_i, \mathcal{P}, f_i(x), \epsilon'$) after selecting a symmetric bivariate polynomial $Q^{P_i}(x, y)$ of degree-t in x and y, such that $Q^{P_i}(x, 0) = q_0^{P_i}(x) = f_i(x)$ and $\epsilon' = \frac{\epsilon}{n}$. We call this instance of AWSS-Share initiated by P_i as AWSS-Share^{P_i}.
3. As a part of the execution of AWSS-Share^{P_j}, wait to receive $q_i^{P_j}(x) = Q^{P_j}(x, i)$ from P_j. Then check $f_i(j) \stackrel{?}{=} q_i^{P_j}(0)$. If the test passes then participate in AWSS-Share^{P_j} and act according to the remaining steps of AWSS-Share^{P_j}.

VCORE CONSTRUCTION: CODE FOR D — Only D executes this code

1. If AWSS-Share^{P_j} is terminated, then denote corresponding $WCORE$ and OKP_k sets by $WCORE^{P_j}$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$. Add P_j in a set $VCORE$ (initially empty).
2. Keep updating $VCORE$, $WCORE^{P_j}$ and corresponding $OKP_k^{P_j}$'s for every $P_j \in VCORE$ upon receiving new A-casts of the form OK(.,.) (during AWSS-Share^{P_j}s), until for at least $2t + 1$ $P_j \in VCORE$, the condition $|VCORE \cap WCORE^{P_j}| \geq 2t + 1$ is satisfied. Remove (from $VCORE$) all $P_j \in VCORE$ for whom the above condition is not satisfied.
3. A-cast $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$.

VCORE VERIFICATION & AGREEMENT ON VCORE : CODE FOR P_i — Every party executes this code

1. Wait to receive $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$ from D's A-cast.
2. Wait to terminate AWSS-Share^{P_j} corresponding to every P_j in $VCORE$.
3. Wait to receive OK(P_m, P_k) for every $P_k \in WCORE^{P_j}$ and every $P_m \in OKP_k^{P_j}$, corresponding to every $P_j \in VCORE$.
4. After receiving all the desired OK's, consider $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$ received from D as valid, accept them and terminate AVSS-Share.

High Level Idea of AVSS-Rec: In AVSS-Rec, we reconstruct D's AVSS-commitment. For this, it is enough to reconstruct the AWSS-shared $f_j(x)$'s of each honest $P_j \in VCORE$. So we execute AVSS-Rec for each $P_j \in VCORE$ to reconstruct either $NULL$ or $f_j(x)$. Now with the reconstructed $f_j(x)$'s, either $F(x,y)$ and hence $s = F(0,0)$ or $NULL$ will be reconstructed. The formal details of AVSS-Rec are given in Fig. 5.

Fig. 5 Reconstruction Phase of AVSS Scheme

AVSS-Rec($D, \mathcal{P}, s, \epsilon$)

SECRET RECONSTRUCTION: CODE FOR P_i — Every party executes this code

1. For every $P_j \in VCORE$, participate in AVSS-Rec($P_j, \mathcal{P}, f_j(x), \epsilon'$) with $WCORE^{P_j}$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$, where $\epsilon' = \frac{\epsilon}{n}$. We call this instance of AVSS-Rec as AVSS-Rec^{P_j}.
2. Wait for termination of AWSS-Rec^{P_j} for every $P_j \in VCORE$ with output either $\overline{f_j(x)}$ or $NULL$. Add P_j to $FINAL$ if AWSS-Rec^{P_j} gives non-$NULL$ output.
3. For every pair $(P_\gamma, P_\delta) \in FINAL$ check $\overline{f_\gamma(\delta)} \stackrel{?}{=} \overline{f_\delta(\gamma)}$. If the test passes then recover $\overline{F(x,y)}$ using $\overline{f_j(x)}$'s corresponding to each $P_j \in FINAL$ and set $\overline{s} = \overline{F(0,0)}$. Else set $\overline{s} = NULL$. Finally output $\overline{s}$ and terminate AVSS-Rec.

We now prove the properties of our AVSS scheme.

Lemma 9 (AVSS-Termination) *Protocols AVSS-Share, AVSS-Rec satisfies termination property of Definition 4.*

PROOF:

– **Termination 1:** In AVSS-Share, D keeps on updating (i.e., adding new parties) to $WCORE^{P_j}$ during AWSS-Share^{P_j}, even after $WCORE^{P_j}$ contains $2t+1$ parties. So if D is honest and all honest parties participate in the protocol, then $2t+1$ honest parties will be eventually included in $WCORE^{P_j}$ of every honest P_j. So eventually at least $2t+1$ honest parties will be included in $VCORE$, such that $|VCORE \cap WCORE^{P_j}| \geq 2t+1$ for each $P_j \in VCORE$. Now from similar argument given in **Termination 1** of Lemma 5, all honest parties will eventually accept $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ and will terminate AVSS-Share.

– **Termination 2:** If some honest party has terminated AVSS-Share then it implies that he has received $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$ from the A-cast of D and checked their validity. So by the property of A-cast, every other honest party will also eventually do the same and terminate AVSS-Share.

– **Termination 3:** Follows from the fact that corresponding to each $P_j \in VCORE$, every honest P_i will eventually terminate AWSS-Rec^{P_j} (from **Termination 3** of Lemma 5), except with an error probability of ϵ'. As there are at least $t+1$ honest parties in $VCORE$, AWSS-Rec corresponding to all honest parties in $VCORE$ will terminate with probability at least $(1-(t+1)\epsilon') \approx (1-\epsilon)$. □

Lemma 10 (AVSS-Correctness) *Protocol AVSS-Share, AVSS-Rec satisfies correctness property of Definition 4.*

PROOF:

– **Correctness 1:** We have to consider the case when D is honest. If D is *honest* then we prove that except with probability ϵ', AWSS-Rec^{P_i} will reconstruct $\overline{f_i(x)} = f_i(x)$ for every $P_i \in FINAL$. If P_i is *honest* then this follows from the **Correctness1** of our AWSS scheme. We now show that same holds even for a *corrupted* $P_i \in FINAL$. If a corrupted P_i belongs to $FINAL$, it implies that AWSS-Rec^{P_i} outputs a degree-t polynomial and AWSS-Share^{P_i} had terminated during AVSS-Share, such that $|VCORE \cap WCORE^{P_i}| \geq 2t+1$. The above statements have the following implications: as a part of AWSS-Share^{P_i}, P_i handed over $q_j^{P_i}(x)$ to an honest P_j (in $WCORE^{P_i}$) satisfying $f_j(i) = q_j^{P_i}(0)$. This further implies that P_i must have AWSS-shared $f_i(x)$. Thus if AWSS-Rec^{P_i} is successful, then except with probability ϵ', $\overline{f_i(x)} = f_i(x)$. In the worst case, there can be at most t corrupted parties in $FINAL$ and hence except with probability $\epsilon' t \approx \epsilon$, $\overline{f_i(x)}$'s corresponding to each $P_i \in FINAL$ will define $\overline{F(x,y)} = F(x,y)$ and thus $s = \overline{F(0,0)} = F(0,0)$ will be recovered.

– **Correctness 2:** Here we have to consider a *corrupted* D. Now there are two cases:
  1. D's AVSS-committed secret $s = NULL$: this implies that there exists some pair of honest parties $P_\gamma, P_\delta \in VCORE$, such that $f_\gamma(\delta) \neq f_\delta(\gamma)$. From **Correctness 1** of our AWSS scheme, for every honest $P_i \in VCORE$, AWSS-Rec^{P_i} will reconstruct $\overline{f_i(x)} = f_i(x)$ and thus P_i will be added to $FINAL$, except with error probability ϵ'. Since there are at least $t+1$ honest parties in $VCORE$, all the honest parties from $VCORE$ will be added to $FINAL$ except with error probability of $n\epsilon' = \epsilon$. Now irrespective of the remaining (corrupted) parties included in $FINAL$, the consistency checking (i.e., $\overline{f_\gamma(\delta)} \stackrel{?}{=} \overline{f_\delta(\gamma)}$) will fail for P_γ, P_δ and $NULL$ will be reconstructed.
  2. D's AVSS-committed secret $s = F(0,0)$: this case completely resembles the case when D is honest and so the proof follows from the proof of **Correctness 1**. □

Lemma 11 (AVSS-Secrecy) *Protocol AVSS-Share satisfies secrecy property of Definition 4.*

PROOF: Without loss of generality, let $P_1, \ldots, P_t$ be under the control of $\mathcal{A}_t$. It is easy to see that through out AVSS-Share, $\mathcal{A}_t$ will know $f_1(x), \ldots, f_t(x)$ and t points on $f_{t+1}(x), \ldots, f_n(x)$. However, from the property of symmetric polynomial of degree-t in x and y [17], the adversary $\mathcal{A}_t$ will lack one more point on $F(x, y)$ to uniquely interpolate $F(x, y)$ and get $s = F(0, 0)$. □

Lemma 12 (AVSS-Communication Complexity) *Protocol AVSS-Share incurs a private communication of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits. Protocol AVSS-Rec incurs A-cast of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits.*

PROOF: Follows from Lemma 8 and the fact that $\Theta(n)$ instances of AWSS-Share and AWSS-Rec are executed, each with an error parameter of $\epsilon' = \frac{\epsilon}{n}$. □

Remark 5 In AVSS-Share, we may assume that if D's AVSS-committed secret is $NULL$, then D has AVSS-committed some predefined value $s^* \in \mathbb{F}$, which is known publicly. Hence in AVSS-Rec, whenever $NULL$ is reconstructed, every honest party replaces $NULL$ by the predefined secret s^*. Interpreting this way, we say that our AVSS scheme allows D to *AVSS-commit* secret from $\mathbb{F}$.

4 Existing Common Coin Protocol

Here we recall the definition of common coin and construction of common coin protocol following the description of [14]. The common coin protocol invokes many instances of AVSS scheme. In the following description, we replace the AVSS scheme of [14] by our AVSS scheme presented in Section 3.3.

Definition 10 (Common Coin [14]) Let π be an asynchronous protocol, where each party has local random input and binary output. We say that π is a $(1-\epsilon)$-terminating, t-resilient common coin protocol if the following requirements hold for every adversary $\mathcal{A}_t$:

1. **Termination:** If all honest parties participate, then with probability at least $(1-\epsilon)$, all honest parties terminate.
2. **Correctness:** For every value $\sigma \in \{0, 1\}$, with probability at least $\frac{1}{4}$ all honest parties output σ.

The Intuition: The common coin protocol, referred as Common-Coin, consists of two stages. In the first stage, each party acts as a dealer and shares n random secrets, using n distinct instances of AVSS-Share each with allowed error probability of $\epsilon' = \frac{\epsilon}{n^2}$. The i^{th} secret shared by *each* party is actually associated with party P_i. Once a party P_i terminates any $t+1$ instances of AVSS-Share corresponding to $t+1$ secrets associated with him, he A-casts the identity of the dealers who have shared these $t+1$ secrets. We say that these $t+1$ secrets are `attached` to P_i and later these $t+1$ secrets will be used to compute a value that will be associated with P_i.

Now in the second stage, after terminating the AVSS-Share instances of all the secrets attached to some P_i, party P_j is sure that a fixed (yet unknown) value is attached to P_i. Once P_j is assured that values have been attached to enough number of parties, he participates in AVSS-Rec instances of the relevant secrets. This process of ensuring that there are enough parties that are attached with values is the core idea of the protocol. Once all the relevant secrets are reconstructed, each party locally computes his binary output based on the reconstructed secrets, in a way described in the protocol, which is presented in Fig. 6.

Fig. 6 Existing Common Coin Protocol

Protocol Common-Coin(ϵ)

CODE FOR P_i: — Every party executes this code

1. For $j = 1, \ldots, n$, choose a random value x_{ij} and execute AVSS-Share($P_i, \mathcal{P}, x_{ij}, \epsilon'$) where $\epsilon' = \frac{\epsilon}{n^2}$.
2. Participate in AVSS-Share($P_j, \mathcal{P}, x_{jk}, \epsilon'$) for every $j, k \in \{1, \ldots, n\}$. We denote AVSS-Share($P_j, \mathcal{P}, x_{jk}, \epsilon'$) by AVSS-Share$_{jk}$.
3. Create a dynamic set $\mathcal{T}_i$. Add party P_j to $\mathcal{T}_i$ if AVSS-Share($P_j, \mathcal{P}, x_{jk}, \epsilon'$) has been terminated for all $k = 1, \ldots, n$. Wait until $|\mathcal{T}_i| = t+1$. Then assign $T_i = \mathcal{T}_i$ and A-cast "`Attach` T_i `to` P_i". We say that the secrets $\{x_{ji} | P_j \in T_i\}$ are `attached` to party P_i.
4. Create a dynamic set $\mathcal{A}_i$. Add party P_j to $\mathcal{A}_i$ if
   (a) "`Attach` T_j `to` P_j" is received from the A-cast of P_j and
   (b) $T_j \subseteq \mathcal{T}_i$
   Wait until $|\mathcal{A}_i| = 2t+1$. Then assign $A_i = \mathcal{A}_i$ and A-cast "P_i `Accepts` A_i".
5. Create a dynamic set $\mathcal{S}_i$. Add party P_j to $\mathcal{S}_i$ if
   (a) "P_j `Accepts` A_j" is received from the A-cast of P_j and
   (b) $A_j \subseteq \mathcal{A}_i$.
   Wait until $|\mathcal{S}_i| = 2t+1$. Then A-cast "`Reconstruct Enabled`". Let H_i be the current content of $\mathcal{A}_i$.
6. Participate in AVSS-Rec($P_k, \mathcal{P}, x_{kj}, \epsilon'$) for every $P_k \in T_j$ of every $P_j \in \mathcal{A}_i$ (note that some parties may be included in $\mathcal{A}_i$ after the A-cast of "`Reconstruct Enabled`". The corresponding AVSS-Rec are invoked immediately). We denote AVSS-Rec($P_k, \mathcal{P}, x_{kj}, \epsilon'$) by AVSS-Rec$_{kj}$.
7. Let $u = \lceil 0.87n \rceil$. Every party $P_j \in \mathcal{A}_i$ is `associated` with a value, say V_j which is computed as follows: $V_j = (\sum_{P_k \in T_j} x_{kj}) \;\; mod \;\; u$ where x_{kj} is reconstructed back from AVSS-Rec($P_k, \mathcal{P}, x_{kj}, \epsilon'$).
8. Wait until the values `associated` with all the parties in H_i are computed. Now if there exits a party $P_j \in H_i$ such that $V_j = 0$, then output 0. Otherwise output 1.

Let E be an event, defined as follows: All invocations of AVSS scheme have been terminated properly. That is, if an honest party has terminated AVSS-Share, then a value, say s' is fixed. All honest parties will terminate the corresponding invocation of AVSS-Rec with output s'. Moreover if the dealer of this invocation of AVSS-Share is honest, then s' is indeed the shared secret of this invocation. It is easy to see that event E occurs with probability at least $1 - n^2\epsilon' = 1 - \epsilon$. We now state the following lemmas which are more or less identical to the Lemmas 5.28-5.31 presented in [14]. For the sake of completeness, the proofs of these lemmas are given in **APPENDIX B**.

Lemma 13 ([14]) *All honest parties terminate Protocol Common-Coin in constant time.*

Lemma 14 ([14]) *In Common-Coin, once some honest P_j receives "`Attach` T_i `to` P_i" from A-cast of P_i and includes P_i in $\mathcal{A}_j$, a unique value V_i is fixed such that*

1. *Every honest party will `associate` V_i with P_i, except with probability $1 - \frac{\epsilon}{n}$.*
2. *V_i is distributed uniformly over $[0, \ldots, u]$ and independent of values `associated` with other parties.*

Lemma 15 ([14]) *Once an honest party A-cast "`Reconstruct Enabled`", there exists a set M such that:*

1. *For every party $P_j \in M$, some honest party has received "`Attach` T_j `to` P_j" from the A-cast of P_j.*
2. *When any honest party P_j A-casts "`Reconstruct Enabled`", then it will hold that $M \subseteq H_j$.*
3. *$|M| \geq \frac{n}{3}$.*

Lemma 16 ([14]) *Let $\epsilon \leq 0.2$ and assume that all the honest parties have terminated protocol Common-Coin. Then for every value $\sigma \in \{0, 1\}$, with probability at least $\frac{1}{4}$, all the honest parties output σ.*

Theorem 4 ([14]) *Common-Coin is a $(1-\epsilon)$-terminating, common coin protocol for every $0 < \epsilon \leq 0.2$.*

PROOF: Follows from Lemma 13, 14, 15 and 16. □

Theorem 5 *Protocol Common-Coin privately communicates $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits and A-cast $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits.*

PROOF: Easy, as n^2 instances of AVSS-Share and AVSS-Rec are executed, each with error parameter $\frac{\epsilon}{n^2}$. □

5 Existing Voting Protocol

The Voting protocol is another requirement for the construction of ABA protocol. In a Voting protocol, every party has a single bit as input. Roughly, Voting protocol tries to find out whether there is a detectable majority for some value among the inputs of the parties. Here we recall the Voting protocol called Vote from [14].

The Intuition: Each party's output in Vote protocol can take *five* different forms:

1. For $\sigma \in \{0, 1\}$, the output $(\sigma, 2)$ stands for 'overwhelming majority for σ';
2. For $\sigma \in \{0, 1\}$, the output $(\sigma, 1)$ stands for 'distinct majority for σ';
3. Output $(\Lambda, 0)$ stands for 'non-distinct majority'.

We can show that:

1. If all the honest parties have the same input σ, then all honest parties will output $(\sigma, 2)$;
2. If some honest party outputs $(\sigma, 2)$, then every other honest party will output either $(\sigma, 2)$ or $(\sigma, 1)$;
3. If some honest party outputs $(\sigma, 1)$ and no honest party outputs $(\sigma, 2)$ then each honest party outputs either $(\sigma, 1)$ or $(\Lambda, 0)$.

The Vote protocol consists of three stages, having similar structure. The protocol is presented in Fig. 7. In the protocol, we assume party P_i has input bit x_i. We now recall the following lemmas and theorem from [14]. For the sake of completeness, the proofs of these lemmas and theorem are given in **APPENDIX C**.

Fig. 7 Existing Vote Protocol

Protocol Vote()

CODE FOR P_i: — Every party executes this code

1. A-cast $(\texttt{input}, P_i, x_i)$.
2. Create a dynamic set $\mathcal{A}_i$. Add (P_j, x_j) to $\mathcal{A}_i$ if $(\texttt{input}, P_j, x_j)$ is received from the A-cast of P_j.
3. Wait until $|\mathcal{A}_i| = n - t$. Assign $A_i = \mathcal{A}_i$. Set a_i to the majority bit among $\{x_j \mid (P_j, x_j) \in A_i\}$ and A-cast $(\texttt{vote}, P_i, A_i, a_i)$.
4. Create a dynamic set $\mathcal{B}_i$. Add (P_j, A_j, a_j) to $\mathcal{B}_i$ if $(\texttt{vote}, P_j, A_j, a_j)$ is received from the A-cast of P_j, $A_j \subseteq \mathcal{A}_i$, and a_j is the majority bit of A_j.
5. Wait until $|\mathcal{B}_i| = n - t$. Assign $B_i = \mathcal{B}_i$. Set b_i to the majority bit among $\{a_j \mid (P_j, A_j, a_j) \in B_i\}$ and A-cast $(\texttt{re-vote}, P_i, B_i, b_i)$.
6. Create a set C_i. Add (P_j, B_j, b_j) to C_i if $(\texttt{re-vote}, P_j, B_j, b_j)$ is received from the A-cast of P_j, $B_j \subseteq \mathcal{B}_i$, and b_j is the majority bit of B_j.
7. Wait until $|C_i| \geq n - t$. If all the parties $P_j \in B_i$ had the same vote $a_j = \sigma$, then output $(\sigma, 2)$ and terminate. Otherwise, if all the parties $P_j \in C_i$ have the same `Re-vote` $b_j = \sigma$, then output $(\sigma, 1)$ and terminate. Otherwise, output $(\Lambda, 0)$ and terminate.

Lemma 17 ([14]) *All the honest parties terminate protocol Vote in constant time.*

Lemma 18 **([14])** *If all honest parties have same input σ, then all honest parties will output $(\sigma, 2)$.*

Lemma 19 **([14])** *If some honest party outputs $(\sigma, 2)$, then every other honest party will eventually output either $(\sigma, 2)$ or $(\sigma, 1)$ in protocol Vote.*

Lemma 20 **([14])** *If some honest party outputs $(\sigma, 1)$ and no honest party outputs $(\sigma, 2)$ then every other honest party will eventually output either $(\sigma, 1)$ or $(\Lambda, 0)$.*

Theorem 6 *Protocol Vote A-cast of $\mathcal{O}(n^2 \log n)$ bits.*

PROOF: Follows from the protocol description. □

6 Efficient ABA Protocol for Single Bit

Once we have an efficient Common Coin protocol and Vote protocol, we can design an efficient ABA protocol using the approach of [14]. The ABA protocol proceeds in iterations where in each iteration every party computes a 'modified input' value. In the first iteration the 'modified input' of party P_i is his private input bit x_i. In each iteration, every party executes protocol Vote and Common-Coin sequentially. If a party outputs $\{(\sigma, 2), (\sigma, 1)\}$ in Vote protocol, then he sets his 'modified input' for next iteration to σ, irrespective of the value which is going to be output in Common-Coin. Otherwise, he sets his 'modified input' for next iteration to be the output of Common-Coin protocol which is invoked by all the honest parties in each iteration irrespective of whether the output of Common-Coin is used or not. Once a party outputs $(\sigma, 2)$, he A-casts σ and once he receives $t + 1$ A-cast for σ, he terminates the ABA protocol with σ as final output. The protocol is given in Fig. 8. We now state the following lemmas which are more or less identical to the Lemmas 5.36-5.39 presented in [14]. For the sake of completeness, their proofs are given in **APPENDIX D**.

Lemma 21 **([14])** *Protocol ABA satisfies* **Validity**.

Lemma 22 **([14])** *Protocol ABA satisfies* **Agreement**.

Lemma 23 **([14])** *If all honest parties have initiated and completed iteration k, then with probability at least $\frac{1}{4}$ all honest parties have same value for v_{k+1}.*

Let C_k be the event that each honest party completes all the iterations he initiated up to (and including) the k^{th} iteration (that is, for each iteration $1 \leq l \leq k$ and for each party P, if P initiated iteration l then he computes v_{l+1}). Let C denote the event that C_k occurs for all k.

Lemma 24 **([14])** *Conditioned on the event C, all honest parties terminate ABA in constant expected time.*

Fig. 8 **Efficient ABA Protocol for Single Bit.**

Protocol ABA(ϵ)

CODE FOR P_i: Every party executes this code

1. Set r = 0. and $v_1 = x_i$.
2. Repeat until terminating.
   (a) Set $r := r + 1$. Invoke Vote with v_r as input. Wait to terminate Vote and assign output of Vote to (y_r, m_r).
   (b) Invoke Common-Coin($\frac{\epsilon}{4}$) and wait until its termination. Let c_r be the output of Common-Coin.
   (c) i. If $m_r = 2$, set $v_{r+1} := y_r$ and A-cast (Terminate with v_{r+1}). Participate in *only one more* instance of Vote and *only one more* instance of Common-Coin protocol /* This is to prevent the parties from participating in an unbounded number of iterations before enough (Terminate with σ) A-casts are completed.*/
   ii. If $m_r = 1$, set $v_{r+1} := y_r$.
   iii. Otherwise, set $v_{r+1} := c_r$.
   (d) Upon receiving $t + 1$ (Terminate with σ) A-cast for same value σ, output σ and terminate ABA.

Lemma 25 **([14])** *$Prob(C) \geq (1 - \epsilon)$.*

Summing up, we have the following theorem.

Theorem 7 **(ABA for Single Bit)** *Let $n = 3t + 1$. Then for every $0 < \epsilon \leq 0.2$, protocol ABA is a $(\epsilon, 0)$-ABA protocol. Given the parties terminate, they do so in constant expected time. The protocol privately communicates $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits and A-cast $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits.*

PROOF: The properties of ABA follows from Lemma 21, 22, 23 and Lemma 24. Let $\mathcal{C}$ be the expected number of time Common-Coin and Vote protocol are executed in ABA protocol. Then from Theorem 5 protocol ABA privately communicates $\mathcal{O}(\mathcal{C}n^6 \log \frac{1}{\epsilon})$ bits and A-cast $\mathcal{O}(\mathcal{C}n^6 \log \frac{1}{\epsilon})$ bits. Substituting $\mathcal{C} = \mathcal{O}(1)$, we get the final communication complexity. □

7 Efficient ABA Protocol for Multiple Bits

Till now we have concentrated on the construction of efficient ABA protocol that allows the parties to agree on a *single* bit. We now present another efficient ABA protocol called ABA-MB [4], which achieves agreement on $n - 2t = t + 1$ bits *concurrently*. Notice that we could *parallely* execute protocol ABA $t + 1$ times to achieve agreement on $t + 1$ bits. This would require a private communication as well as A-cast of $\mathcal{O}(n^7 \log \frac{1}{\epsilon})$ bits. However our protocol ABA-MB requires private communication and A-cast of $\mathcal{O}(n^5 \log \frac{1}{\epsilon})$ bits for the same task. Consequently, in protocol ABA-MB, the *amortized*

[4] Here MB stands for multiple bits.

cost to reach agreement on a *single* bit is $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits of private and A-cast communication.

In asynchronous multiparty computation (AMPC) [6,14,3,46], where typically lot of ABA invocations are required, many of the invocations can be parallelized and optimized to a single invocation with a long message. Hence ABA protocols with long message are very relevant to many situations. All existing protocols for ABA [50,4,10,27,28,15,14,1,47] are designed for single bit message. A naive approach to design ABA for $\ell > 1$ bit message is to parallelize ℓ invocations of existing ABA protocols dealing with single bit. This approach requires a communication complexity that is ℓ times the communication complexity of the existing protocols for single bit and hence is inefficient. In this article, we provide a far better way to design an ABA with multiple bits. For ℓ bits message with $\ell \geq t+1$, we may break the message into blocks of $t+1$ bits and invoke one instance of our ABA-MB for each one of the $t+1$ blocks. To design ABA-MB, we extend our AWSS and AVSS scheme to share $\ell > 1$ secrets *simultaneously*. This involves less communication complexity than ℓ parallel invocations of our AWSS and AVSS scheme sharing *single* secret.

7.1 AWSS Scheme for Sharing Multiple Secrets

We now extend protocol AWSS-Share and AWSS-Rec to AWSS-MS-Share and AWSS-MS-Rec respectively [5]. Protocol AWSS-MS-Share allows $D \in \mathcal{P}$ to concurrently share a secret $S = (s^1 \dots s^\ell)$, containing ℓ elements. On the other hand, protocol AWSS-MS-Rec allows the parties in $\mathcal{P}$ to reconstruct either S or $NULL$.

The Intuition: The high level idea of protocol AWSS-MS-Share is similar to AWSS-Share. For each $s^l, l = 1, \dots, \ell$, the dealer D selects a random symmetric bivariate polynomial $F^l(x, y)$ of degree-t in x and y, where $F^l(0,0) = s^l$ and gives his IC-signature on $f_i^l(1), \dots, f_i^l(n)$ to party P_i, for $i = 1, \dots, n$. However, to reduce the communication complexity, instead of executing ℓn^2 instances of AICP (each dealing with a *single* secret), D executes n^2 instances of AICP (each dealing with ℓ secrets) and D gives his IC-signature *collectively* on $(f_i^1(j), f_i^2(j), \dots, f_i^\ell(j))$ to P_i.

Next, every P_i, P_j exchange their IC signatures on common values. Notice that now P_i, P_j have ℓ common values, namely $f_i^1(j), \dots, f_i^\ell(j)$. Intead of exchanging IC signatures on individual common value, they exchange IC signatures *collectively* on $(f_i^1(j), \dots, f_i^\ell(j))$ and $(f_j^1(i), \dots, f_j^\ell(i))$. Next the parties check whether $f_i^l(j) = f_j^l(i)$ *for all* $l = 1, \dots, \ell$ and if so they A-cast OK signal. After this, the remaining steps (like $WCORE$ construction, agreement on $WCORE$, etc) are same as in AWSS-Share. The protocol is given in Fig. 9.

[5] Here MS stands for multiple secrets

Fig. 9 Sharing Phase of AWSS Scheme for Sharing S Containing $\ell \geq 1$ Secrets

AWSS-MS-Share($D, \mathcal{P}, S = (s^1 \dots s^\ell), \epsilon$)

DISTRIBUTION: CODE FOR D – Only D executes this code.

1. For $l = 1, \dots, \ell$, select a random, symmetric bivariate polynomial $F^l(x, y)$ of degree-t in x and y such that $F^l(0,0) = s^l$. Let $f_i^l(x) = F^l(x, i)$, for $l = 1, \dots, \ell$.
2. For $i = 1, \dots, n$, give $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \dots, f_i^\ell(j))$ for each $j = 1, \dots, n$ to P_i. For this, D initiates n^2 instances of Gen, each with an error parameter of $\epsilon' = \frac{\epsilon}{n^2}$.

VERIFICATION: CODE FOR P_i – Every party including D executes this code.

1. Wait to receive $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \dots, f_i^\ell(j)))$ for $j = 1, \dots, n$ from D.
2. Check if $(f_i^l(1), \dots, f_i^l(n))$ defines degree-t polynomial for $l = 1, \dots, \ell$. If yes then give $ICSig(P_i, P_j, \mathcal{P}, (f_i^1(j), \dots, f_i^\ell(j)))$ to P_j for $j = 1, \dots, n$.
3. If $ICSig(P_j, P_i, \mathcal{P}, (f_j^1(i), \dots, f_j^\ell(i)))$ is received from P_j and if $f_j^l(i) = f_i^l(j)$ for $l = 1, \dots, \ell$, then A-cast OK(P_i, P_j).

WCORE CONSTRUCTION : CODE FOR D — This is same as in protocol AWSS-Share.

WCORE VERIFICATION & AGREEMENT ON WCORE — This is same as in protocol AWSS-Share.

Remark 6 (D's **AWSS-commitment)** In AWSS-MS-Share, we say that D has AWSS-committed $S = (s^1, \dots, s^\ell) \in \mathbb{F}^\ell$ if for every $l = 1, \dots, \ell$, there is a unique degree-t polynomial $f^l(x)$ such that $f^l(0) = s^l$ and every *honest* P_i in $WCORE$ receives $f^l(i)$ from D. Otherwise, we say that D has AWSS-committed $NULL$.

An *honest* D always AWSS-commits $S \in \mathbb{F}^\ell$, as in this case $f^l(x) = f_0^l(x) = F^l(x, 0)$, where $F^l(x, y)$ is the symmetric bivariate polynomial of degree-t chosen by D. But AWSS-MS-Share can *not* ensure that a *corrupted* D also AWSS-commits $S \in \mathbb{F}^\ell$. Protocol AWSS-MS-Rec is a straightforward extension of protocol AWSS-Rec and is given in Fig. 10.

Since technique wise, protocols (AWSS-MS-Share, AWSS-MS-Rec) are very similar to protocols (AWSS-Share, AWSS-Rec), we do not provide the proofs of the properties of protocols (AWSS-MS-Share, AWSS-MS-Rec) for the sake of avoiding repetition. Rather, we give the following theorem on the communication complexity.

Theorem 8 (AWSS-MS-Communication Complexity) *Protocol AWSS-MS-Share incurs a private communication of $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}((\ell n^2+n^3) \log \frac{1}{\epsilon})$ bits. Protocol AWSS-MS-Rec involves A-cast of $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits.*

Fig. 10 Reconstruction Phase of AWSS Scheme for Sharing S Containing ℓ Secrets

AWSS-MS-Rec($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \epsilon$)

SIGNATURE REVELATION: CODE FOR P_i —

1. If P_i belongs to OKP_j for some $P_j \in WCORE$, then reveal $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \ldots, f_i^\ell(j)))$ and $ICSig(P_j, P_i, \mathcal{P}, (f_j^1(i), \ldots, f_j^\ell(i)))$.

LOCAL COMPUTATION: CODE FOR P_i

1. For every $P_j \in WCORE$, reconstruct P_j's *IC-commitment*, say $(\overline{f_j^1(0)}, \ldots, \overline{f_j^\ell(0)})$ as follows:
   (a) Construct a set $ValidP_j = \emptyset$.
   (b) Add $P_k \in OKP_j$ to $ValidP_j$ if the following conditions hold:
      i. Revelation of $ICSig(D, P_k, \mathcal{P}, (f_k^1(j), \ldots, f_k^\ell(j)))$ and $ICSig(P_j, P_k, \mathcal{P}, (f_j^1(k), \ldots, f_j^\ell(k)))$ are completed with Reveal$_i = (\overline{f_k^1(j)}, \ldots, \overline{f_k^\ell(j)})$ and Reveal$_i = (\overline{f_j^1(k)}, \ldots, \overline{f_j^\ell(k)})$ respectively; and
      ii. $\overline{f_k^l(j)} = \overline{f_j^l(k)}$, for $l = 1, \ldots, \ell$.
   (c) Wait until $|ValidP_j| = t + 1$. For $l = 1, \ldots, \ell$, construct a degree-t polynomial $\overline{f_j^l(x)}$ passing through the points $(k, \overline{f_j^l(k)})$ where $P_k \in ValidP_j$. For $l = 1, \ldots, \ell$, associate $\overline{f_j^l(0)}$ with $P_j \in WCORE$.
2. Wait for $\overline{f_j^1(0)}, \ldots, \overline{f_j^\ell(0)}$ to be reconstructed for every P_j in $WCORE$.
3. For $l = 1, \ldots, \ell$, do the following:
   (a) Check whether the points $(j, \overline{f_j^l(0)})$ for $P_j \in WCORE$ lie on a unique degree-t polynomial $\overline{f_0^l(x)}$. If yes, then set $\overline{s^l} = \overline{f_0^l(0)}$, else set $\overline{s^l} = NULL$.
4. If $\overline{s^l} = NULL$ for any $l \in \{1, \ldots, \ell\}$, then output $\overline{S} = NULL$ and terminate AWSS-MS-Rec. Else output $\overline{S} = (\overline{s^1}, \ldots, \overline{s^\ell})$ and terminate AWSS-MS-Rec.

PROOF: Follows from the fact that n^2 instances of AICP, each dealing with ℓ values and having error parameter of $\epsilon' = \frac{\epsilon}{n^2}$ are executed. □

Notation 4 (AWSS Sharing of ℓ Polynomials) *As in Notation 3, we abuse the notion of AWSS-sharing of ℓ secrets and say that:*

1. *D executes AWSS-MS-Share($D, \mathcal{P}, (f^1(x), \ldots, f^\ell(x)), \epsilon$) to mean that D AWSS-shares degree-t polynomials $f^1(x), \ldots, f^\ell(x)$ during AWSS-MS-Share. To do so, D will choose ℓ symmetric bivariate polynomial $F^l(x, y)$, for $l = 1, \ldots, \ell$, each of degree-t in x and y, where $F^l(x, 0) = f^l(x)$ holds and will execute the steps of protocol AWSS-MS-Share.*
2. *Parties execute AWSS-MS-Rec($D, \mathcal{P}, (f^1(x), \ldots, f^\ell(x)), \epsilon$), which allows the (honest) parties to reconstruct either the AWSS-Shared polynomials $f^1(x), \ldots, f^\ell(x)$ or $NULL$, except with probability ϵ.* □

7.2 AVSS Scheme for Sharing Multiple Secrets

We now extend protocol AVSS-Share and AVSS-Rec to AVSS-MS-Share and AVSS-MS-Rec respectively Protocol AVSS-MS-Share allows $D \in \mathcal{P}$ to concurrently share a secret $S = (s^1 \ldots s^\ell)$, containing ℓ elements. Moreover, if D is corrupted then either $S \in \mathbb{F}^\ell$, where each element of S belongs to $\mathbb{F}$ or $S = NULL$ (in a sense explained in the sequel). Protocol AVSS-MS-Rec allows the parties in $\mathcal{P}$ to reconstruct S.

The Intuition: The high level idea of AVSS-MS-Share is similar to AVSS-Share. Specifically, for each $s^l \in S$, the dealer D selects a symmetric bivariate polynomial $F^l(x, y)$ of degree-t in x and y, such that $F^l(0, 0) = s^l$ and sends $f_i^l(x) = F^l(x, i)$ to party P_i. Then each party P_i is asked to *AWSS-share* his received polynomials $f_i^1(x), \ldots, f_i^\ell(x)$. However, instead of executing ℓ instances of AWSS-Share, one for sharing each $f_i^l(x)$, party P_i executes a *single* instance of AWSS-MS-Share to share $f_i^1(x), \ldots, f_i^\ell(x)$ simultaneously. It is this step, which leads to the reduction in the communication complexity of AVSS-MS-Share. The remaining steps like $VCORE$ construction, agreement on $VCORE$, etc are similar to protocol AVSS-Share. Protocol AVSS-MS-Share is formally presented in Fig. 11.

Remark 7 (D's **AVSS-commitment**) We say that D has AVSS-committed $S = (s^1, \ldots, s^\ell) \in \mathbb{F}^\ell$ in AVSS-MS-Share if for every $l = 1, \ldots, \ell$ there is a unique degree-t symmetric bivariate polynomial $F^l(x, y)$ such that $F^l(0, 0) = s^l$ and every *honest* P_i in $VCORE$ receives $f_i^l(x) = F^l(x, i)$ from D. Otherwise, we say that D has committed $NULL$ and D's AVSS-committed secrets are not *meaningful*.

If a *corrupted* D commits $NULL$, the $f_i^l(x)$ polynomials of the honest parties in $VCORE$ do not define a symmetric bivariate polynomial of degree-t in x and y for at least one $l \in \{1, \ldots, \ell\}$. This further implies that there will be an honest pair (P_γ, P_δ) in $VCORE$ such that $f_\gamma^l(\delta) \neq f_\delta^l(\gamma)$.

Protocol AVSS-MS-Rec is a straightforward extension of protocol AVSS-Rec and is given in Fig. 12. The properties of AVSS-MS-Share and AVSS-MS-Rec follows from AVSS-Share and AVSS-Rec. For the sake of completeness, we state the communication complexity of AVSS-MS-Share and AVSS-MS-Rec.

Theorem 9 (AVSS-MS-Communication Complexity) *Protocol AVSS-MS-Share incurs a private communication and A-cast of $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$ bits. Protocol AVSS-MS-Rec involves A-cast of $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$ bits.*

Fig. 11 Sharing Phase of AVSS Scheme for Sharing a Secret S Containing ℓ Elements

AVSS-MS-Share($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \epsilon$)

DISTRIBUTION: CODE FOR D — Only D executes this code.

1. For $l = 1, \ldots, \ell$, select a random symmetric bivariate polynomial $F^l(x, y)$ of degree-t in x and y such that $F^l(0, 0) = s^l$ and send $f_i^l(x) = F^l(x, i)$ to party P_i, for $i = 1, \ldots, n$.

AWSS SHARING OF POLYNOMIALS: CODE FOR P_i — Every party in $\mathcal{P}$, including D, executes this code.

1. Wait to obtain $f_i^1(x), \ldots, f_i^\ell(x)$ from D.
2. If $f_i^1(x), \ldots, f_i^\ell(x)$ are degree-t polynomials then as a dealer, execute AWSS-MS-Share($P_i, \mathcal{P}, (f_i^1(x), \ldots, f_i^\ell(x)), \epsilon'$) by selecting symmetric bivariate polynomials $Q^{(P_i,1)}(x, y), \ldots, Q^{(P_i,\ell)}(x, y)$ of degree-t in x and y, such that $Q^{(P_i,l)}(x, 0) = q_0^{(P_i,l)}(x) = f_i^l(x)$, for $l = 1, \ldots, \ell$ and $\epsilon' = \frac{\epsilon}{n}$. We call this instance of AWSS-MS-Share initiated by P_i as AWSS-MS-Share^{P_i}.
3. As a part of the execution of AWSS-MS-Share^{P_j}, wait to receive $q_i^{(P_j,l)}(x) = Q^{(P_j,l)}(x, i)$, for $l = 1, \ldots, \ell$ from P_j. Then check $f_i^l(j) \stackrel{?}{=} q_i^{P_j,l}(0)$. If the test passes for all $l = 1, \ldots, \ell$ then participate in AWSS-MS-Share^{P_j} and act according to the remaining steps of AWSS-MS-Share^{P_j}.

VCORE CONSTRUCTION: CODE FOR D – This is same as in protocol AVSS-Share except that AWSS-Share is replaced by AWSS-MS-Share everywhere.

VCORE VERIFICATION & AGREEMENT ON VCORE : CODE FOR P_i — This is same as in protocol AVSS-Share except that AWSS-Share is replaced by AWSS-MS-Share everywhere.

Fig. 12 Reconstruction Phase of AVSS Scheme for Sharing Secret S Containing ℓ Elements

AVSS-MS-Rec($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \epsilon$)

SECRET RECONSTRUCTION: CODE FOR P_i — Every party in $\mathcal{P}$ executes this code.

1. For every $P_j \in VCORE$, participate in AWSS-MS-Rec($P_j, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x)), \epsilon'$). We call this instance of AWSS-MS-Rec as AWSS-MS-Rec^{P_j}.
2. Wait for termination of AWSS-MS-Rec^{P_j} for every $P_j \in VCORE$ with output either $(\overline{f_j^1(x)}, \ldots, \overline{f_j^\ell(x)})$ or $NULL$. Add P_j to $FINAL$ if AWSS-MS-Rec^{P_j} gives non-$NULL$ output.
3. For $l = 1, \ldots, \ell$, do the following: for every pair $(P_\gamma, P_\delta) \in FINAL$ check $\overline{f_\gamma^l(\delta)} \stackrel{?}{=} \overline{f_\delta^l(\gamma)}$. If the test passes for every pair of parties then recover $\overline{F^l(x, y)}$ using $\overline{f_j^l(x)}$'s corresponding to each $P_j \in FINAL$ and reconstruct $\overline{s^l} = \overline{F^l(0, 0)}$. Else reconstruct $\overline{s^l} = NULL$.
4. For $l = 1, \ldots, \ell$, if any $\overline{s^l} = NULL$ then output $\overline{S} = NULL$, else output $\overline{S} = (\overline{s^1}, \ldots, \overline{s^\ell})$ and terminate.

PROOF: Follows from the fact that n instances of AWSS-MS-Share and AWSS-MS-Rec are executed. □

Remark 8 In AVSS-MS-Share, we may assume that if D's AVSS-committed secret is $NULL$, then D has AVSS-committed some predefined $S^\star \in \mathbb{F}^\ell$, which is known publicly. Hence in AVSS-MS-Rec, whenever $NULL$ is reconstructed, every honest party replaces $NULL$ by the predefined $S^\star$. Interpreting this way, we say that our AVSS scheme allows D to *AVSS-commit* secrets from $\mathbb{F}$.

7.3 An Incorrect Common Coin Protocol

Recall that in protocol Common-Coin, each party invokes n instances of protocol AVSS-Share each sharing a single secret. Simple thinking would suggest that those n instances of AVSS-Share could be replaced by more efficient, single instance of AVSS-MS-Share, sharing n secrets simultaneously. This would naturally lead to more efficient common coin protocol, which would further imply more efficient ABA protocol. In the following, we do the same in protocol Common-Coin-Wrong. But as the name suggests, we then show that this direct replacement of AVSS-Share by AVSS-MS-Share without further modification will lead to an incorrect common coin protocol. Protocol Common-Coin-Wrong is given in Fig. 13.

We now show that protocol Common-Coin-Wrong does not satisfy second part of Lemma 14. That is, the adversary can behave in such a way that unique value V_i, `associate`ed with an *honest* P_i may not be distributed uniformly over $[0, \ldots, u]$. More specifically, $\mathcal{A}_t$ can decide V_i for up to $t - 1$ honest parties and thus those V_i's are no longer random and uniformly distributed over $[0, \ldots, u]$. Consequently, $\mathcal{A}_t$ can enforce *some honest parties* to *always output* 0, while other honest parties *may output* $\sigma \in \{0, 1\}$ with probability at least $\frac{1}{4}$. This will strictly violate the property of of common coin.

Let P_i be an *honest* party. We now describe a specific behavior of $\mathcal{A}_t$ in Common-Coin-Wrong which would allow $\mathcal{A}_t$ to decide V_i to be 0 and thus make honest P_i to output 0 (this can be extended for $t - 1$ honest P_is) whereas the remaining honest parties output $\sigma \in \{0, 1\}$ with probability at least $\frac{1}{4}$. The specific behavior is given in Fig. 14.

The Reason for the Problem: The adversary behavior specified in Fig. 14 become possible due to the fact that a *corrupted* P_j is able to select his secret x_{ji} for an *honest* P_i after knowing the secrets which other *honest* parties has selected for P_i. This was not possible in Common-Coin because every party $P_k \in T_i$ shared their secrets *independently* using *different* instance of AVSS-Share and as per requirement, corresponding AVSS-Rec was invoked to reconstruct the desired secret. However

Fig. 13 An Incorrect Common Coin Protocol Obtained by Replacing AVSS-Share and AVSS-Rec by AVSS-MS-Share and AVSS-MS-Rec Respectively in Protocol Common-Coin

Protocol Common-Coin-Wrong(ϵ)

CODE FOR P_i: — Every party in $\mathcal{P}$ executes this code.

1. For $j = 1, \ldots, n$, choose a random value x_{ij} and execute AVSS-MS-Share($P_i, \mathcal{P}, (x_{i1}, \ldots, x_{in}), \epsilon'$) where $\epsilon' = \frac{\epsilon}{n}$.
2. Participate in AVSS-MS-Share($P_j, \mathcal{P}, (x_{j1}, \ldots, x_{jn}), \epsilon'$) for every $j \in \{1, \ldots, n\}$. We denote AVSS-MS-Share($P_j, \mathcal{P}, (x_{j1}, \ldots, x_{jn}), \epsilon'$) by AVSS-MS-Share$_j$.
3. Create a dynamic set $\mathcal{T}_i$. Add party P_j to $\mathcal{T}_i$ if AVSS-MS-Share($P_j, \mathcal{P}, (x_{j1}, \ldots, x_{jn}), \epsilon'$) has been completed. Wait until $|\mathcal{T}_i| = t + 1$. Then assign $T_i = \mathcal{T}_i$ and A-cast "`Attach` T_i `to` P_i". We say that the secrets $\{x_{ji} | P_j \in T_i\}$ are the secrets `attached` to party P_i.
4. Create a dynamic set $\mathcal{A}_i$. Add P_j to $\mathcal{A}_i$ if following holds:
   (a) "`Attach` T_j `to` P_j" is received from A-cast of P_j;
   (b) $T_j \subseteq \mathcal{T}_i$.
   Wait until $|\mathcal{A}_i| = n - t$. Then assign $A_i = \mathcal{A}_i$ and A-cast "P_i `Accepts` A_i".
5. Create a dynamic set $\mathcal{S}_i$. Add P_j to $\mathcal{S}_i$ if following holds:
   (a) "P_j `Accepts` A_j" is received from the A-cast of P_j and
   (b) $A_j \subseteq \mathcal{A}_i$.
   Wait until $|\mathcal{S}_i| = n - t$. Then A-cast "`Reconstruct Enabled`". Let H_i be the current content of $\mathcal{A}_i$.
6. Participate in AVSS-MS-Rec($P_k, \mathcal{P}, (x_{k1}, \ldots, x_{kn}), \epsilon'$) for every $P_k \in T_j$ of every $P_j \in \mathcal{A}_i$ (Note that some parties may be included in $\mathcal{A}_i$ after the A-cast of "`Reconstruct Enabled`". The corresponding AVSS-MS-Rec are invoked immediately). We denote AVSS-MS-Rec($P_k, \mathcal{P}, (x_{k1}, \ldots, x_{kn}), \epsilon'$) by AVSS-MS-Rec$_k$.
7. Let $u = \lceil 0.87n \rceil$. Every party $P_j \in \mathcal{A}_i$ is `associated` with a value, say V_j which is computed as follows: $V_j = (\sum_{P_k \in T_j} x_{kj}) \;\; mod \;\; u$ where x_{kj} is reconstructed back after executing AVSS-MS-Rec($P_k, \mathcal{P}, (x_{k1}, \ldots, x_{kn}), \epsilon'$).
8. Wait until the values `associated` with all the parties in H_i are computed. Now if there exits a party $P_j \in H_i$ such that $V_j = 0$, then output 0. Otherwise output 1.

Fig. 14 Adversary Behavior in Common-Coin-Wrong

Possible Behavior of $\mathcal{A}_t$ in Protocol Common-Coin-Wrong() with respect to an honest P_i

1. Let P_j be a *corrupted* party. All corrupted parties participate in Common-Coin-Wrong honestly. However, P_j does not start AVSS-MS-Share$_j$.
2. Except for AVSS-MS-Share$_i$ and corresponding AVSS-MS-Rec$_i$, $\mathcal{A}_t$ (as a scheduler) stops all the messages sent to P_i and sent by P_i in every other AVSS-MS-Share$_k$ and corresponding AVSS-MS-Rec$_k$. This will prevent P_i to participate in any AVSS-MS-Share$_k$ and corresponding AVSS-MS-Rec$_k$ and hence to construct $\mathcal{T}_i$. However, this will not prevent P_i to be part of $\mathcal{T}_k$ for some P_k. $\mathcal{A}_t$ does so until the following happen:
   (a) $n - t - 1$ honest parties (except P_i) and $t - 1$ corrupted parties (except P_j) carry out all the steps of Common-Coin-Wrong honestly, construct respective sets, A-cast "`Reconstruct Enabled`" and start invoking corresponding AVSS-MS-Rec$_k$ protocols. This way the n secrets of each of $n - t - 1$ honest parties (except P_i) and $t - 1$ corrupted parties will be revealed. /* It is to be noted that the corrupted parties can successfully reconstruct secrets in each AVSS-MS-Rec$_k$ by behaving honestly, even if the honest P_i is unable to participate in AVSS-MS-Rec$_k$'s.*/
   (b) Now $\mathcal{A}_t$ computes a set T_i of size $t + 1$ containing the *corrupted* P_j and *any t honest* P_k's, whose AVSS-MS-Rec$_k$'s have been terminated. Notice that now the shared values $(x_{k1}, \ldots, x_{kn})$, corresponding to each honest $P_k \in T_i$ are known to the adversary.
   (c) Now $\mathcal{A}_t$ selects x_{ji}, corresponding to P_j, such that $V_i = (\sum_{P_k \in T_i} x_{ki}) \;\; mod \;\; u = 0$. Now $\mathcal{A}_t$ asks the corrupted P_j to invoke AVSS-MS-Share$_j$ with x_{ji} as the secret assigned to P_i.
3. $\mathcal{A}_t$ now schedules the messages to and from P_i corresponding to every AVSS-MS-Share$_k$ in such a way that T_i computed by $\mathcal{A}_t$ (in step 2(b)) indeed becomes T_i for P_i and P_i A-casts "`Attach` T_i `to` P_i" and eventually includes P_i in $\mathcal{A}_i$. So clearly H_i will contain P_i and hence P_i will output 0 since V_i is 0.

in Common-Coin-Wrong, *simultaneous* sharing and reconstruction of n secrets is performed using AVSS-MS-Share and AVSS-MS-Rec. So if a party P_l containing an *honest* P_k in T_l A-cast "`Reconstruct Enabled`" early and starts executing AVSS-MS-Rec$_k$, then it will disclose the *desired* secret x_{kl}; but at the same time it will disclose other $n - 1$ *undesired* secrets, selected by P_k corresponding to other $n - 1$ parties. Now later the adversary may always schedule messages such that P_i includes such *honest* P_k's in T_i and some other corrupted parties who have seen the secrets shared by P_k for P_i and then have shared their secrets for P_i. This clearly shows that the adversary can completely control the final output of P_i by deciding the value to be associated with P_i. This problem can be eliminated if we can ensure that no corrupted party can ever share any secret after any honest party starts reconstructing secrets. This is what we have achieved in our new common coin protocol presented in the next section.

7.4 A New Common Coin Protocol for Multiple Bits

In this section, we show how to enhance protocol Common-Coin, so that it can handle the problem described in the previous section and can still use protocols AVSS-MS-Share and AVSS-MS-Rec as black-boxes. We first give the following definition:

Definition 11 (Multi-Bit Common Coin) Let π be an asynchronous protocol, where each party has local random input and ℓ bit output, where $\ell \geq 1$. We say that π is a $(1 - \epsilon)$-terminating, t-resilient, multi-bit common coin protocol if the following holds:

1. **Termination:** If all honest parties participate, then with probability $(1-\epsilon)$, all honest parties terminate.
2. **Correctness:** For $l = 1, \ldots, \ell$, all honest parties output σ_l with probability at least $\frac{1}{4}$ for every $\sigma_l \in \{0, 1\}$.

We now present a multi-bit common coin protocol, called Common-Coin-MB, which goes almost in the same line as Common-Coin-Wrong except that we add some more steps and modify some of the steps due to which the corrupted parties are forced to share their secrets much before they can reconstruct anybody elses' secrets. We now discuss the high level idea of the protocol.

The Intuition: Each party shares n random secrets, using a single instance of AVSS-MS-Share, where the i^{th} secret is associated with P_i. Now a party P_i adds a party P_j to $\mathcal{T}_i$, only when at least $n-t$ parties have terminated P_j's instance of AVSS-MS-Share. Recall that in Common-Coin-Wrong, P_i adds P_j to $\mathcal{T}_i$, when P_i *himself* has terminated P_j's instance of AVSS-MS-Share. After that party P_i constructs $\mathcal{T}_i$, $\mathcal{A}_i$ and $\mathcal{S}_i$ and A-cast T_i, A_i and "`Reconstruct Enabled`" in the same way as performed in Common-Coin-Wrong, except with the following difference: P_i ensures T_i to contain $n-t$ parties (contrary to $t+1$ parties in Common-Coin-Wrong). The reason for enforcing $|T_i| = n - t$ is to obtain multiple bit output in protocol Common-Coin-MB and will be clear in the sequel. Now what follows is the *most important step* of Common-Coin-MB. Party P_i starts participating in AVSS-MS-Rec of the parties who are in his $\mathcal{T}_i$ only after receiving at least $n-t$ "`Reconstruct Enabled`" A-casts. Moreover party P_i halts execution of all the instances of AVSS-MS-Share corresponding to the parties not in $\mathcal{T}_i$ currently and later resume them only when they are included in $\mathcal{T}_i$. This step along with the step for constructing $\mathcal{T}_i$ will ensure the desired property that in order to be part of any honest party's $\mathcal{T}_i$, a corrupted party must have to commit his secrets well before the first honest party receives $n-t$ "`Reconstruct Enabled`" A-casts and starts reconstructing secrets. This ensures that a corrupted party who is in $\mathcal{T}_i$ of any honest party had no knowledge what so ever about the secrets committed by other honest parties at the time he commits to his own secrets.

Let us see, how our protocol steps achieve the above task. Let P_i be the *first honest party* to receive $n-t$ "`Reconstruct Enabled`" A-casts and start invoking reconstruction process. Also let P_k be a *corrupted party* who belongs to $\mathcal{T}_j$ of some honest party P_j. This means that at least $t+1$ honest parties have already terminated AVSS-MS-Share instance of P_k (this is because P_j has added P_k in $\mathcal{T}_j$ only after confirming that $n-t$ parties have terminated P_k's instance of AVSS-MS-Share). This further means that there is at least one honest party, say P_α, who terminated P_k's instance of AVSS-MS-Share before A-casting "`Reconstruct Enabled`" (because if it not the case, then the honest party P_α would have halted the execution of P_k's instance of AVSS-MS-Share for ever and would never terminate it). This indicates that P_k is already committed to his secrets before the first honest party receives $n-t$ "`Reconstruct Enabled`" A-casts and starts the reconstruction. A more detailed proof is given in Lemma 27.

Another important feature of protocol Common-Coin-MB is that it is a multi-bit common coin protocol. This is attained by using the ability of Vandermonde matrix [53, 19] for extracting randomness. As a result, we could associate $n-2t$ values with each P_i, namely $V_{i1}, \ldots, V_{i(n-2t)}$ in Common-Coin-MB, while a single value V_i was associated with P_i in Common-Coin. This leads every party to output $\ell = n-2t$ bits in protocol Common-Coin-MB. We now briefly recall the properties of Vandermonde matrix and then present our protocol.

Vandermonde Matrix and Randomness Extraction [53, 19]: Let $\beta_1, \ldots, \beta_c$ be distinct and publicly known elements of $\mathbb{F}$. We denote an $(r \times c)$ Vandermonde matrix by $V^{(r,c)}$, where for $i = 1, \ldots, c$, the i^{th} column of $V^{(r,c)}$ is $(\beta_i^0, \ldots, \beta_i^{r-1})^T$. The idea behind extracting randomness using $V^{(r,c)}$ is as follows: without loss of generality, assume that $r > c$. Moreover, let $(x_1, \ldots, x_r)$ be such that:

1. *Any* c elements of it are completely random and are unknown to adversary $\mathcal{A}_t$.
2. The remaining $r-c$ elements are completely independent of the c elements and also known to $\mathcal{A}_t$.

Now if we compute $(y_1, \ldots, y_c) = (x_1, \ldots, x_r)V$, then $(y_1, \ldots, y_c)$ is a random vector of length c unknown to $\mathcal{A}_t$ [53, 19]. This principle is used in protocol Common-Coin-MB, which is given in Fig. 15.

Let E be an event, defined as follows: All invocations of AVSS scheme in Common-Coin-MB have been terminated properly, with correct outputs. It is easy to see that event E occurs with probability at least $1 - n\epsilon' = 1 - \epsilon$. We now prove the properties of protocol Common-Coin-MB.

Lemma 26 *All honest parties terminate Common-Coin-MB in constant time.*

Proof: We structure the proof in the following way. We first show that assuming every honest party has A-casted "`Reconstruct Enabled`", every honest party will terminate protocol Common-Coin-MB in constant time. Then we show that there exists at least one honest party who will A-cast "`Reconstruct Enabled`". Consequently, we prove that if one honest party A-casts

Fig. 15 Multi-Bit Common Coin Protocol

Protocol Common-Coin-MB(ϵ)

CODE FOR P_i: — All parties execute this code

1. For $j = 1, \ldots, n$, choose a random value x_{ij} and execute AVSS-MS-Share($P_i, \mathcal{P}, (x_{i1,\ldots,x_{in}}), \epsilon'$) where $\epsilon' = \frac{\epsilon}{n}$.
2. Participate in AVSS-MS-Share($P_j, \mathcal{P}, (x_{j1}, \ldots, x_{jn}), \epsilon'$) for every $j \in \{1, \ldots, n\}$. We denote AVSS-MS-Share($P_j, \mathcal{P}, (x_{j1}, \ldots, x_{jn}), \epsilon'$) by AVSS-MS-Share$_j$.
3. Upon terminating AVSS-MS-Share$_j$, A-cast "P_i `terminated` P_j".
4. Create a dynamic set $\mathcal{T}_i$. Add party P_j to $\mathcal{T}_i$ if "P_k `terminated` P_j" is received from the A-cast of at least $n - t$ P_k's. Wait until $|\mathcal{T}_i| = n - t$. Then assign $T_i = \mathcal{T}_i$ and A-cast "`Attach` T_i `to` P_i". We say that the secrets $\{x_{ji} | P_j \in T_i\}$ are the secrets `attached` to party P_i.
5. Create a dynamic set $\mathcal{A}_i$. Add party P_j to $\mathcal{A}_i$ if
   (a) "`Attach` T_j `to` P_j" is received from the A-cast of P_j and
   (b) $T_j \subseteq \mathcal{T}_i$.
   Wait until $|\mathcal{A}_i| = n - t$. Then assign $A_i = \mathcal{A}_i$ and A-cast "P_i `Accepts` A_i".
6. Create a dynamic set $\mathcal{S}_i$. Add party P_j to $\mathcal{S}_i$ if
   (a) "P_j `Accepts` A_j" is received from the A-cast of P_j and
   (b) $A_j \subseteq \mathcal{A}_i$.
   Wait until $|\mathcal{S}_i| = n - t$. Then A-cast "`Reconstruct Enabled`". Let H_i be the current content of $\mathcal{A}_i$. Stop participating in AVSS-MS-Share$_j$ for all P_j who are are not yet included in current $\mathcal{T}_i$. Later resume all such instances of AVSS-MS-Share$_j$'s if P_j is included in $\mathcal{T}_i$.
7. Wait to receive "`Reconstruct Enabled`" from A-cast of at least $n - t$ parties. Participate in AVSS-MS-Rec($P_k, \mathcal{P}, (x_{k1}, \ldots, x_{kn}), \epsilon'$) for every $P_k \in \mathcal{T}_i$. We denote AVSS-MS-Rec($P_k, \mathcal{P}, (x_{k1}, \ldots, x_{kn}), \epsilon'$) by AVSS-MS-Rec$_k$. Notice that as on when new parties are added to $\mathcal{T}_i$, P_i participates in corresponding AVSS-MS-Rec.
8. Let $u = \lceil 0.87n \rceil$. Every party $P_j \in \mathcal{A}_i$ is `associated` with $n - 2t$ values, say $V_{j1}, \ldots, V_{j(n-2t)}$ in the following way. Let x_{kj} for every $P_k \in T_j$ has been reconstructed. Let X_j be the $n - t$ length vector consisting of $\{x_{kj} \mid P_k \in T_j\}$. Then set $(v_{j1}, \ldots, v_{j(n-2t)}) = X_j \cdot V^{(n-t,n-2t)}$, where $V^{(n-t,n-2t)}$ is an $(n - t) \times (n - 2t)$ Vandermonde Matrix. Now $V_{jl} = v_{jl} \mod u$ for $l = 1, \ldots, n - 2t$.
9. Wait until $n - 2t$ values `associated` with all the parties in H_i are computed. Now for every $l = 1, \ldots, n - 2t$ if there exits a party $P_j \in H_i$ such that $V_{jl} = 0$, then set 0 as the l^{th} binary output; otherwise set 1 as the l^{th} binary output. Finally output the $n - 2t$ length binary vector.

"`Reconstruct Enabled`", then eventually every other honest party will do the same.

So let us first prove the first statement. Assuming every honest party has A-casted "`Reconstruct Enabled`", it will hold that eventually every honest party P_i will receive $n - t$ A-casts of "`Reconstruct Enabled`" from $n - t$ honest parties and will invoke AVSS-MS-Rec corresponding to every party in $\mathcal{T}_i$. It clear that a party P_k that is included in $\mathcal{T}_i$ of some honest P_i, will be eventually included in $\mathcal{T}_j$ of every other P_j. Hence if P_i participates in AVSS-MS-Rec$_k$, then eventually every other honest party will do the same. Given event E, all invocations of AVSS-MS-Rec terminate in constant time. Also black box protocol for A-cast terminates in constant time. This proves the first statement.

We next show that there is at least one honest party who will A-cast "`Reconstruct Enabled`". So assume that P_i is the *first honest party* to A-cast "`Reconstruct Enabled`". We will show that this event will always take place. First notice that till P_i A-cast "`Reconstruct Enabled`", no honest party would halt any AVSS-MS-Share$_j$. By the termination property of AVSS-MS-Share, every honest party will eventually terminate the instance of AVSS-MS-Share of every other honest party. Moreover, there are at least $n-t$ honest parties. So from the protocol steps, it is easy to see that for honest P_i, $\mathcal{T}_i$ will eventually contain at least $n - t$ parties and hence P_i will eventually A-cast "`Attach` T_i `to` P_i". Similarly, every other honest P_j will be eventually included in $\mathcal{A}_i$ and so $\mathcal{A}_i$ will eventually contain at least $n - t$ parties and hence P_i will A-cast "P_i `Accepts` A_i". Similarly, $\mathcal{S}_i$ will eventually be of size $n - t$ and hence P_i will A-cast "`Reconstruct Enabled`".

Now we show that every other honest party P_j will also A-cast "`Reconstruct Enabled`" eventually. It is easy to see that every party that is included in $\mathcal{T}_i$ will also be included in $\mathcal{T}_j$ eventually. And hence, all the conditions that are satisfied for honest P_i above will be eventually satisfied for every other honest P_j. So P_j will eventually A-cast "`Reconstruct Enabled`". □

We now prove the following important lemma, which is at the heart of Common-Coin-MB. The lemma shows that the adversary behavior of Fig. 14 is not applicable in Common-Coin-MB.

Lemma 27 *Let a corrupted party P_k is included in $\mathcal{T}_j$ of an honest P_j in protocol Common-Coin-MB. Then the values shared by P_k in AVSS-MS-Share$_k$ are completely independent of the values shared by the honest parties.*

PROOF: Let P_i be the *first honest* party who receives A-cast of "`Reconstruct Enabled`" from at least $n - t$ parties and starts participating in AVSS-MS-Rec, corresponding to each party in $\mathcal{T}_i$. To prove the lemma, we first assert that a *corrupted* party P_k will never be included in $\mathcal{T}_j$ of *any* honest P_j, if P_k invokes AVSS-MS-Share$_k$ *only after* P_i starts participating in AVSS-MS-Rec corresponding to each party in $\mathcal{T}_i$. We prove this by contradiction. Let P_i has received "`Reconstruct Enabled`" from the parties in $\mathcal{B}_1$ with $|\mathcal{B}_1| \geq n - t$. Moreover, assume P_k invokes AVSS-MS-Share$_k$ only after P_i received "`Reconstruct Enabled`" from the parties in $\mathcal{B}_1$ and starts participating in AVSS-MS-Rec corresponding to each party in $\mathcal{T}_i$. Furthermore, assume that P_k is still in $\mathcal{T}_j$ of some honest P_j. Now $P_k \in \mathcal{T}_j$ implies that P_j must have received "P_m `terminated` P_k" from A-cast of at least $n - t$ P_m's, say $\mathcal{B}_2$. Now $|\mathcal{B}_1 \cap \mathcal{B}_2| \geq n - 2t$ and thus the intersection set contains at least one honest party, say P_α, as $n = 3t + 1$. This implies that honest $P_\alpha \in \mathcal{B}_1$ and must have terminated AVSS-MS-Share$_k$ before A-casting "`Reconstruct Enabled`". Otherwise P_α would have halted the execution of AVSS-MS-Share$_k$ and would never A-cast "P_α `terminated` P_k" (see step 6 in the protocol). This further implies that P_k must have invoked AVSS-MS-Share$_k$ before P_i starts participating in AVSS-MS-Recs. But this is a contradiction to our assumption.

Hence if the corrupted P_k is included in $\mathcal{T}_j$ of *any* honest P_j then he must have invoked AVSS-MS-Share$_k$ before any AVSS-MS-Rec has been invoked by any honest party. Thus P_k will have no knowledge of the secrets shared by honest parties when he chooses his own secrets for AVSS-MS-Share$_k$. □

Lemma 28 *In protocol Common-Coin-MB, once some honest party P_j receives "`Attach` T_i `to` P_i" from the A-cast of P_i and includes P_i in $\mathcal{A}_j$, $n-2t$ unique values $V_{i1}, \ldots, V_{i(n-2t)}$ are fixed such that*

1. *Every honest party will `associate` $V_{i1}, \ldots, V_{i(n-2t)}$ with P_i, except with probability ϵ.*
2. *Each of $V_{i1}, \ldots, V_{i(n-2t)}$ is distributed uniformly over $[0, \ldots, u]$ and independent of the values `associated` with the other parties.*

Proof: The values $V_{i1}, \ldots, V_{i(n-2t)}$ are defined in step 8 of the protocol. We now prove the first part of the lemma. According to the lemma condition, $P_i \in \mathcal{A}_j$. This implies that $T_i \subseteq \mathcal{T}_j$. So honest P_j will participate in AVSS-MS-Rec$_k$ corresponding to each $P_k \in T_i$. Moreover, eventually $T_i \subseteq \mathcal{T}_m$ and $P_i \in \mathcal{A}_m$ will hold for *every other honest* P_m. So, every other honest party will also participate in AVSS-MS-Rec$_k$ corresponding to each $P_k \in T_i$. Now by the property of AVSS-MS-Rec, each honest party will reconstruct x_{ki} at the completion of AVSS-MS-Rec$_k$, except with probability ϵ'. Thus, with probability $1-(n-t)\epsilon' \approx 1-\epsilon$, every honest party will associate $V_{i1}, \ldots, V_{i(n-2t)}$ with P_i.

We now prove second part of the lemma. By Lemma 27, when T_i is fixed, the values that are shared by corrupted parties in T_i are completely independent of the values shared by the honest parties in T_i. Now, each T_i contains at least $n - 2t$ honest parties and every honest partys' shared secrets are uniformly distributed and mutually independent. Hence by the property of Vandermonde matrix the values $v_{i1}, \ldots, v_{i(n-2t)}$ are completely random and thus $V_{i1}, \ldots, V_{i(n-2t)}$ are uniformly and independently distributed over $[0, \ldots, u]$. □

Lemma 29 *In protocol Common-Coin-MB, once an honest party A-casts "`Reconstruct Enabled`", there exists a set M of size $|M| \geq \frac{n}{3}$, such that:*

1. *For every party $P_j \in M$, some honest party has received "`Attach` T_j `to` P_j" from the A-cast of P_j.*
2. *When any honest party P_j A-casts "`Reconstruct Enabled`", then it will hold that $M \subseteq H_j$.*

Proof: Follows from the proof of Lemma 15 □

Lemma 30 *Let $\epsilon \leq 0.2$ and assume that all honest parties have terminated protocol Common-Coin-MB. Then for every $l \in \{1, \ldots, n-2t\}$, all honest parties output σ_l with probability at least $\frac{1}{4}$ for every value of $\sigma_l \in \{0, 1\}$.*

Proof: Follows from Lemma 28 and similar arguments as given in the proof of Lemma 16. □

Theorem 10 *Common-Coin-MB is a $(1-\epsilon)$-terminating, t-resilient multi-bit common coin protocol with $t+1$ bits output for every $0 < \epsilon \leq 0.2$.*

Proof: Follows from Lemma 26, 27, 28, 29 and 30. □

Theorem 11 *Protocol Common-Coin-MB privately communicates $\mathcal{O}(n^5 \log \frac{1}{\epsilon})$ bits and A-cast $\mathcal{O}(n^5 \log \frac{1}{\epsilon})$ bits for $(t + 1) = \Theta(n)$ bit output.*

Proof: Easy, as n instances of AVSS-MS-Share and AVSS-MS-Rec with $\ell = n$ secrets are executed. □

From Theorem 11, we get the following corollary.

Corollary 1 *The amortized communication cost of generating a single bit output in Common-Coin-MB is $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits of private communication and $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits of A-cast communication.*

The above corollary shows that the amortized communication complexity of generating single bit output in Common-Coin-MB is $\mathcal{O}(n^2)$ times better than Common-Coin. In the next section, we use Common-Coin-MB to design an ABA protocol which allows the parties to reach agreement on $t + 1$ bits *concurrently.*

7.5 ABA Protocol for Agreement on $t + 1$ Bits

We now design protocol ABA-MB, which attains agreement on $n - 2t = t + 1$ bits concurrently. So initially every party has a private input of $n - 2t$ bits. Let the $n - 2t$ bit input of P_i be denoted by $x_{i1}, \ldots, x_{i(n-2t)}$.

The Intuition: The high level idea of ABA-MB is similar to ABA (given in Section 6). The ABA protocol proceeds in iterations where in each iteration every party computes his 'modified input', consisting of $n-2t$ bits.

In the first iteration the 'modified input' of P_i is the private input bits of P_i. In *each* iteration, every party executes the following protocols *sequentially*:

1. $n-2t$ parallel instances of Vote protocol, one corresponding to each bit of the 'modified input';
2. A *single* instance of Common-Coin-MB.

Notice that the parties participate in Common-Coin-MB, only after terminating all the $n-2t$ instances of Vote protocol. Now the parties parallely perform *almost* similar computation as in protocol ABA, corresponding to each of the $t+1$ bits. However, instead of executing $n-2t$ instances of Common-Coin protocol, the parties execute *only a single instance* of Common-Coin-MB. The protocol is given in Fig. 16. We now prove the properties of protocol ABA-MB.

Fig. 16 ABA Protocol for Agreement on $t+1$ Bits

Protocol ABA-MB(ϵ)

CODE FOR P_i: — Every party executes this code

1. Set $r := 0$. For $l = 1, \ldots, n-2t$, set $v_{1l} = x_{il}$.
2. Repeat until terminating.
   (a) Set $r := r+1$. Participate in $n-2t$ instances of Vote protocol, with v_{rl} as the input in the l^{th} instance of Vote protocol, for $l = 1, \ldots, n-2t$. Set (y_{rl}, m_{rl}) as the output of the l^{th} instance of Vote protocol.
   (b) Wait to terminate all the $n-2t$ instances of Vote protocol. Then invoke Common-Coin-MB($\frac{\epsilon}{4}$) and wait until its termination. Let $c_{r1}, \ldots, c_{r(n-2t)}$ be the output of Common-Coin-MB.
   (c) For every $l \in \{1, \ldots, n-2t\}$ such that agreement on l^{th} bit is not achieved, parallely do the following:
      i. If $m_{rl} = 2$, then set $v_{(r+1)l} = y_{rl}$ and A-cast ("Terminate with $v_{(r+1)l}$", l). Participate in only one more instance of Vote corresponding to l^{th} bit with $v_{(r+1)l}$ as the input. Participate in only one more instance of Common-Coin-MB if ("Terminate with $v_{(r+1)l}$", l) is A-casted for *all* $l = 1, \ldots, n-2t$.
      ii. If $m_{rl} = 1$, set $v_{(r+1)l} = y_{rl}$.
      iii. Otherwise, set $v_{(r+1)l} = c_{rl}$.
   (d) Upon receiving ("Terminate with σ_l", l) from the A-cast of at least $t+1$ parties, for some value σ_l, output σ_l as the l^{th} bit and terminate all the computation regarding l^{th} bit. In this case, we say that agreement on l^{th} bit is achieved.
   (e) Terminate ABA-MB when agreement is achieved on all l bits, for $l = 1, \ldots, n-2t$.

Lemma 31 *In protocol ABA-MB, if all the honest parties have input $\sigma_1, \ldots, \sigma_{n-2t}$, then all the honest parties terminate and output $\sigma_1, \ldots, \sigma_{n-2t}$.*

PROOF: Directly follows from Lemma 21 and protocol steps. □

Lemma 32 *If some honest party terminates protocol ABA-MB with output $\sigma_1, \ldots, \sigma_{n-2t}$, then all honest parties will eventually terminate ABA-MB with output σ_1, $\ldots, \sigma_{n-2t}$.*

PROOF: To prove the lemma, it is enough to show that for every $l = 1, \ldots, n-2t$, if an honest party terminates ABA-MB with output σ_l, then all honest parties will eventually terminate ABA-MB with output σ_l. However, this follows from the proof of Lemma 22. □

Lemma 33 *If all honest parties have initiated and completed some iteration k, then with probability at least $\frac{1}{4}$, all honest parties will have same value for 'modified input' $v_{(k+1)l}$, for every $l = 1, \ldots, n-2t$.*

PROOF: Follows from the proof of Lemma 23. □

We now recall event C_k and C from section 6. Let C_k be the event that each honest party completes all the iterations he initiated up to (and including) the k^{th} iteration (that is, for each iteration $1 \leq r \leq k$ and for each party P, if P initiated iteration r then he computes $v_{(r+1)l}$ for every l^{th} bit). Let C denote the event that C_k occurs for all k.

Lemma 34 *Conditioned on event C, all honest parties terminate protocol ABA-MB in constant expected time.*

PROOF: Let the *first* instance of A-cast of ("Terminate with σ_l", l) is initiated by some honest party in iteration τ_l. Following Lemma 22, every other honest party will A-cast ("Terminate with σ_l", l) in iteration $\tau_l + 1$. Now it is true that agreement on l^{th} bit will be achieved within constant time after $(\tau_l+1)^{th}$ iteration (this is because the A-casts can be completed in constant time). Let m be such that τ_m is the maximum among $\tau_1, \ldots, \tau_{n-2t}$. We first show that all honest parties will terminate protocol ABA-MB within constant time after some honest party initiates the first instance of A-cast ("Terminate with σ_m", m). Since the first instance of A-cast of ("Terminate with σ_m", m) is initiated by some honest party in iteration τ_m, all the parties will participate in Vote and Common-Coin-MB in iteration $\tau_m + 1$. Both the executions can be completed in constant time. Moreover, by Lemma 22 every honest party will A-cast ("Terminate with σ_m", m) by the end of iteration $\tau_m + 1$. The A-casts can be completed in constant time. Moreover, it is to be noted that for all other bits l, agreement will be reached either before reaching agreement on m^{th} bit or within constant time of reaching agreement on m^{th} bit. Hence all honest parties will terminate ABA-MB within constant time after the *first* instance of A-cast of ("Terminate with σ_m", m) is initiated by some honest party in iteration τ_m.

Now conditioned on event C, all honest parties terminate each iteration in constant time. So it is left to show that $E(\tau_m|C)$ is constant. We have

$$Prob(\tau_m > k|C_k) \leq Prob(\tau_m \neq 1|C_k) \times \ldots \times Prob(\tau_m \neq k \cap \ldots \cap \tau_m \neq 1|C_k)$$

From the Lemma 33, it follows that each one of the k multiplicands of the right hand side of the above equation is at most $\frac{3}{4}$. Thus we have $Prob(\tau_m > k|C_k) \leq (\frac{3}{4})^k$. Now simple calculation gives $E(\tau_m|C) \leq 16$. □

Lemma 35 *$Prob(C) \geq (1 - \epsilon)$.*

PROOF: Follows from the proof of Lemma 25. □

Summing up, we have the following theorem.

Theorem 12 (ABA for $t + 1$ Bits) *Let $n = 3t + 1$. Then for every $0 < \epsilon \leq 0.2$, protocol ABA-MB is a t-resilient, $(\epsilon, 0)$-ABA protocol for n parties. Given the parties terminate, they do so in constant expected time. The protocol allows the parties to reach agreement on $t + 1$ bits simultaneously and involves private communication and A-cast of $\mathcal{O}(n^5 \log \frac{1}{\epsilon})$ bits.*

8 Conclusion and Open Problems

We have presented a novel, constant expected time, optimally resilient, $(\epsilon, 0)$-ABA protocol whose communication complexity is significantly better than best known existing ABA protocols of [15,1] (though the ABA protocol of [1] has a strong property of being *almost surely terminating*) with optimal resilience. Here we summarize the key factors that have contributed to the gain in the communication complexity of our ABA protocol: (a) A shorter route: $ICP \rightarrow AWSS \rightarrow AVSS \rightarrow ABA$, (b) Improving each of the building blocks by introducing new techniques and (c) By exploiting the advantages of dealing with multiple secrets concurrently in each of these blocks. It is to be mentioned that our new AVSS scheme significantly outperforms the existing AVSS schemes in the same settings in terms of communication complexity. An interesting open problem is to further improve the communication complexity of ABA protocols. Also one can try to provide an *almost surely terminating*, optimally resilient, constant expected time ABA protocol whose communication complexity is less than the ABA protocol of [1].

References

1. I. Abraham, D. Dolev, and J. Y. Halpern. An almost-surely terminating polynomial protocol for asynchronous Byzantine Agreement with optimal resilience. In R. A. Bazzi and B. Patt-Shamir, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC 2008, Toronto, Canada, August 18-21, 2008*, pages 405–414. ACM Press, 2008.
2. B. Altmann, M. Fitzi, and U. M. Maurer. Byzantine Agreement secure against general adversaries in the dual failure model. In P. Jayanti, editor, *Distributed Computing, 13th International Symposium, Bratislava, Slavak Republic, September 27-29, 1999, Proceedings*, volume 1693 of *Lecture Notes in Computer Science*, pages 123–137. Springer Verlag, 1999.
3. Z. Beerliová-Trubíniová and M. Hirt. Simple and efficient perfectly-secure asynchronous MPC. In K. Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 376–392. Springer Verlag, 2007.
4. M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the Second Annual ACM SIGACT-SIGOPS Symposium on Princiles of Distributed Computing, August 17-19, 1983, Montreal, Quebec, Canada*, pages 27–30. ACM Press, 1983.
5. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM Press, 1988.
6. M. BenOr, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience. In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, California, USA, August 14-17*, pages 183–192. ACM Press, 1994.
7. P. Berman and J. A. Garay. Asymptotically optimal distributed consensus (extended abstract). In G. Ausiello, M. Dezani-Ciancaglini, and S. R. D. Rocca, editors, *Automata, Languages and Programming, 16th International Colloquium, ICALP89, Stresa, Italy, July 11-15, 1989, Proceedings*, volume 372 of *Lecture Notes in Computer Science*, pages 80–94. Springer, 1989.
8. P. Berman and J. A. Garay. Cloture votes: n/4-resilient distributed consensus in t+1 rounds. *Mathematical Systems Theory*, 26(1):3–19, 1993.
9. P. Berman, J. A. Garay, and K. J. Perry. Towards optimal distributed consensus (extended abstract). In *Proceedings of 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, 30 October - 1 November 1989*, pages 410–415. IEEE Computer Society, 1989.
10. G. Bracha. An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. In *Proceedings of the Third Annual ACM Symposium on Princiles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 154 – 162. ACM Press, 1984.
11. G. Bracha. Asynchronous Byzantine Agreement protocols. *Inf and Computation*, 75(2):130–143, 1987.
12. G. Bracha. An $O(\log n)$ expected rounds randomized Byzantine generals protocol. *J. ACM*, 34(4):910–920, 1987.
13. G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985.

14. R. Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute, Israel, 1995.
15. R. Canetti and T. Rabin. Fast asynchronous Byzantine Agreement with optimal resilience. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 42–51. ACM Press, 1993.
16. B. A. Coan and J. L. Welch. Modular construction of a Byzantine Agreement protocol with optimal message bit complexity. *Information and Computation*, 97(1):61–85, 1992.
17. R. Cramer and I. Damgård. *Multiparty Computation, an Introduction*. Contemporary Cryptography. Birkhuser Basel, 2005.
18. R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In J. Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 311–326. Springer Verlag, 1999.
19. I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In A. Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer Verlag, 2007.
20. D. Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982.
21. D. Dolev, M. J. Fischer, R. J. Fowler, N. A. Lynch, and H. R. Strong. An efficient algorithm for Byzantine Agreement without authentication. *Information and Control*, 52(3):257–274, 1982.
22. D. Dolev and R. Reischuk. Bounds on information exchange for Byzantine Agreement. *Journal of ACM*, 32(1):191–204, 1985.
23. D. Dolev, R. Reischuk, and H. R. Strong. Early stopping in Byzantine Agreement. *Journal of ACM*, 37(4):720–741, 1990.
24. D. Dolev and H. R. Strong. Polynomial algorithms for multiple processor agreement. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 401–407. ACM Press, 1982.
25. D. Dolev and H. R. Strong. Authenticated algorithms for Byzantine Agreement. *SIAM Journal of Computing*, 12(4):656–666, 1983.
26. P. Feldman and S. Micali. Byzantine Agreement in constant expected time (and trusting no one). In *Proceedings of 26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, 21-23 October 1985*, pages 267–276. IEEE Computer Society, 1985.
27. P. Feldman and S. Micali. An optimal algorithm for synchronous Byzantine Agreemet. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 639–648. ACM Press, 1988.
28. P. Feldman and S. Micali. An optimal probabilistic protocol for synchronous Byzantine Agreement. *SIAM Journal of Computing*, 26(4):873–933, 1997.
29. M. J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *FCT*, pages 127–140, 1983.
30. M. J. Fischer, N. A. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. In *Fault-Tolerant Distributed Computing*, pages 147–170, 1986.
31. M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *JACM*, 32(2):374–382, 1985.
32. M. Fitzi. *Generalized Communication and Security Models in Byzantine Agreement*. PhD thesis, ETH Zurich, 2002.
33. M. Fitzi, J. Garay, S. Gollakota, C. Pandu Rangan, and K. Srinathan. Round-optimal and efficient verifiable secret sharing. In S. Halevi and T. Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 329–342. Springer Verlag, 2006.
34. M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein, and A. Smith. Detectable Byzantine Agreement secure against faulty majorities. In *PODC 2002, Proceedings of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing, July 21-24, 2002 Monterey, California, USA*, pages 118–126. ACM Press, 2002.
35. M. Fitzi and M. Hirt. Optimally efficient multi-valued Byzantine Agreement. In Ruppert E and Malkhi D, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006*, pages 163–168, 2006.
36. M. Fitzi and U. M. Maurer. From partial consistency to global broadcast. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 494–503. ACM Press, 2000.
37. Z. Galil, A. J. Mayer, and M. Yung. Resolving message complexity of Byzantine Agreement and beyond. In *Proceedings of 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995*, pages 724–733. IEEE Computer Society, 1995.
38. J. A. Garay and K. J. Perry. A continuum of failure models for distributed computing. In A. Segall and S. Zaks, editors, *Distributed Algorithms, 6th International Workshop, WDAG '92, Haifa, Israel, November 2-4, 1992, Proceedings*, volume 647 of *Lecture Notes in Computer Science*, pages 153–165. Springer Verlag, 1992.
39. R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin. The round complexity of verifiable secret sharing and secure multicast. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece. ACM*, pages 580–589. ACM Press, 2001.
40. J. Katz, C. Koo, and R. Kumaresan. Improving the round complexity of VSS in point-to-point networks. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 499–510. Springer Verlag, 2008.
41. J. Katz and C. Y. Koo. On expected constant-round protocols for Byzantine Agreement. In C. Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, Lecture Notes in Computer Science, pages 445–462. Springer Verlag, 2006.
42. L. Lamport. The weak Byzantine generals problem. *Journal of ACM*, 30(3):668–676, 1983.
43. Y. Lindell, A. Lysyanskaya, and T. Rabin. On the composition of authenticated Byzantine Agreement. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montral, Qubec, Canada*, pages 514–523. ACM Press, 2002.

44. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
45. A. Patra, A. Choudhary, T. Rabin, and C. Pandu Rangan. The round complexity of verifiable secret sharing revisited. In S. Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 487–504. Springer Verlag, 2009.
46. A. Patra, A. Choudhary, and C. Pandu Rangan. Efficient asynchronous multiparty computation with optimal resilience. Cryptology ePrint Archive, Report 2008/425, 2008.
47. A. Patra, A. Choudhary, and C. Pandu Rangan. Simple and efficient asynchronous Byzantine Agreement with optimal resilience. In S. Tirthapura and L. Alvisi, editors, *Proceedings of the 28th Annual ACM Symposium on Principles of Distributed Computing, PODC 2009, Calgary, Alberta, Canada, August 10-12, 2009*, pages 92–101. ACM Press, 2009.
48. M. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *JACM*, 27(2):228–234, 1980.
49. B. Pfitzmann and M. Waidner. Unconditional Byzantine Agreement for any number of faulty processors. In A. Finkel and M. Jantzen, editors, *STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, February 13-15, 1992, Proceedings*, volume 577 of *Lecture Notes in Computer Science*, pages 339–350. Springer Verlag, 1992.
50. M. O. Rabin. Randomized Byzantine generals. In *34th Annual Symposium on Foundations of Computer Science, Palo Alto California, 3-5 November 1993*, pages 403–409. IEEE Computer Society, 1983.
51. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 73–85. ACM Press, 1989.
52. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
53. K. Srinathan, A. Narayanan, and C. Pandu Rangan. Optimal perfectly secure message transmission. In M. K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 545–561. Springer Verlag, 2004.
54. S. Toueg. Randomized Byzantine Agreements. In *Proceedings of the Third Annual ACM Symposium on Princiles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 163–178. ACM Press, 1984.
55. S. Toueg, K. J. Perry, and T. K. Srikanth. Fast distributed agreement. *SIAM Journal of Computing*, 16(3):445–457, 1987.
56. R. Turpin and B. A. Coan. Extending binary Byzantine Agreement to multivalued Byzantine Agreement. *Information Processing Letters*, 18(2):73–76, 1984.

APPENDIX A: Analysis of the Communication Complexity of the AVSS, ABA Scheme of [15]

The communication complexity analysis of the AVSS and ABA protocol of [15] was not reported anywhere so far. So we have carried out the same at this juncture. To do so, we have considered the detailed description of the AVSS protocol of [15] given in Canetti's Thesis [14]. To bound the error probability by ϵ, all the communication and computation in the protocol of [15] is done over a finite field $\mathbb{F}$, where $|\mathbb{F}| = GF(2^\kappa)$ and $\epsilon = 2^{-\Omega(\kappa)}$. Thus each field element can be represented by $\mathcal{O}(\kappa) = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

To begin with, in the ICP protocol of [15], D gives $\mathcal{O}(\kappa)$ field elements to INT and $\mathcal{O}(\kappa)$ field elements to verifier R. Though the ICP protocol of [14] is presented with a *single* verifier, it is executed with n verifiers in protocol A-RS. In order to execute ICP with n verifiers, D gives $\mathcal{O}(n\kappa)$ field elements to INT and $\mathcal{O}(\kappa)$ field elements to each of the n verifiers. So the communication complexity of ICP of [14] when executed with n verifiers is $\mathcal{O}(n\kappa)$ field elements and hence $\mathcal{O}(n\kappa^2)$ bits.

Now by incorporating their ICP protocol with n verifiers in Shamir secret sharing [52], the authors in [15] designed an asynchronous primitive called A-RS, which consists of two sub-protocols, namely A-RS-Share and A-RS-Rec. In the A-RS-Share protocol, D generates n shares (Shamir shares) of a secret s and for each of the n shares, D executes an instance of ICP protocol with n verifiers. So the A-RS-Share protocol of [15] involves a private communication of $\mathcal{O}(n^2\kappa^2)$ bits. In addition to this, the A-RS-Share protocol also involves an A-cast of $\mathcal{O}(\log(n))$ bits. In the A-RS-Rec protocol, the IC signatures given by D in A-RS-Share are revealed, which involves a private communication of $\mathcal{O}(n^2\kappa^2)$ bits. In addition, the A-RS-Rec protocol involves A-cast of $\mathcal{O}(n^2 \log(n))$ bits.

Proceeding further, by incorporating their A-RS protocol, the authors in [15] designed an AWSS scheme. The AWSS protocol consists of two sub-protocols, namely AWSS-Share and AWSS-Rec. In the AWSS-Share protocol, D generates n shares (Shamir shares [52]) of the secret and instantiate n instances of the ICP protocol for each of the n shares. Now each individual party A-RS-Share all the values that it has received in the n instances of the ICP protocol. Since each individual party receives a total of $\mathcal{O}(n\kappa)$ field elements in the n instances of ICP, the above step incurs a private communication of $\mathcal{O}(n^4\kappa^3)$ bits and A-cast of $\mathcal{O}(n^2\kappa \log(n))$ bits. In the AWSS-Rec protocol, each party P_i tries to reconstruct the values which are A-RS-Shared by each party P_j in a set $\mathcal{E}_i$. Here $\mathcal{E}_i$ is a set which is defined in the AWSS-Share protocol. In the worst case, the size of each $\mathcal{E}_i$ is $\mathcal{O}(n)$. So in the worst case, the AWSS-Rec protocol privately communicates $\mathcal{O}(n^5\kappa^3)$ bits and A-cast $\mathcal{O}(n^5\kappa \log(n))$ bits.

The authors in [15] then further extended their AWSS-Share protocol to Two&Sum AWSS-Share protocol, where each party P_i has to A-RS-Share $\mathcal{O}(n\kappa^2)$ field elements.

So the communication complexity of Two&Sum AWSS-Share is $\mathcal{O}(n^4\kappa^4)$ bits and A-cast of $\mathcal{O}(n^2\kappa^2\log(n))$ bits.

Finally using their Two&Sum AWSS-Share and AWSS-Rec protocol, the authors in [15] have deigned their AVSS scheme, which consists of two sub-protocols, namely AVSS-Share and AVSS-Rec. In the AVSS-Share protocol, the most communication expensive step is the one where each party has to AWSS-Rec $\mathcal{O}(n^3\kappa)$ field elements. So in total, the AVSS-Share protocol of [15] involves a communication complexity of $\mathcal{O}(n^9\kappa^4)$ bits and A-cast $\mathcal{O}(n^9\kappa^2\log(n))$ bits. The AVSS-Rec protocol involves n instances of AWSS-Rec, resulting in a communication complexity of $\mathcal{O}(n^6\kappa^3)$ bits and A-cast of $\mathcal{O}(n^6\kappa\log(n))$ bits.

Now in the *common coin* protocol, each party in $\mathcal{P}$ acts as a dealer and invokes n instances of AVSS-Share to share n secrets. So the communication complexity of the common protocol of [15] is $\mathcal{O}(n^{11}\kappa^4)$ bits of private communication and $\mathcal{O}(n^{11}\kappa^2\log(n))$ bits of A-cast. Now in the ABA protocol of [15], AVSS-Share protocol is called for $\mathcal{C} = \mathcal{O}(1)$ expected time. Hence the ABA protocol of [15] involves a private communication of $\mathcal{O}(n^{11}\kappa^4)$ bits and A-cast of $\mathcal{O}(n^{11}\kappa^2\log(n))$ bits. As mentioned earlier, $\mathcal{O}(\kappa) = \mathcal{O}(\log\frac{1}{\epsilon})$. Thus the ABA protocol of [15] involves a private communication of $\mathcal{O}(n^{11}\log(\frac{1}{\epsilon})^4)$ bits and A-cast of $\mathcal{O}(n^{11}\log(\frac{1}{\epsilon})^2\log(n))$ bits.

APPENDIX B: Proof for Protocol Common-Coin

Lemma 13 [14] *All honest parties terminate Protocol Common-Coin in constant time.*

PROOF: First we show that every *honest* party P_i will A-cast "Reconstruct Enabled". By the termination property of our AVSS scheme, every honest party will eventually terminate *all* the n instances of AVSS-Share of every other honest party. As there are at least $2t+1$ honest parties, it implies that $\mathcal{T}_i$ of every *honest* P_i will eventually contain at least $2t+1$ honest parties. Also from termination property of AVSS protocol, eventually $T_j \subseteq \mathcal{T}_i$ will hold good for every *honest* P_j, P_i. So for every honest P_i, $\mathcal{A}_i$ will eventually be of size $2t+1$ and similarly $\mathcal{S}_i$ will eventually be of size $2t+1$ and hence P_i will A-cast "Reconstruct Enabled".

Now it remains to show that AVSS-Rec protocols invoked by any honest party will be terminated eventually. Once this is proved, every honest party will terminate protocol Common-Coin after executing the remaining steps of Common-Coin such as computing V_i etc. By the properties of our AVSS scheme, if an honest party P_i receives "Attach T_j to P_j" from P_j and includes P_j in $\mathcal{A}_i$, then eventually every other honest party will do the same. Hence if P_i invokes AVSS-Rec$_{kj}$ for $P_j \in \mathcal{A}_i$ and $P_k \in T_j$, then eventually every other honest party will also do the same. Now by the termination property of AVSS protocol, every AVSS-Rec$_{kj}$ protocols will be terminated by every honest party.

Given event E, all invocations of AVSS-Share and AVSS-Rec terminate in constant time. The black box protocol for A-cast terminates in constant time. Thus protocol Common-Coin terminates in constant time. □

Lemma 14 [14] *In Common-Coin, once some honest P_j receives "Attach T_i to P_i" from A-cast of P_i and includes P_i in $\mathcal{A}_j$, a unique value V_i is fixed such that*

1. *Every honest party will associate V_i with P_i, except with probability $1-\frac{\epsilon}{n}$.*
2. *V_i is distributed uniformly over $[0,\ldots,u]$ and independent of values associated with other parties.*

PROOF: Once some *honest* P_j receives "Attach T_i to P_i" from A-cast of P_i and includes P_i in $\mathcal{A}_j$, a unique value V_i is fixed. Here $V_i = (\sum_{P_k\in T_i} x_{ki})\ mod\ u$, where x_{ki} is shared by P_k as a dealer during AVSS-Share$_{ki}$. According to the protocol steps eventually all honest parties will invoke AVSS-Rec$_{ki}$ corresponding to each $P_k \in T_i$ and consequently each honest party will reconstruct x_{ki} at the completion of AVSS-Rec$_{ki}$, except with probability ϵ'. Now since $|T_i| = t+1$, every honest party will associate V_i with P_i with probability at least $1-(t+1)\epsilon' \approx 1-\frac{\epsilon}{n}$.

An honest party starts invoking AVSS-Rec$_{ki}$ for every $P_k \in T_i$ only after it receives "Attach T_i to P_i" from A-cast of P_i. So the set T_i is fixed before any honest party invokes AVSS-Rec$_{ki}$ for some k. The secrecy property of AVSS-Share ensures that corrupted parties will have no information about the value shared by any honest party until the value is reconstructed after executing corresponding AVSS-Rec. Thus when T_i is fixed, the values that are shared by corrupted parties corresponding to P_i are completely independent of the values shared by the honest parties corresponding to P_i. Now, each T_i contains at least one honest party and every honest party's shared secrets are uniformly distributed and mutually independent. Hence the sum V_i is uniformly and independently distributed over $[0,\ldots,u]$. □

Lemma 15 [14] *Once an honest party A-cast "Reconstruct Enabled", there exists a set M such that:*

1. *For every party $P_j \in M$, some honest party has received "Attach T_j to P_j" from the A-cast of P_j.*
2. *When any honest party P_j A-casts "Reconstruct Enabled", then it will hold that $M \subseteq H_j$.*
3. *$|M| \geq \frac{n}{3}$.*

PROOF: Let P_i be the *first honest party* to A-cast "`Reconstruct Enabled`". Then let $M = \{P_k \mid P_k$ belongs to $A_l's$ of at least $t+1$ $P_l's$ who belongs to $\mathcal{S}_i$ when P_i A-casted `Reconstruct Enabled` $\}$. It is clear that $M \subseteq H_i$. Thus party P_i has received "`Attach` T_j `to` P_j" from the A-cast of every $P_j \in M$. So this proves the first part of the lemma.

An *honest* P_j A-casts "`Reconstruct Enabled`" only when $\mathcal{S}_j$ contains $2t+1$ parties. Now note that $P_k \in M$ implies that P_k belongs to A_l's of at least $t+1$ P_l's who belong to $\mathcal{S}_i$. This ensures that there is at least one such P_l who belongs to $\mathcal{S}_j$, as well as $\mathcal{S}_i$. Now $P_l \in \mathcal{S}_j$ implies that P_j had ensured that $A_l \subseteq \mathcal{A}_j$. This implies that $P_k \in M$ belongs to $\mathcal{A}_j$ before party P_j A-casted "`Reconstruct Enabled`". Since H_j is the instance of $\mathcal{A}_j$ at the time when P_j A-casts "`Reconstruct Enabled`", it is obvious that $P_k \in M$ belongs to H_j also. Using similar argument, it can be shown that *every* $P_k \in M$ also belong to H_j, thus proving second part of the lemma.

To prove the third part of the lemma, we use counting argument. Let $m = |\mathcal{S}_i|$ at the time P_i A-casted "`Reconstruct Enabled`". So we have $m \geq 2t+1$. Now consider an $n \times n$ table Λ_i (relative to party P_i), whose l^{th} row and k^{th} column contains 1 for $k, l \in \{1, \ldots, n\}$ iff the following hold: (a) P_i has received "P_l `Accepts` A_l" from A-cast of P_l and included P_l in $\mathcal{S}_i$ before A-casting "`Reconstruct Enabled`" AND (b)$P_k \in A_l$. The remaining entries (if any) of Λ_i are left blank. Then M is the set of parties P_k such that k^{th} column in Λ_i contains 1 at least at $t+1$ positions. Notice that each row of Λ_i contains 1 at $n-t$ positions. Thus Λ_i contains 1 at $m(n-t)$ positions. Let q denote the minimum number of columns in Λ_i that contain 1 at least at $t+1$ positions. We will show that $q \geq \frac{n}{3}$. The worst distribution of 1 entries in Λ_i is letting q columns to contain all 1 entries and letting each of the remaining $n-q$ columns to contain 1 at t locations. This distribution requires Λ_i to contain 1 at no more than $qm + (n-q)t$ positions. But we have already shown that Λ_i contains 1 at $m(n-t)$ positions. So we have

$$qm + (n-q)t \geq m(n-t).$$

This gives $q \geq \frac{m(n-t)-nt}{m-t}$. Since $m \geq n-t$ and $n \geq 3t+1$, we have

$$\begin{aligned} q &\geq \frac{m(n-t)-nt}{m-t} \geq \frac{(n-t)^2 - nt}{n-2t} \\ &\geq \frac{(n-2t)^2 + nt - 3t^2}{n-2t} \geq n - 2t + \frac{nt - 3t^2}{n-2t} \\ &\geq n - 2t + \frac{t}{n-2t} \geq \frac{n}{3} \end{aligned}$$

This shows that $|M| = q \geq \frac{n}{3}$ □

Lemma 16[14] *Let $\epsilon \leq 0.2$ and assume that all the honest parties have terminated protocol Common-Coin. Then for every value $\sigma \in \{0,1\}$, with probability at least $\frac{1}{4}$, all the honest parties output σ.*

PROOF: By Lemma 14, for every P_i that is included in $\mathcal{A}_j$ of some *honest* P_j, there exists some fixed (yet unknown) value V_i that is distributed uniformly and independently over $[0, \ldots, u]$ and with probability $1 - \frac{\epsilon}{n}$ all honest parties will associate V_i with P_i. Consequently, with probability at least $(1-\epsilon)$, all honest parties will agree on the value associated with every party. Now we consider two cases:

- We now show that the probability of outputting $\sigma = 0$ by all honest parties is at least $\frac{1}{4}$. Let M be the set of parties discussed in Lemma 15. Clearly if $V_j = 0$ for some $P_j \in M$ and all honest parties associate V_j with P_j, then all the honest parties will output 0. The probability that for at least one party $P_j \in M$, $V_j = 0$ is $1 - (1 - \frac{1}{u})^{|M|}$. Now $u = \lceil 0.87n \rceil$. Also $|M| \geq \frac{n}{3}$. Therefore for all $n > 4$, we have $1 - (1 - \frac{1}{u})^{|M|} \geq 0.316$. So, `Prob`(all honest parties output 0) $\geq 0.316 \times (1-\epsilon) \geq 0.25 = \frac{1}{4}$.
- We now show that the probability of outputting $\sigma = 1$ by all honest parties is at least $\frac{1}{4}$. It is obvious that if *no* party P_j has $V_j = 0$ and all honest parties associate V_j with P_j, then all honest parties will output 1. The probability of the first event is at least $(1 - \frac{1}{u})^n \geq e^{-1.15}$. Thus `Prob`(all honest parties output 1) $\geq e^{-1.15} \times (1-\epsilon) \geq 0.25 = \frac{1}{4}$. □

APPENDIX C: Proof for Protocol Vote

Lemma 17 [14] *All honest parties terminate Vote in constant time.*

PROOF (SKETCH): Every honest party P_i will A-cast his input x_i. As there are at least $n-t$ honest parties, from the properties of A-cast, every honest P_i will eventually have $|\mathcal{A}_i| = n-t$ and then will eventually have $|\mathcal{B}_i| = n-t$ and finally will eventually have $|C_i| = n-t$. Consequently, every honest P_i will terminate the protocol in constant time. □

Lemma 18 [14] *If all honest parties have same input σ, then all honest parties will output $(\sigma, 2)$.*

PROOF: Consider an honest party P_i. If all honest parties have same input σ, then at most t (corrupted) parties may A-cast $\overline{\sigma}$ as their input. Therefore, it is easy to see that *every* $P_k \in \mathcal{B}_i$ must have A-casted his `vote`

$b_k = \sigma$. Hence honest P_i will output $(\sigma, 2)$. □

Lemma 19 [14] *If some honest party outputs* $(\sigma, 2)$*, then every other honest party will eventually output either* $(\sigma, 2)$ *or* $(\sigma, 1)$ *in protocol* Vote.

Proof: Let an *honest* P_i outputs $(\sigma, 2)$. This implies that every $P_j \in B_i$ had A-casted `vote` $a_j = \sigma$. As $|B_i| = 2t + 1$, it implies that for every other *honest* P_j, it holds that $|B_i \cap B_j| \geq t + 1$. So every other *honest* P_j is bound to A-cast re-vote b_i as σ and hence will eventually output either $(\sigma, 2)$ or $(\sigma, 1)$. □

Lemma 20 [14] *If some honest party outputs* $(\sigma, 1)$ *and no honest party outputs* $(\sigma, 2)$ *then every other honest party will eventually output either* $(\sigma, 1)$ *or* $(\Lambda, 0)$.

Proof: Assume that some *honest* party P_i outputs $(\sigma, 1)$. This implies that all the parties $P_j \in C_i$ had A-casted the same `re-vote` $b_j = \sigma$. Since $|C_i| \geq n - t$, in the worst case there are *at most* t parties (outside C_i) who may A-cast `re-vote` $\overline{\sigma}$. Thus it is clear that no honest party will output $(\overline{\sigma}, 1)$. Now since the honest parties in C_i had `re-vote` as σ, there must be at least $t + 1$ parties who have A-casted their `vote` as σ. Thus no honest party can output $(\overline{\sigma}, 2)$ for which at least $n - t = 2t + 1$ parties are required to A-cast their `vote` as $\overline{\sigma}$. So we have proved that no honest party will output from $\{(\overline{\sigma}, 2), (\overline{\sigma}, 1)\}$. Therefore the honest parties will output either $(\sigma, 1)$ or $(\Lambda, 0)$. □

APPENDIX D: Proof for Protocol ABA

Lemma 21 [14] *Protocol ABA satisfies validity.*

Proof: The proof follows from the fact that if all honest parties have input σ, then by Lemma 18 every honest party will output $(y_1, m_1) = (\sigma, 2)$ upon termination of Vote and consequently A-cast (Terminate with σ) in the first iteration. □

Lemma 22 [14] *Protocol ABA satisfies* **Agreement**.

Proof: We show that if an honest party A-casts (Terminate with σ), then eventually every other honest party will A-cast the same. Let k be the first iteration when an honest party P_i A-casts (Terminate with σ). Then we prove that every other honest party will A-cast the same either in k^{th} iteration or in $(k+1)^{th}$ iteration. Since honest P_i has A-casted (Terminate with σ), it implies that $y_k = \sigma$ and $m_k = 2$ and P_i has outputted $(\sigma, 2)$ in the Vote protocol invoked in k^{th} iteration. By Lemma 19, every other honest party P_j will output either $(\sigma, 2)$ or $(\sigma, 1)$ in the Vote protocol invoked in k^{th} iteration. In case P_j outputs $(\sigma, 2)$, the it will A-cast (Terminate with σ) in k^{th} iteration itself. Furthermore every honest P_j will execute Vote with input $v_{k+1} = \sigma$ in the $(k+1)^{th}$ iteration. So clearly, in $(k+1)^{th}$ iteration every honest party will have same input σ. Therefore by Lemma 18, every honest party will output $(\sigma, 2)$ in Vote protocol invoked in $(k + 1)^{th}$ iteration. Hence all the honest parties will A-cast (Terminate with σ) either in iteration k or iteration $k + 1$. As all honest parties will eventually A-cast (Terminate with σ), every honest party will receive $n - t$ A-casts of (Terminate with σ) and will eventually output σ. □

Lemma 23 [14] *If all honest parties have initiated and completed iteration* k*, then with probability at least* $\frac{1}{4}$ *all honest parties have same value for* v_{k+1}.

Proof: We have two cases here:

1. If all honest parties execute step 4(c) in iteration k, then they have set their v_{k+1} as the output of Common-Coin. So by the property of Common-Coin, all the honest party have same v_{k+1} with probability at least $\frac{1}{4}$.
2. If some honest party has set $v_{k+1} = \sigma$ for some $\sigma \in \{0, 1\}$, either in step 4(a) or step 4(b) of iteration k, then by Lemma 20 no honest party will set $v_{k+1} = \overline{\sigma}$ in step 4(a) or step 4(b). Moreover, all the honest honest parties will output σ from Common-Coin with probability at least $\frac{1}{4}$. Now the parties starts executing Common-Coin, only after the termination of Vote. Hence the outcome of Vote is *fixed* before Common-Coin is invoked. Thus corrupted parties can not decide the output of Vote to prevent agreement. Hence with probability at least $\frac{1}{4}$, all the honest parties will set $v_{k+1} = \sigma$. □

Lemma 24 [14] *Conditioned on the event* C*, all honest parties terminate ABA in constant expected time.*

Proof: We first show that all the honest parties terminate protocol ABA within constant time after the *first* instance of A-cast of (Terminate with σ) is initiated by some *honest* party. Let the *first* instance of A-cast of (Terminate with σ) is initiated by some *honest* party in iteration k. Then all the parties will participate in Vote and Common-Coin protocols of all iterations up to iteration $k+1$. Both the executions can be completed in constant time. Moreover, by the proof of Lemma 22 every honest party will A-cast (Terminate with σ) by the end of iteration $k + 1$. These A-casts can be completed in constant time. Since an honest party terminates ABA after completing $t+1$ such A-casts, all the honest parties will terminate ABA within constant time after the *first* instance of A-cast of (Terminate with σ) is initiated by some *honest* party.

Now let the random variable τ be the count of number of iterations until the *first* instance of A-cast of (Terminate with σ) is initiated by some *honest* party. Obviously if no honest party ever A-casts (Terminate with σ) then $\tau = \infty$. Now conditioned on event C, all the honest parties terminate each iteration in constant time. So it is left to show that $E(\tau|C)$ is constant. We have

$$Prob(\tau > k|C_k) \leq Prob(\tau \neq 1|C_k) \times \ldots \times Prob(\tau \neq k \cap \ldots \cap \tau \neq 1|C_k)$$

From Lemma 23, it follows that each one of the k multiplicands of the right hand side of the above equation is at most $\frac{3}{4}$. Thus we have $Prob(\tau > k|C_k) \leq (\frac{3}{4})^k$. Now simple calculation shows that $E(\tau|C) \leq 16$. □

Lemma 25 [14] $Prob(C) \geq (1-\epsilon)$.

PROOF: We have

$$Prob(\overline{C}) \leq \sum_{k \geq 1} Prob(\tau > k \cap \overline{C_{k+1}}|C_k) \leq \sum_{k \geq 1} Prob(\tau > k|C_k) \cdot Prob(\overline{C_{k+1}}|C_k \cap \tau > k)$$

From the proof of Lemma 23, we have $Prob(\tau > k|C_k) \leq (\frac{3}{4})^k$. We will now bound $Prob(\overline{C_{k+1}}|C_k \cap \tau \geq k)$. If all the honest parties execute the k^{th} iteration and complete the k^{th} invocation of Common-Coin, then all the honest parties complete k^{th} iteration. Protocol Common-Coin is invoked with termination parameter $\frac{\epsilon}{4}$. Thus with probability $1 - \frac{\epsilon}{4}$, all the honest parties complete the k^{th} invocation of Common-Coin. Therefore, for each k, $Prob(\overline{C_{k+1}}|C_k \cap \tau \geq k) \leq \frac{\epsilon}{4}$. So we get

$$Prob(\overline{C}) \leq \sum_{k \geq 1} \frac{\epsilon}{4}(\frac{3}{4})^k = \epsilon$$ □