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

Efficient Unconditional Asynchronous Byzantine Agreement with Optimal Resilience

Arpita Patra · Ashish Choudhury · C. Pandu Rangan

Received: date / Accepted: date

Abstract We present an 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 at most t parties out of the n parties. The *amortized* communication complexity of our ABA protocol is $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits for attaining agreement on a *single* bit, where ϵ ($\epsilon > 0$) denotes the probability of non-termination. We compare our protocol with the best known optimally resilient ABA protocols of Canetti et al. (STOC 1993) and Abraham et al. (PODC 2008) and show that our protocol gains by a factor of $\mathcal{O}(n^8(\log \frac{1}{\epsilon})^3)$ over the ABA protocol of Canetti et al. and by a factor of $\mathcal{O}(n^5 \frac{\log n}{\log \frac{1}{\epsilon}})$ over the ABA protocol of Abraham et al. in terms of the communication complexity.

To design our protocol, we first present a new, 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 Canetti et al. Our AVSS protocol shares multiple secrets *simultaneously* and incurs lower communication complexity than executing multiple instances of an AVSS protocol sharing a single secret. To design our AVSS protocol, we further present a new asynchronous primitive called asynchronous weak commitment (AWC), which acts as a substitute for asynchronous weak secret sharing (AWSS), which was used as a primitive for designing AVSS by Canetti et al. We observe that AWC has weaker requirements than the AWSS and hence can be designed more efficiently.

The common coin primitive is one of the most important building blocks for the construction of an ABA protocol. The best known common coin protocol of Feldman et al. requires multiple instances of an AVSS protocol sharing a single secret as a black-box. Unfortunately, this common coin protocol does not achieve its goal when the *multiple invocations* of AVSS sharing a single secret is replaced by a *single invocation* of an AVSS protocol sharing multiple secrets simultaneously. Therefore in this paper, we extend the existing common coin protocol to make it compatible with our new AVSS protocol (sharing multiple secrets). As a byproduct, our new common coin protocol is much more communication efficient than the existing common coin protocol.

Few of the results in this paper appeared in PODC 2009 and PODC 2012.

Arpita Patra
Department of Computer Science
ETH Zurich, Switzerland
E-mail: arpitapatra10@gmail.com, arpita.patra@inf.ethz.ch

Ashish Choudhury
Department of Computer Science
University of Bristol, United Kingdom
E-mail: Ashish.Choudhary@bristol.ac.uk, partho31@gmail.com

C. Pandu Rangan
Department of Computer Science and Engineering
IIT Madras, Chennai India
E-mail: prangan55@yahoo.com, prangan55@gmail.com

1 Introduction

The problem of Byzantine Agreement (BA) was introduced in [26] and since then it has emerged as one of the most fundamental problems in distributed computing. Informally, a (threshold) BA protocol allows a set of n mutually distrusting parties, each holding a private bit, to agree on a common bit, even though t of the n parties may act maliciously in any arbitrary manner to make the honest parties disagree. The BA problem has been investigated extensively in various models (see for example [23, 18, 8, 2] and their references). It has been considered to be very interesting to study

the BA problem in the asynchronous settings, as an asynchronous network models a real-life network like the Internet more appropriately than a synchronous network. Though considered to be interesting, the problem of asynchronous BA (called ABA) has got relatively less attention in comparison to the BA problem in the synchronous setting. Unlike a synchronous network, there is no upper bound on the message delivery time in an asynchronous network and the messages can be arbitrarily delayed. The inherent difficulty in designing an asynchronous protocol is that we cannot distinguish between a slow but honest sender (whose messages are delayed) and a corrupted sender (who did not send any message); due to this, at any stage of an asynchronous protocol, a party cannot wait to receive the communication from all the n parties and the communication from t (potentially slow but honest) parties may have to be ignored. Due to this, designing asynchronous protocols calls for new techniques (for a comprehensive introduction to the asynchronous protocols, see [8]).

Compared to the synchronous BA protocols, the best known ABA protocols involve huge communication complexity (more on this in the sequel). The main goal of this paper is to provide a communication efficient ABA protocol.

1.1 Existing Results

We consider a computationally unbounded threshold adversary Adv, who can corrupt any t parties out of the n parties in a Byzantine[1] fashion. From [26], a BA (and hence an ABA) protocol tolerating Adv is possible if and only if $t < n/3$. Thus, an ABA protocol designed with exactly $n = 3t + 1$ parties is called an *optimally resilient* protocol. Fisher, Lynch and Paterson's impossibility result on deterministic ABA protocols [17] implies that any (randomized) ABA protocol must have non-terminating runs, where some honest party(ies) may not output any value and thus may not terminate at all. An ABA protocol is called $(1-\epsilon)$-terminating [9,8]), if the honest parties terminate the protocol with probability[2] at least $(1-\epsilon)$, where $\epsilon > 0$. On the other hand, an ABA protocol is called *almost-surely terminating* [1], if the probability of the occurrence of a non-terminating execution is asymptotically zero. The important parameters of an ABA protocol are:

- *Resilience*: it is the maximum number of corruptions (t) that the protocol can tolerate.
- *Communication Complexity* (CC): it is the total number of bits communicated by the honest parties in the protocol. The communication complexity has two parts: the *private* communication, done privately among the honest parties and the *broadcast* communication, done publicly. The broadcast primitive in the asynchronous setting is implemented using the Bracha's A-cast protocol [7].
- *Expected Running Time* (ERT): we consider the expected running time R of an ABA protocol, *conditioned on the event that the parties terminate*; this notion of expectancy is weaker than the usual notion of expectation, where the expectancy is taken over all possible events. Since an $(1-\epsilon)$-terminating ABA protocol may have non-terminating runs, in which case the (usual) expected running time will be infinity, we measure the expected running time with respect to the executions where the parties terminate. This notion of ERT was used in [9,8] (more on this later) and we follow the same notion.

Based on the above parameters, we summarize the best known ABA protocols in Table 1.

Table 1 Summary of the best known ABA protocols. Here $poly(x)$ stands for polynomial in x and AST stands for almost surely terminating. The ERT is conditioned on the event that the parties terminate the protocol.

Ref.	Type	Resilience	CC	ERT (R)
[7]	AST	$t < n/3$	$\mathcal{O}(2^n)$	$\mathcal{O}(2^n)$
[15,16]	AST	$t < n/4$	$poly(n)$	$\mathcal{O}(1)$
[9,8]	$(1-\epsilon)$	$t < n/3$	$poly(n, \frac{1}{\epsilon})$	$\mathcal{O}(1)$
[1]	AST	$t < n/3$	$poly(n)$	$\mathcal{O}(n^2)$

Common Approach Used to Design ABA Protocols : Over a period of time, the techniques and the design approaches of ABA protocols have evolved spectacularly. Rabin [28] 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 bit with some probability (called the success probability). Bracha [7] presented a simple implementation of the common coin protocol, whose success probability is $\Theta(2^{-n})$. Feldman and Micali [15,16] were the first to come up with a common coin protocol that has a constant success probability. The essence of [15] is the reduction of the common coin to that of designing an *Asynchronous Verifiable Secret Sharing* (AVSS) protocol. Here AVSS is a two phase protocol (sharing and reconstruction) carried out among the parties. Informally, the goal of an AVSS protocol is to allow a special party called *dealer* to share a secret s among the parties during the sharing phase in a way that would later allow for a unique reconstruction of the secret in the reconstruction phase, while preserving the secrecy of s until the reconstruction phase. Following [15, 16], almost all the protocols for ABA followed the same approach of reducing the problem of ABA to that of AVSS.

[1] A Byzantine corrupted party can behave in any arbitrary manner during the execution of a protocol.

[2] In the rest of the paper, all probabilities are taken over the random inputs of the honest parties.

The same approach is followed by the authors in [9,1] for designing their optimally resilient ABA protocols[3] and in this paper, we follow the same approach (a more detailed discussion on our and the existing approaches will appear in the sequel).

1.2 Our Motivation and Contribution

In the literature, a lot of attention has peen paid for constructing communication efficient BA protocols in the synchronous setting (see for example [6,10,14,27,20]). Unfortunately, the same is not the case for ABA protocols with optimal resilience. Designing an optimally resilient, communication efficient ABA protocol that runs in constant expected time is an important and interesting problem to work on. Our result in this paper marks a significant progress in this direction.

We present an optimally resilient, $(1-\epsilon)$-terminating ABA protocol that requires a private communication of $\mathcal{O}(\mathsf{R}n^4 \log \frac{1}{\epsilon})$ bits and broadcast of $\mathcal{O}(\mathsf{R}n^4 \log \frac{1}{\epsilon})$ bits for reaching agreement on $t+1 = \Theta(n)$ bits *concurrently*; here R is the expected running time of the protocol, conditioned on the event that the protocol terminates. So the (expected) *amortized* communication complexity of our protocol for reaching agreement on a single bit is $\mathcal{O}(\mathsf{R}n^3 \log \frac{1}{\epsilon})$ bits of private, as well as broadcast communication. Moreover, conditioned on the event that our ABA protocol terminates, it does so in constant expected time; i.e. $\mathsf{R} = \mathcal{O}(1)$. In Table 2, we compare our ABA protocol with the optimally resilient ABA protocols of [9,1].

Table 2 Comparison of our optimally resilient ABA protocol with the best known optimally resilient ABA protocols. Here AST stands for almost surely terminating. The communication complexity is the expected communication complexity as it depends on the expected running time of the protocol. The expected running time is conditioned on the event that the honest parties terminate the protocol.

Ref.	Type	ERT (R)	CC
[9]	$(1-\epsilon)$	$\mathcal{O}(1)$	Private– $\mathcal{O}(\mathsf{R}n^{11}(\log \frac{1}{\epsilon})^4)$ broadcast– $\mathcal{O}(\mathsf{R}n^{11}(\log \frac{1}{\epsilon})^2 \log n)$
[1]	AST	$\mathcal{O}(n^2)$	Private– $\mathcal{O}(\mathsf{R}n^6 \log n)$ broadcast– $\mathcal{O}(\mathsf{R}n^6 \log n)$
This[a] Article	$(1-\epsilon)$	$\mathcal{O}(1)$	Private– $\mathcal{O}(\mathsf{R}n^3(\log \frac{1}{\epsilon}))$ broadcast– $\mathcal{O}(\mathsf{R}n^3(\log \frac{1}{\epsilon}))$

[a] The communication complexity of our protocol is the amortized complexity for reaching agreement on a single bit.

From the table, we find that our ABA protocol achieves a huge gain in the communication complexity over the ABA protocol of [9], while keeping all other properties in place (namely constant expected running time and $(1-\epsilon)$-terminating). On the other hand, our ABA protocol enjoys the following merits over the ABA protocol of [1]:

1. Our ABA protocol is better in terms of the communication complexity when $(\log \frac{1}{\epsilon}) < n^5 \log n$.
2. Our ABA protocol runs in constant expected time (conditioned on the event that the protocol terminates). However, we stress that our ABA protocol is $(1-\epsilon)$-terminating, whereas the ABA protocol of [1] is almost surely terminating.

A Brief Discussion on the Approaches Used in the ABA Protocols of [9,1] and the Current Article : We now briefly discuss the approaches used in the ABA protocols of [9], [1] and the current article.

• The ABA protocol of Canetti et al. [9,8] uses the reduction from the ABA to AVSS. Hence they have first designed an AVSS protocol with $n = 3t+1$. There are well known inherent difficulties in designing an AVSS protocol with $n = 3t+1$ (see [9,8]). To overcome these difficulties, the authors in [9] used the following route to design their AVSS protocol: *ICP* $\rightarrow$ *A-RS* $\rightarrow$ *AWSS* $\rightarrow$ *Two & Sum AWSS* $\rightarrow$ *AVSS*, where $X \rightarrow Y$ means that protocol Y is designed using the protocol X as a black-box. Since the final AVSS protocol is designed on the top of so many sub-protocols, it is highly communication intensive as well as very much involved. The protocol incurs a private communication of $\mathcal{O}(n^9(\log \frac{1}{\epsilon})^4)$ bits and broadcast of $\mathcal{O}(n^9(\log \frac{1}{\epsilon})^2 \log(n))$ bits during the sharing phase; during the reconstruction phase, it incurs a private communication of $\mathcal{O}(n^6(\log \frac{1}{\epsilon})^3)$ bits and broadcast of $\mathcal{O}(n^6(\log \frac{1}{\epsilon}) \log(n))$ bits[4]. The protocol shares a single secret and all the (honest) parties terminate the protocol with probability at least $(1-\epsilon)$.

• The ABA protocol of [1] followed the same reduction from the ABA to AVSS as in [9], except that the use of AVSS is replaced by a variant of AVSS called *shunning* (asynchronous) VSS (SVSS), where each (honest) party is guaranteed to terminate almost-surely (i.e. with probability one). 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 fails to 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. In order to design their SVSS protocol, the authors in [1] first designed a weaker primitive called *weak* SVSS (W-SVSS), which is used as a black-box to design the SVSS protocol. Here W-SVSS is the "shunning" variant of the *asynchronous weak secret sharing*

[3] The authors in [1] followed a slightly different approach.

[4] The exact communication complexity analysis of the AVSS (and the ABA) protocol of [9] was not done earlier. For the sake of completeness, we carry out the same in **APPENDIX A**.

(AWSS) primitive, which was used in [9,8] for designing the AVSS protocol.

The use of SVSS instead of AVSS in generating the common coin causes the ABA protocol of [1] to run for $\mathcal{O}(n^2)$ expected time; however unlike the AVSS of [9] (which is $(1-\epsilon)$-terminating), the SVSS protocol of [1] is almost surely terminating. The protocol shares a single secret and requires a private communication of $\mathcal{O}(n^4 \log(n))$ bits and broadcast of $\mathcal{O}(n^4 \log(n))$ bits.

• Similar to [9,8] and [1], we too follow the same path of constructing an AVSS protocol to design our ABA protocol. So we first design a communication efficient AVSS protocol with $n = 3t + 1$. But instead of following the fairly complex route taken by [9] for the design of their AVSS protocol, we follow a much shorter route: *ICP* $\rightarrow$ *AWC* $\rightarrow$ *AVSS*. Here *asynchronous weak commitment* (AWC) is a new primitive introduced by us, which acts as a substitute for the AWSS primitive, which was used as a black-box in [9] as well as in [1] (the shunning variant) for designing the AVSS (SVSS). We find that AWC has "weaker" requirements than the AWSS and hence can be designed more efficiently than the existing AWSS protocols (the details are elaborated in Section 2.3.2 and section 4.3). More specifically, while the existing AWSS and W-SVSS are based on the idea of using *bi-variate* polynomials of degree t in each variable, we design an AWC scheme using Shamir secret sharing [30], which is based on *univariate* polynomials of degree t; this immediately implies a gain of $\Theta(n)$ in the communication complexity. In addition to introducing the new primitive AWC, we also extend the existing notion of *information checking protocol* (ICP) [9] to deal with multiple verifiers simultaneously, instead of a single verifier (the details will appear later). This helps us to use the ICP primitive in designing our AWC protocol.

Besides following a shorter route and introducing a new primitive for designing the AVSS, we significantly improve the communication complexity of each building block. In addition, each of the building blocks is designed to deal with multiple values concurrently (unlike the existing protocols) and this leads to a significant gain in the communication complexity. Specifically, our AVSS protocol requires a private communication and broadcast communication of $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits to share ℓ secrets concurrently, where $\ell \geq 1$. Moreover, it requires a broadcast communication of $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits to reconstruct the ℓ secrets. Like the AVSS protocol of [9], our AVSS protocol is also $(1-\epsilon)$-terminating, where all the parties terminate with probability at least $(1-\epsilon)$.

To design our ABA protocol, we also make several changes to the existing common coin protocol. As discussed earlier, the common coin protocol is a very important building block for an ABA protocol. The best known common coin protocol of [16,8] employs AVSS sharing a single secret. Informally, in the common coin protocol of [16], each party is asked to act as a dealer and share n random secrets using an AVSS protocol. So each party invokes n parallel instances of an AVSS protocol as a dealer to share n secrets in parallel. It is obvious that we can do better if each party invokes a single instance of our new AVSS protocol that can share n secrets concurrently. However, our detailed analysis of the existing common coin protocol shows that the above "trivial" modification leads to an incorrect common coin protocol. Hence we bring several modifications to the existing common coin protocol, so that it can use our new AVSS protocol (that shares multiple secrets concurrently). As a result, our new common coin protocol is more communication efficient than the existing common coin protocol of [8,9].

Interestingly, we show that our new common coin protocol is actually a multi-bit common protocol, which allows the parties to generate $(t+1)$ random common coins concurrently, which further allows the parties to reach agreement on $t+1$ bits concurrently.

1.3 Organization of the Paper

In the next section, we describe the asynchronous network model and formally define ABA, AVSS, AWC, AWSS and AICP. This is followed by the description of the existing tools used in our protocols. In section 3, we present our AICP, followed by our new primitive AWC in section 4; in the same section, we also compare our AWC scheme with the best known existing AWSS scheme of [25] and the existing W-SVSS scheme of [1]. In section 5, we present our AVSS scheme. The existing common coin protocol from [8] is presented in section 6. In the same section, we show that a simple substitution of the existing AVSS scheme (sharing a single secret) by our AVSS scheme (sharing multiple secrets) in the common coin protocol may lead to an incorrect common coin protocol. This is followed by the modifications needed to get a correct (multi-bit) common coin protocol. In section 7, we recall the existing voting protocol from [8], which is required along with the common coin protocol to get an ABA scheme. Finally, in section 8, we present our (multi-bit) ABA protocol.

2 Model and Definitions

We consider an asynchronous network consisting of n parties, say $\mathcal{P} = \{P_1, \ldots, P_n\}$, where each party is modelled as a probabilistic polynomial time interactive Turing machine. Each pair of parties is 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 Adv. The adversary Adv, completely dictates the parties under its control and can force them to deviate in any arbitrary manner during the execution of a protocol. The parties not under the influence of Adv 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 their destinations eventually). Moreover, the order in which the messages reach their destinations may be different from the order in which they were sent. To model the worst case scenario, Adv is given the power to schedule the delivery of every message in the network. Note that, while Adv can schedule the messages of the honest parties at its will, it has no access to the "contents" of the messages communicated between the honest parties.

As in [8], we consider a computation (protocol execution) in the asynchronous model as a sequence of *atomic steps*, where in each such step, a single party is *active*. The party is activated by receiving a message after which it performs an internal computation and then possibly sends messages on its outgoing channels. The order of the atomic steps are controlled by a "scheduler", which will be under the control of Adv. At the beginning of the computation, each party will be in a special *start* state. We say a party has *terminated/completed* the computation if it reaches a *halt* state, after which it does not perform any further computation. A protocol execution is said to be *complete* if each (honest) party terminates the protocol. Notice that the executions that complete do so after a finite number of steps.

Running Time of an Asynchronous Protocol : We now recall the definition of running time of an asynchronous protocol from [9]. Consider a virtual "global clock", measuring the 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 (infinite executions have infinite duration).

More precisely, consider a protocol execution, with some inputs and random inputs for the parties. Recall that such an execution is a sequence of atomic steps. Assume that the first atomic step is a fictitious step, where a special "wakeup" message is sent to all the parties. We now assign round-numbers to the atomic steps as follows: The fictitious atomic step, denoted l_0 is the only step at round zero. Next, for each $i > 0$, let l_i be the last atomic step where an $(i-1)$-message is delivered; we call a message an $(i-1)$-message if it was sent in an atomic step that belongs to round $(i-1)$. All the steps after l_{i-1} until (and including) the step l_i are in round i. The duration of the execution is the round number of the last atomic step.

Let C denote the event that the (honest) parties terminate the execution of a given protocol. The *expected running time* (ERT) of a protocol, relative to an adversary and some input values for the parties and conditioned on the event C, is the expected value of the duration of a *complete* execution (thus the expectancy is taken only over the random inputs of the parties in which the event C occurs). The (non-relative) expected running time $\mathsf{R}(\pi|C)$ of a protocol π, conditioned on the event C, is the maximum over all inputs $\vec{x} = (x_1, \ldots, x_n)$ and adversaries Adv, of the expected running time of the protocol relative to input $\vec{x}$ and adversary Adv and conditioned on the event C. That is:

$$\mathsf{R}(\pi|C) = \mathrm{Max}_{\vec{x},\mathsf{Adv}}\{\mathrm{Exp}_{\vec{r}}[D(\pi, \mathsf{Adv}, \vec{x}, \vec{r})|C]\},$$

where $D(\pi, \mathsf{Adv}, \vec{x}, \vec{r})$ is the duration of the execution of the protocol π with inputs $\vec{x} = (x_1, \ldots, x_n)$ and random inputs $\vec{r} = (r_1, \ldots, r_n)$ for the parties and with adversary Adv.

We now present the definition of ABA and the primitives which are used for the construction of our ABA protocol. Our ABA protocol is $(1-\epsilon)$-terminating, where $\epsilon > 0$. To bound the probability of non-termination by ϵ, all our protocols work over a finite field $\mathbb{F}$ where $\mathbb{F} = GF(2^\kappa)$, such that $|\mathbb{F}| > 2n$ and $\epsilon = 2^{-\Omega(\kappa)}$, for some non-zero κ. Thus each field element can be represented by $\kappa = \mathcal{O}(\log \frac{1}{\epsilon})$ bits. Moreover, we assume $n = poly(\kappa)$. That is, n is polynomial in κ. Thus we have that $n = poly(\log \frac{1}{\epsilon})$.

2.1 Asynchronous Byzantine Agreement (ABA)

Definition 1 (ABA [9]) : Let π be an asynchronous protocol executed among the set of parties in $\mathcal{P}$, where each party has a private binary input. We say that π is an $(1-\epsilon)$-terminating ABA protocol for a single bit, for an allowed error parameter ϵ (where $\epsilon > 0$) if the following holds for every possible Adv and every input vector of the parties:

1. **Termination**: If all the honest parties participate in the protocol then with probability[5] at least $(1-\epsilon)$, all honest parties eventually terminate the protocol.
2. **Correctness**: All the honest parties who have terminated the protocol hold identical bit as the output. Moreover, if all the honest parties had the same input, say ρ, then all honest parties upon termination output ρ.

The above definition can be extended in a straight forward way for agreement on ℓ bits, where $\ell > 1$ and we call such a protocol as *multi-bit* ABA protocol.

[5] Here and in the sequel the probabilities are taken over the random inputs of the parties.

2.2 Asynchronous Verifiable Secret Sharing (AVSS)

We now present the definition of AVSS, which is the major component in the design of the existing common coin protocol. As mentioned earlier, any AVSS protocol consists of two protocols: a sharing protocol (called Sh) where a special player called dealer shares a secret among the parties and a reconstruction protocol (called Rec) where the parties reconstruct the secret from its shares. More formally:

Definition 2 (Asynchronous Verifiable Secret Sharing (AVSS) [9]) Let (Sh, Rec) be a pair of protocols for the n parties, where a dealer $\mathsf{D} \in \mathcal{P}$ has a private input $s \in \mathbb{F}$ for Sh. Then (Sh, Rec) is a $(1-\epsilon)$-AVSS scheme[6], for an allowed error parameter ϵ (where $\epsilon > 0$), if the following requirements hold for every possible behavior of Adv:

- **Termination**: With probability at least $(1-\epsilon)$, the following requirements hold:
  1. If D is *honest* and all the honest parties participate in the protocol Sh, then each *honest* party eventually terminates the protocol Sh.
  2. If some honest party terminates Sh, then irrespective of the behavior of D, each *honest* party eventually terminates Sh.
  3. If all the honest parties have terminated Sh and invoked Rec, then each *honest* party eventually terminates Rec.
- **Correctness**: If some honest party terminates Sh, then there exists a fixed value $\overline{s}$, where $\overline{s} \in \mathbb{F} \cup \{\perp\}$, such that the following requirements hold with probability at least $(1-\epsilon)$:
  1. Each *honest* party outputs $\overline{s}$ upon terminating Rec (i.e. $\overline{s}$ is the reconstructed secret). This property is also called the strong-commitment property[7].
  2. If D is *honest*, then $\overline{s}$ is the shared secret; i.e. $\overline{s} = s$.
- **Secrecy**: Let VIEW(s) denote the random variable (over the random inputs of the parties), describing the view of the adversary during an execution of Sh, where D has the input s. If D is *honest* during Sh and no honest party has begun executing the protocol Rec, then the distribution of VIEW(s) is the same (i.e. identically distributed) for all $s \in \mathbb{F}$.

Before proceeding further, we make the following note:

Note 1 There exists a "stronger" definition of VSS (AVSS) which requires that D's committed secret $\overline{s} \in \mathbb{F}$, instead of $\mathbb{F} \cup \{\perp\}$ [22]. Such a stronger definition[8] is required if VSS is used for secure multi-party computation MPC [3]. However, VSS (AVSS) satisfying the above (weak) definition is enough for the construction of (asynchronous) BA protocols. We also note that the above weak definition of VSS is used in [24] to study the round complexity of VSS. The above definition is equivalent to saying that $\overline{s} \in \mathbb{F}$, by fixing a default value in $\mathbb{F}$, which may be output in case the Rec protocol ends with a $\perp$. □

The above definition can be extended in a straight forward way for a secret $\vec{S} = (s^1, \ldots, s^\ell)$, containing ℓ elements from $\mathbb{F}$.

2.3 Asynchronous Weak Commitment (AWC)

To design our AVSS protocol, we introduce a new asynchronous primitive called AWC. This new primitive will act as a "substitute" for the *asynchronous weak secret sharing* (AWSS) primitive; recall that AWSS was used as a primitive in [9] to design a $(1-\epsilon)$-AVSS, while a "shunning variant" of AWSS was used as a primitive in [1] to design an almost surely terminating shunning AVSS.

Informally, an AWC scheme consists of two protocols: a commitment protocol (called Com) and a decommitment protocol (called DeCom). In the commitment protocol, a special party called Committer "commits" a secret to the parties in a distributed fashion and during the decommitment protocol, Committer "decommits" the secret and the parties verify whether this was the secret committed earlier and either accept or reject the secret. More formally:

Definition 3 (Asynchronous Weak Commitment (AWC)) Let (Com, DeCom) be a pair of protocols for the n parties, where a Committer $\in \mathcal{P}$ has a private input $s \in \mathbb{F}$ for Com (the secret to be committed). In the protocol DeCom, Committer has a private input $s^\star \in \mathbb{F} \cup \{\perp\}$ (the secret to be decommitted) and each party upon terminating DeCom either outputs $s^\star$ or outputs $\perp$. Then (Com, DeCom) is a $(1-\epsilon)$-AWC scheme, for an allowed error parameter ϵ (where $\epsilon > 0$), if the following requirements hold for every possible behavior of Adv:

- **Termination**: With probability at least $(1-\epsilon)$, the following requirements hold:
  1. If Committer is *honest* and all the honest parties participate in the protocol Com, then each *honest* party eventually terminates the protocol Com.
  2. If some *honest* party terminates Com, then irrespective of the behavior of Committer, each *honest* party eventually terminates Com.
  3. If all the honest parties have terminated Com, Committer invokes DeCom and all the honest parties participate in DeCom, then each *honest* party eventually terminates DeCom.
- **Correctness**: If some honest party terminates Com, then there exists a fixed value $\overline{s}$, where $\overline{s} \in \mathbb{F} \cup \{\perp\}$, such

[6] Such schemes are also called statistical AVSS.

[7] We often say that D has committed $\overline{s}$ during the sharing protocol.

[8] The interpretation of $\overline{s} = \perp$ will be clear during the description of our protocol.

that the following requirements[9] hold with probability at least $(1-\epsilon)$ upon the completion (if it completes) of DeCom:

1. If an *honest* party outputs $s^\star \in \mathbb{F}$ (namely $s^\star$ is accepted as the decommitted secret and $s^\star$ is from the field $\mathbb{F}$), then all the honest parties output $s^\star$. Moreover, $s^\star = \overline{s}$ (so in this case, $\overline{s} \in \mathbb{F}$).
2. If Committer is *honest*, then all the *honest* parties output $s^\star$, where $s^\star = \overline{s} = s$.

– **Secrecy**: Let VIEW(s) denote the random variable (over the random inputs of the parties), describing the view of the adversary during an execution of Com, where Committer has the input s. If Committer is *honest* during Com and has not begun executing the protocol DeCom, then the distribution of VIEW(s) is the same (i.e. identically distributed) for all $s \in \mathbb{F}$.

The above definition can be extended in a straight forward way for a secret $\vec{S} = (s^1, \ldots, s^\ell)$, containing ℓ elements from $\mathbb{F}$.

Comparison of AVSS and AWC: There are two (subtle) differences between the AVSS and AWC. The first is regarding the termination of the reconstruction protocol and the decommitment protocol respectively. In AVSS, we demand that irrespective of the behavior of D, if the honest parties terminate the sharing protocol and invoke the reconstruction protocol, then all the honest parties should eventually terminate the reconstruction protocol. But this need not be the case for the decommitment protocol. Specifically, the decommitment protocol requires the *participation* of Committer and so if Committer is *corrupted* and does not invoke the decommitment protocol, then the honest parties will never terminate the decommitment protocol.

The second difference is regarding the committed value. In AVSS, we demand that irrespective of the behavior of D, *only* the committed value $\overline{s} \in \mathbb{F} \cup \{\perp\}$ should be reconstructed by the honest parties in the reconstruction protocol. That is, a *corrupted* D cannot later change the committed secret during the reconstruction protocol in co-operation with the corrupted parties. However, in AWC, a *corrupted* Committer can later decommit an $s^\star$ during the decommitment protocol, where $s^\star$ is *different* from the secret $\overline{s}$, committed during the commitment protocol, provided $\overline{s} \neq \perp$ and $s^\star = \perp$ (so $\overline{s} \in \mathbb{F}$). Notice that even though Committer can later "change" the committed value, the change is allowed *only* from a non-$\perp$ committed value to $\perp$; so a corrupted Committer cannot commit an $\overline{s} \in \mathbb{F}$ and later decommit $s^\star \in \mathbb{F}$, where $s^\star \neq \overline{s}$.

[9] We often say that $\overline{s}$ is the value committed by Committer during the commitment protocol. The interpretation of $\overline{s} = \perp$ will be clear during the description of our protocol.

Finally, we note that the above differences between AVSS and AWC holds only when Committer is *corrupted*; otherwise AWC provides the same properties as the AVSS.

2.3.1 Asynchronous Weak Secret Sharing (AWSS)

Like AVSS, an AWSS protocol also consists of two protocols: a sharing protocol (called WSh), where the dealer shares a secret and a reconstruction protocol (called WRec), where the parties reconstruct the secret from its shares. An AWSS protocol also has three properties, namely **Termination, Correctness** and **Secrecy**. The **Termination** and the **Secrecy** conditions are the same as for the AVSS (see Definition 2), except that Sh is replaced by WSh and Rec is replaced by WRec. The **Correctness** condition is also the same as for the AVSS for the case when the dealer is *honest*; however, the **Correctness** condition is *weakened* in AWSS for the case when the dealer is *corrupted* as follows:

> If an honest party terminates WSh then a value, say $\overline{s} \in \mathbb{F} \cup \{\perp\}$ is fixed. Moreover, with probability at least $(1-\epsilon)$, each honest party will output *either* $\overline{s}$ or $\perp$ at the end[10] of WRec.

2.3.2 Comparison of AWSS and AWC

Property wise, the sharing protocol of AWSS and the commitment protocol of AWC demands the *same*; namely a distributed commitment to a unique value $\overline{s}$, which should be secure if D (resp. Committer) is honest. However, the (subtle) difference is between the decommitment protocol and the reconstruction protocol. The difference is about the role of Committer and D for the termination of the respective protocols: the reconstruction protocol of an AWSS scheme does not demand a special role by D to enforce the termination. So this protocol will always terminate (if it is invoked by the honest parties) even if D is corrupted and does not participate. On the contrary, the decommitment protocol demands a special role from Committer to enforce the termination. Here Committer has to invoke the protocol; so if Committer is corrupted and does not invoke the decommitment protocol, this protocol may never terminate.

The above difference intuitively suggests that D has to work/communicate "more" during the sharing protocol of an AWSS scheme, as compared to what is done by Committer during the commitment protocol of an AWC scheme. The veracity of the intuition is confirmed by our ability to design an AWC protocol more efficiently than the existing AWSS protocols (for the comparison, see Section 4.3).

[10] It is possible that some honest parties output $\overline{s}$, while others output $\perp$.

2.4 Asynchronous Information Checking Protocol (AICP)

An Information Checking Protocol (ICP) is used for authenticating data in the presence of computationally unbounded corrupted parties. The notion of ICP was first introduced by Rabin et al. [29]. As described in [29,9,12], an ICP is executed among three parties: a Signer, an intermediary INT and a Verifier. Informally, an ICP consists of two phases (where each phase is implemented by different protocol(s)):

- *Signature generation phase*: here Signer computes his IC (information checking) signature on a secret $s \in \mathbb{F}$, denoted by $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathsf{Verifier}, s)$ and hands it to INT. Signer also computes some verification information and hands it to Verifier.
- *Signature revelation phase*: here INT reveals the signature $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathsf{Verifier}, s)$, claiming that he has received it from Signer. Verifier then verifies the signature, using the verification information and either accepts or rejects the signature (and hence s).

IC signature may be considered as the information theoretically secure substitute of traditional digital signatures. It provides the properties like unforgeability and non-repudiation; in addition, it also provides information theoretic security. That is, if Signer and INT are honest, then at the end of the signature generation phase, s remains secure in the information theoretic sense.

We extend the above notion of ICP in two directions: First, we consider *multiple* verifiers, where each party in $\mathcal{P}$ acts as a Verifier. This will be later helpful in using the ICP as a tool in our AWC protocol. It is important here to note that Signer and INT can be any two parties from the set $\mathcal{P}$. They just play their "special" role as Signer and INT. Second, instead of a *single* secret, we consider ICP that can deal with *multiple* secrets concurrently and thus achieves better communication complexity, than executing multiple instances of ICP, dealing with a single secret. Our ICP is executed in the asynchronous settings and thus we call it AICP. We now formally define AICP.

Definition 4 ((Multi-Verifier) Asynchronous Information Checking Protocol (AICP)) An AICP involves three entities: a Signer $\in \mathcal{P}$, an intermediary INT $\in \mathcal{P}$ and the set of n parties in $\mathcal{P}$ acting as verifiers. The protocol is carried out in three phases (where each phase is implemented by a protocol):

1. *Generation Phase*: this phase is initiated by Signer, where Signer has a secret input $\vec{S} = (s^1, \ldots, s^\ell) \in \mathbb{F}^\ell$. In this phase, Signer sends $\vec{S}$ to INT, along with some *authentication* information. In addition, Signer sends some *verification* information to each individual verifier.
2. *Verification Phase*: this phase is initiated by INT, where INT interacts with Signer and the verifiers to ensure that the secret $\vec{S}$ received from Signer will be later accepted by each (honest) verifier in $\mathcal{P}$ during the revelation phase. During the interaction, Signer has the option of *replacing* the secret $\vec{S}$, as well as the authentication information and the verification information (by sending new values for them) . The secret $\vec{S}$, along with the authentication information, which is finally possessed by INT at the end of this phase is called Signer's *IC signature* on $\vec{S}$, denoted as $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$, *given* to INT by Signer.
3. *Revelation Phase*: this is carried out by INT and the verifiers in $\mathcal{P}$, where INT and the verifiers interact with each other. Here INT reveals $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$ and each verifier checks $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$ with respect to his verification information and possibly let his findings known to the other verifiers. Based on the verification information and possibly on the response received from the other verifiers, each individual verifier $P_i \in \mathcal{P}$ either outputs $\mathsf{Reveal}_i = \vec{S}$ (indicating that P_i is convinced that INT indeed obtained $\vec{S}$ from Signer) or $\mathsf{Reveal}_i = \perp$ (indicating that P_i is not convinced that INT obtained $\vec{S}$ from Signer). Accordingly, we say that the verifier P_i accepted (resp. rejected) the ICSig and hence $\vec{S}$.

A triplet of protocols $(\mathsf{Gen}, \mathsf{Ver}, \mathsf{RevealPublic})$ (for the generation, verification and revelation phase respectively), where Signer has a private input $\vec{S} \in \mathbb{F}^\ell$ for the protocol Gen, is called an $(1-\epsilon)$-AICP, for an allowed error parameter ϵ (where $\epsilon > 0$), if the following requirements hold for every possible behavior of Adv:

1. **AICP-Correctness1**: If Signer and INT are *honest*, then each *honest* verifier $P_i \in \mathcal{P}$ will output $\mathsf{Reveal}_i = \vec{S}$ at the end of RevealPublic.
2. **AICP-Correctness2**: If INT is *honest* and possesses ICSig $(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$ at the end of Ver, then with probability at least $(1-\epsilon)$, each *honest* verifier $P_i \in \mathcal{P}$ will output $\mathsf{Reveal}_i = \vec{S}$ at the end of RevealPublic.
3. **AICP-Correctness3:** If Signer is *honest* and INT possesses $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$ at the end of Ver, then the probability that an *honest* verifier P_i outputs $\mathsf{Reveal}_i = \vec{S^\star}$ at the end of RevealPublic, where $\vec{S^\star} \neq \vec{S}$ is at most ϵ.
4. **AICP-Secrecy:** If Signer and INT are *honest*, then the information received by Adv till the end of Ver is distributed independently of the secret $\vec{S}$. This implies that if Signer and INT are honest and INT has not executed RevealPublic, then Adv has no information about $\vec{S}$.

2.5 Existing Tools

A-cast : In our protocols, we use the asynchronous broadcast primitive, called A-cast, which was introduced and ele-

gantly implemented by Bracha [7] with $3t + 1$ parties. Formally, A-cast is defined as follows:

Definition 5 (A-cast [9]) Let π be an asynchronous protocol initiated by a special party in $\mathcal{P}$ (called Sender), having an input m (the message to be broadcast). We say that π is an A-cast protocol if the following requirements hold, for every possible behavior of Adv:

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

For the sake of completeness, we recall the Bracha's A-cast protocol from [8] and present it in Fig. 1.

Fig. 1 Bracha's A-cast protocol with $n = 3t + 1$.

A-cast(m)

The following step is executed only by Sender:

1. Send the message $(\texttt{MSG}, m)$ to all the parties.

For $i = 1, \ldots, n$, every party P_i in $\mathcal{P}$ (including Sender) executes the following steps:

1. Upon receiving a message $(\texttt{MSG}, m)$ from Sender, send the message $(\texttt{ECHO}, m)$ to all the parties.
2. Upon receiving $(n - t)$ messages $(\texttt{ECHO}, m^\star)$ that agree on the value of $m^\star$, send the message $(\texttt{READY}, m^\star)$ to all the parties.
3. Upon receiving $(t+1)$ messages $(\texttt{READY}, m^\star)$ that agree on the value of $m^\star$, send the message $(\texttt{READY}, m^\star)$ to all the parties.
4. Upon receiving $(n - t)$ messages $(\texttt{READY}, m^\star)$ that agree on the value of $m^\star$, send the message $(\texttt{OK}, m^\star)$ to all the parties, output $m^\star$ and terminate the protocol.

Theorem 1 ([8]) *Protocol A-cast requires a private communication of $\mathcal{O}(\ell n^2)$ bits to broadcast an ℓ bit message.*

In the rest of the paper, we use the following notation while invoking the A-cast protocol:

Notation 1 (Notation for Using the A-cast Protocol) We say that a party P_j receives the message m from the broadcast of P_i, if P_j (as a receiver) completes the execution of P_i's A-cast (namely the instance of the A-cast protocol where P_i is Sender), with m as the output.

Randomness Extractor : In our common coin protocol, we will use a well known method for randomness extraction in the information theoretic settings. The setting is as follows: we are given a set of N values from $\mathbb{F}$, say $a_1, \ldots, a_N$, such that *at least* K out of these N values are selected uniformly and randomly from $\mathbb{F}$; however, the exact identity of those K values are not known. The goal is to compute K values $b_1, \ldots, b_K$ from $a_1, \ldots, a_N$, each of which is uniformly distributed over $\mathbb{F}$. This is achieved through the following well-known method introduced in [5,4]: let $f(x)$ be the polynomial of degree at most $N - 1$, such that $f(i) = a_{i+1}$, for $i = 0, \ldots, (N - 1)$. Then set $b_1 = f(N), \ldots, b_K = f(N + K - 1)$ (of course we require $|\mathbb{F}| \geq N + K$ for this, which will be the case in our protocol). The elements $b_1, \ldots, b_K$ are uniformly distributed over $\mathbb{F}$, as there exists a one-to-one mapping between $b_1, \ldots, b_K$ and the K random elements in the vector $(a_1, \ldots, a_N)$. We call this algorithm as EXT and invoke it as $(b_1, \ldots, b_K) = \mathsf{EXT}(a_1, \ldots, a_N)$.

3 Asynchronous Information Checking Protocol

We present an AICP called MultiVerifierAICP. The underlying idea behind the protocol is as follows: let $\vec{S} = (s^1, \ldots, s^\ell) \in \mathbb{F}^\ell$ be the secret, on which Signer wants to give his IC signature to INT. For this, during the generation phase, Signer selects a polynomial $F(x)$ of degree at most $\ell+t$ over $\mathbb{F}$, which is an otherwise random polynomial such that $F(\beta_i) = s^i$, for $i = 1, \ldots, \ell$. Here $\beta_1, \ldots, \beta_\ell$ are pre-selected, distinct elements from $\mathbb{F}$, which are known publicly. The polynomial $F(x)$ is given to INT. In addition, to each verifier P_i, Signer gives the value of $F(x)$ at a random *evaluation point* α_i (different from all β_j's). During the revelation phase, INT publicly discloses $F(x)$ (by broadcasting) and each verifier P_i checks whether the value held by him is indeed the value of $F(x)$ at α_i. It is easy to note that the above simple protocol satisfies the **AICP-Correctness3** property. Specifically, if Signer is *honest* and INT is *corrupted*, then INT will not know the evaluation point α_i of an honest verifier P_i and so with high probability, INT cannot disclose an incorrect polynomial $\overline{F}(x)$, different from $F(x)$, and still remain unnoticed by an honest verifier P_i.

The above protocol also satisfies the **AICP-Secrecy** property, as the degree of $F(x)$ is at most $\ell + t$ and at most t points on $F(x)$ will be disclosed to Adv (through t corrupted verifiers); so Adv will lack ℓ additional points on $F(x)$ to uniquely interpolate $F(x)$ and obtain the value of $F(x)$ at $\beta_1, \ldots, \beta_\ell$ (which are the secrets). More specifically, from the view point of Adv, who holds t random points on a polynomial of degree at most $\ell + t$, there is a single polynomial that will be "consistent" with those t random points and *any* ℓ secrets.

Unfortunately, the above protocol steps alone are not enough to achieve the **AICP-Correctness2**. This is because

if Signer is *corrupted*, then he might give $F(x)$ to INT, but evaluations of a different polynomial $\overline{F}(x) \neq F(x)$ to each honest verifier. To avoid this situation, we have to add additional steps in the above protocol, which allows INT to interact with the verifiers, to check the consistency of $F(x)$ (received by him) and the values at the random evaluation points (received by the verifiers). The interaction should be in a "zero-knowledge" fashion, meaning that it should not compromise the privacy of the information held by INT and the (honest) verifiers.

To enable the zero-knowledge interaction, Signer distributes some additional information to INT and the verifiers during the generation phase. Specifically, in addition to $F(x)$, Signer gives to INT another random polynomial $R(x)$ of degree at most $\ell+t$. In parallel, to each individual verifier P_i, Signer gives the value of $R(x)$ at α_i. Now the specific details of the zero-knowledge consistency checking, along with the other formal steps of the protocol MultiVerifierAICP are given in Fig. 2.

We now prove the properties of the protocol MultiVerifier AICP. Before that, we prove the following two claims, which will be required to prove the properties of the protocol.

Claim 1. *Let $F(x), R(x)$ be two polynomials of degree at most $\ell + t$ and (α_i, v_i, r_i) be a tuple such that $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$. Then for a random $d \in \mathbb{F} \setminus \{0\}$, the condition $B(\alpha_i) \neq dv_i + r_i$ will be true except with probability at most ϵ, where $B(x) \stackrel{def}{=} dF(x) + R(x)$.*

PROOF: We first argue that there exists *only one* non-zero $d \in \mathbb{F}$, for which the condition $B(\alpha_i) = dv_i + r_i$ 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 $e \in \mathbb{F}$, where $e \neq d$, 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

$$dF(\alpha_i)+R(\alpha_i) = dv_i+r_i \quad \text{and} \quad eF(\alpha_i)+R(\alpha_i) = ev_i+r_i.$$

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 from $\mathbb{F} \setminus \{0\}$, the condition $B(\alpha_i) = dv_i + r_i$ holds with probability at most $\frac{1}{|\mathbb{F}|-1} \approx \epsilon$. □

Claim 2. *In the protocol* MultiVerifierAICP*, if* Signer *and* INT *are honest, then* Signer *will broadcast the* `OK` *message (and not the polynomial $F(x)$) during the protocol* Ver.

PROOF: Follows from the fact that if Signer and INT are honest then $F(\alpha_i) = v_i$ and $R(\alpha_i) = r_i$ holds for every verifier $P_i \in \mathcal{R}$. Moreover, INT will correctly broadcast the polynomial $B(x)$ during the protocol Ver, where $B(x) \stackrel{def}{=} dF(x) + R(x)$ and so $dv_i + r_i = B(\alpha_i)$ will hold for every verifier $P_i \in \mathcal{R}$. □

Lemma 1 (AICP-Correctness1) *If* Signer *and* INT *are honest, then in the protocol* RevealPublic*, each honest verifier $P_i \in \mathcal{P}$ will output $Reveal_i = \vec{S^\star} = \vec{S}$.*

PROOF: From Claim 2, if Signer and INT are honest, then Signer will broadcast the `OK` message during Ver, as $v_i = F(\alpha_i)$ and $r_i = R(\alpha_i)$ will be true for each verifier $P_i \in \mathcal{R}$ and there are at least $t+1$ honest verifiers in $\mathcal{R}$. Now during RevealPublic, INT will correctly broadcast ICSig $= F(x)$ (so $F^\star(x) = F(x)$) and each honest verifier $P_i \in \mathcal{R}$ will broadcast the `Accept` message, as the condition *C1*, i.e. $v_i = F(\alpha_i)$ will hold for each of them. Hence each honest verifier $P_i \in \mathcal{P}$ will eventually receive $t+1$ `Accept` messages from at least $t+1$ verifiers in $\mathcal{R}$ and hence will output $\mathsf{Reveal}_i = \vec{S^\star}$. Now it is easy to see that $\vec{S^\star} = \vec{S}$. □

Lemma 2 (AICP-Correctness2) *If* INT *is honest and possesses* ICSig$(\text{Signer}, \text{INT}, \mathcal{P}, \vec{S})$ *at the end of* Ver*, then with probability at least $(1-\epsilon)$, each honest verifier $P_i \in \mathcal{P}$ will output $\mathsf{Reveal}_i = \vec{S}$ at the end of* RevealPublic*.*

PROOF: If Signer is honest, then the lemma follows from Lemma 1. So we consider the case when Signer is corrupted. We claim that in this case, except with probability ϵ, each *honest* verifier $P_i \in \mathcal{R}$ will broadcast the `Accept` message during RevealPublic, in response to ICSig$(\text{Signer}, \text{INT}, \mathcal{P}, \vec{S})$ broadcasted by the honest INT. Now since there are at least $t+1$ honest verifiers (and at most t corrupted verifiers) in the set $\mathcal{R}$, it implies that each honest verifier $P_i \in \mathcal{P}$ will eventually receive the `Accept` message from at least $t+1$ different verifiers in $\mathcal{R}$ and will output $\mathsf{Reveal}_i = \vec{S}$. We consider the following two cases, depending upon what Signer broadcasts during Ver:

1. Signer *broadcasts a polynomial $\overline{F}(x)$ during* Ver: In this case, the above claim is true, as INT will set ICSig(Signer, INT, $\mathcal{P}, \vec{S}) = \overline{F}(x)$ as the IC signature and each *honest* verifier $P_i \in \mathcal{R}$ will set $v_i = \overline{F}(\alpha_i)$ as his verification information. During RevealPublic, the honest INT will correctly broadcast ICSig$(\text{Signer}, \text{INT}, \vec{S}, \mathcal{P}) = F^\star(x) = \overline{F}(x)$ and so the condition *C1*, namely $F^\star(\alpha_i) = v_i$ will hold for each honest verifier $P_i \in \mathcal{R}$.
2. Signer *broadcasts the* `OK` *message during* Ver: In this case, INT will broadcast ICSig$(\text{Signer}, \text{INT}, \vec{S}, \mathcal{P}) = F^\star(x) = F(x)$ during RevealPublic. Now we have the following cases depending on the relationship that holds between $(F(x), R(x))$ held by INT and the tuple $(\alpha_i, v_i, r_i))$ held by an *honest* verifier $P_i \in \mathcal{R}$:
   (a) $F(\alpha_i) = v_i$: clearly P_i will broadcast the `Accept` message, as the condition *C1*, i.e. $F^\star(\alpha_i) = v_i$ will hold for P_i.
   (b) $F(\alpha_i) \neq v_i$ *and* $R(\alpha_i) = r_i$: Here also P_i will broadcast the `Accept` message, as the condition *C2*, i.e. $B(\alpha_i) \neq dv_i + r_i$ will hold for P_i.
   (c) $F(\alpha_i) \neq v_i$ *and* $R(\alpha_i) \neq r_i$: In this case, P_i will broadcast the `Accept` message, except with probability at most ϵ, as the condition *C2*, i.e. $B(\alpha_i) \neq dv_i + r_i$ will hold for P_i, which follows from Claim 1. More specifically, if $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$,

Fig. 2 AICP with $n = 3t + 1$.

MultiVerifierAICP(Signer, INT, $\mathcal{P}$, $\vec{S} = (s^1, \ldots, s^\ell)$, ϵ)

Gen(Signer, INT, $\mathcal{P}$, $\vec{S}$, ϵ)

The following steps are executed only by Signer:

1. Select a random polynomial $F(x)$ over $\mathbb{F}$ of degree at most $\ell + t$, such that $F(\beta_i) = s^i$, for $i = 1, \ldots, \ell$, where $\beta_1, \ldots, \beta_\ell$ are publicly known distinct elements from $\mathbb{F}$.
2. Select a random polynomial $R(x)$ over $\mathbb{F}$ of degree at most $\ell + t$.
3. For $i = 1, \ldots, n$, select α_i randomly from $\mathbb{F}$ as the evaluation point, corresponding to the verifier P_i, subject to the condition that $\alpha_i \in \mathbb{F} \setminus \{\beta_1, \ldots, \beta_\ell\}$.
4. Send $F(x), R(x)$ to INT and for $i = 1, \ldots, n$, send (α_i, v_i, r_i) to the verifier P_i, where $v_i = F(\alpha_i)$ and $r_i = R(\alpha_i)$.

Ver(Signer, INT, $\mathcal{P}$, $\vec{S}$, ϵ)

Signer, INT and the verifiers in $\mathcal{P}$ interact as follows:

1. For $i = 1, \ldots, n$, verifier P_i sends the message `(Received, i)` to INT after receiving (α_i, v_i, r_i) from Signer.
2. Upon receiving the message `(Received, i)` from the verifier P_i, INT includes P_i to a dynamic set $\mathcal{R}$, which is initially $\emptyset$. If $|\mathcal{R}| \geq 2t + 1$, then INT randomly selects $d \in \mathbb{F} \setminus \{0\}$, computes $B(x) = dF(x) + R(x)$ and broadcasts $(d, B(x), \mathcal{R})$ (as a Sender).
3. Upon receiving $(d, B(x), \mathcal{R})$ from the broadcast of INT, Signer checks $dv_i + r_i \stackrel{?}{=} B(\alpha_i)$ for every $P_i \in \mathcal{R}$.
   If $dv_i + r_i \neq B(\alpha_i)$ for every verifier $P_i \in \mathcal{R}$, then Signer broadcasts the polynomial $F(x)$.
   Otherwise Signer broadcasts the message `OK`.
4. Depending upon the message received from the broadcast of Signer, INT and the verifiers do the following:
   (a) If `OK` is received from the broadcast of Signer then:
      i. INT sets ICSig(Signer, INT, $\vec{S}, \mathcal{P}$) $= F(x)$, where $F(x)$ is received from Signer during Gen.
      ii. For $i = 1, \ldots, n$, verifier P_i sets (α_i, v_i) as his verification information, where α_i and v_i is received by P_i from Signer during Gen.
   (b) If a polynomial $\overline{F}(x)$ of degree at most $\ell + t$ is received from the broadcast of Signer, then:
      i. INT sets ICSig(Signer, INT, $\vec{S}, \mathcal{P}$) $= \overline{F}(x)$.
      ii. For $i = 1, \ldots, n$, verifier P_i computes $v_i = \overline{F}(\alpha_i)$ and sets (α_i, v_i) as his verification information.

RevealPublic(Signer, INT, $\mathcal{P}$, $\vec{S}$, ϵ)

1. INT broadcasts ICSig(Signer, INT, $\vec{S}, \mathcal{P}$).
2. For $i = 1, \ldots, n$, verifier P_i does the following:
   (a) Wait to receive ICSig(Signer, INT, $\vec{S}, \mathcal{P}$) from the broadcast of INT. Upon receiving, interpret ICSig(Signer, INT, $\vec{S}, \mathcal{P}$) as a polynomial $F^\star(x)$ of degree at most $\ell + t$.
   (b) If $P_i \in \mathcal{R}$, then broadcast the message `Accept` if one of the following conditions holds:
      i. $v_i = F^\star(\alpha_i)$ — we call this as condition as *C1*.
      ii. $B(\alpha_i) \neq dv_i + r_i$ and Signer broadcasted the `OK` message during the protocol Ver— we call this as condition as *C2*.
   If $P_i \in \mathcal{R}$ and neither *C1* nor *C2* holds, then broadcast the message `Reject`.
   (c) If the `Accept` message is received from the broadcast of $t+1$ different verifiers in the set $\mathcal{R}$, then output Reveal$_i = \vec{S^\star}$, where $\vec{S^\star}$ is the vector of ℓ values of the polynomial $F^\star(x)$ at $x = \beta_1, \ldots, \beta_\ell$.
   (d) If the `Reject` message is received from the broadcast of $t + 1$ different verifiers in $\mathcal{R}$, then output Reveal$_i = \perp$.

then there exists a unique, non-zero, random d, for which $B(\alpha_i) = dv_i + r_i$ will hold. However, a corrupted Signer while distributing the tuple (α_i, v_i, r_i) to an honest $P_i \in \mathcal{R}$ during the protocol Gen, has no idea (other than guessing) about the random d, which will be selected by the honest INT only during the protocol Ver (when (α_i, v_i, r_i) has been already delivered to P_i). □

Lemma 3 (AICP-Correctness3) *If* Signer *is honest and* INT *possesses* ICSig(Signer, INT, $\mathcal{P}$, $\vec{S}$) *at the end of* Ver, *then the probability that an honest verifier P_i outputs* Reveal$_i$ $= \vec{S^\star}$ *at the end of* RevealPublic, *where $\vec{S^\star} \neq \vec{S}$ is ϵ.*

PROOF: First of all we have to consider a *corrupted* INT, as an honest INT always correctly reveals ICSig(Signer, INT, $\mathcal{P}$, $\vec{S}$). Now in order that an honest verifier $P_i \in \mathcal{P}$ outputs Reveal$_i$ $= \vec{S^\star}$ at the end of RevealPublic, where $\vec{S^\star} \neq \vec{S}$, it must be the case that INT revealed incorrect ICSig during RevealPublic. More specifically, INT must have broadcasted an incorrect polynomial $F^\star(x)$ during RevealPublic, which when evaluated at $x = \beta_1, \ldots, \beta_\ell$ gives the elements of $\vec{S^\star}$ as the output. We now claim that if INT does so, then except with probability ϵ, every *honest* verifier $P_i \in \mathcal{R}$ will broadcast the `Reject` message during RevealPublic. As there can be *at most* t corrupted verifiers in the set $\mathcal{R}$ (who may broadcast the `Accept` message in response to the incorrect polynomial), no honest verifier in the set $\mathcal{P}$ will output $\vec{S^\star}$. We consider the following two cases, depending upon what the honest Signer broadcasts during Ver:

1. Signer *broadcasts the polynomial $F(x)$ during* Ver: This implies that $B(\alpha_i) = dv_i + r_i$ does not hold for *all* the verifiers $P_i \in \mathcal{R}$ during Ver, otherwise Signer would have broadcast the `OK` message. So clearly the condition

C2 will not hold for any honest verifier $P_i \in \mathcal{R}$. So the only condition in which an honest verifier $P_i \in \mathcal{R}$ would broadcast the `Accept` message is that the condition *C1*, namely $F^\star(\alpha_i) = v_i$ holds for P_i. However, $F^\star(x) \neq F(x)$ and the corrupted INT will have no information about α_i, as both Signer and P_i are honest. Moreover, α_i's are randomly selected from $\mathbb{F} \setminus \{\beta_1, \ldots, \beta_\ell\}$. So the probability that INT can ensure $F^\star(\alpha_i) = v_i = F(\alpha_i)$ is the same as the probability that INT can correctly guess α_i, which is at most $\frac{\ell+t}{|\mathbb{F}-\ell|} \approx 2^{-\Omega(\kappa)} \approx \epsilon$ (since $F^\star(x)$ and $F(x)$ can have the same value for at most $\ell+t$ values of x).

2. Signer *broadcasts the* `OK` *message during* Ver: This implies that $B(\alpha_i) = dv_i + r_i$ holds for *all* the verifiers $P_i \in \mathcal{R}$, otherwise Signer would have broadcast the $F(x)$ polynomial. We show that in this case, the conditions under which an *honest* verifier $P_i \in \mathcal{R}$ would broadcast the `Accept` message (in response to the polynomial $F^\star(x) \neq F(x)$) during RevealPublic are either impossible or may happen with probability at most ϵ:
   (a) $F^\star(\alpha_i) = v_i = F(\alpha_i)$: As discussed above, this can happen with probability at most ϵ.
   (b) $B(\alpha_i) \neq dv_i + r_i$ *and* Signer *broadcasted the* `OK` *message during* Ver: This case is never possible because if $B(\alpha_i) \neq dv_i + r_i$, then an honest Signer would have broadcasted $F(x)$ during Ver, which is a contradiction. □

Lemma 4 (AICP-Secrecy) *If* Signer *and* INT *are honest, then the information received by* Adv *till the end of* Ver *is distributed independently of the secret* $\vec{S}$.

PROOF: First of all notice that if Signer and INT are honest, then Signer will broadcast the `OK` message during Ver. Now without loss of generality, let the verifiers $P_1, \ldots, P_t \in \mathcal{P}$ be under the control of Adv. So at the end of Ver, Adv will know d and the polynomial $B(x) = dF(x) + R(x)$, as they are broadcasted. In addition, Adv will also know α_i and $v_i = F(\alpha_i), r_i = R(\alpha_i)$, for $i = 1, \ldots, t$. The adversary Adv can also compute $B(\beta_1), \ldots, B(\beta_\ell)$, from which he gets $dF(\beta_1)+R(\beta_1), \ldots, dF(\beta_\ell)+R(\beta_\ell)$. However, the degree of the polynomials $F(x)$ and $R(x)$ is at most $\ell+t$ and the two polynomials are independent of each other. Now it is easy to see that $d, F(\alpha_1), \ldots, F(\alpha_t), R(\alpha_1), \ldots R(\alpha_t)$, $dF(\beta_1) + R(\beta_1), \ldots, dF(\beta_\ell) + R(\beta_\ell)$ has distribution independent of $F(\beta_1) = s^1, \ldots, F(\beta_\ell) = s^\ell$. This is because from the adversary's point of view, for every possible choice of the $R(x)$ polynomial of degree $\ell + t$, which is consistent with $r_1 = R(\alpha_1), \ldots, r_t = R(\alpha_t)$ and $dF(\beta_1) + R(\beta_1), \ldots, dF(\beta_\ell) + R(\beta_\ell)$, there exists a unique polynomial $F(x)$ of degree $\ell+t$, consistent with $v_1 = F(\alpha_1), \ldots, v_t = F(\alpha_t)$ and $dF(\beta_1) + R(\beta_1), \ldots, dF(\beta_\ell) + R(\beta_\ell)$. □

Theorem 2 *Protocol* Gen *requires a private communication of* $\mathcal{O}((\ell + n) \log \frac{1}{\epsilon})$ *bits. Protocol* Ver *requires broadcast of* $\mathcal{O}((\ell + n) \log \frac{1}{\epsilon})$ *bits and private communication of* $\mathcal{O}(n \log n)$ *bits. Protocol* RevealPublic *requires broadcast of* $\mathcal{O}((\ell + n) \log \frac{1}{\epsilon})$ *bits.*

PROOF: In the protocol Gen, Signer privately sends $\ell + t$ field elements to INT and three field elements to each verifier. Since each field element can be represented by $\kappa = \mathcal{O}(\log \frac{1}{\epsilon})$ bits, Gen incurs a private communication of $\mathcal{O}((\ell+n) \log \frac{1}{\epsilon})$ bits. In the protocol Ver, every verifier privately sends the message `(Received, ⋆)` to INT, thus incurring a private communication of $\mathcal{O}(n \log n)$ bits (assuming that the identity of each party can be represented by $\log n$ bits). In addition, INT broadcasts the polynomial $B(x)$ containing $\ell+t$ field elements, while Signer may also broadcast the polynomial $F(x)$ of degree at most $\ell + t$, thus incurring a broadcast of $\mathcal{O}((\ell + n) \log \frac{1}{\epsilon})$ bits. In the protocol RevealPublic, INT broadcasts ICSig, which is a polynomial of degree at most $\ell + t$, consisting of $\ell + t$ field elements, while each verifier in $\mathcal{R}$ broadcasts `Accept`/`Reject` message. So RevealPublic involves broadcast of $\mathcal{O}((\ell+n) \log \frac{1}{\epsilon})$ bits. □

In the rest of the paper, we will use the following notations while using the protocol MultiVerifierAICP.

Notation 2 (Notation for Using the Gen, Ver **and** Reveal Public **Protocols)** Recall that Signer and INT can be any party from the set $\mathcal{P}$. We say that:

1. "Party P_i gives $\mathsf{ICSig}(P_i, P_j, \mathcal{P}, \vec{S})$ to party P_j" to mean that P_i as a Signer executes the protocol $\mathsf{Gen}(P_i, P_j, \mathcal{P}, \vec{S}, \epsilon)$, considering P_j as an INT to give P_i's IC signature on $\vec{S}$ to P_j.
2. "Party P_i receives $\mathsf{ICSig}(P_j, P_i, \mathcal{P}, \vec{S})$ from the party P_j" to mean that P_i as an INT has completed the protocol $\mathsf{Ver}(P_j, P_i, \mathcal{P}, \vec{S}, \epsilon)$ and finally possesses $\mathsf{ICSig}(P_j, P_i, \mathcal{P}, \vec{S})$, where P_j is Signer.
3. "Party P_i reveals $\mathsf{ICSig}(P_j, P_i, \mathcal{P}, \vec{S})$" to mean that P_i as an INT executes the protocol $\mathsf{RevealPublic}(P_j, P_i, \mathcal{P}, \vec{S}, \epsilon)$ along with the participation of the verifiers in $\mathcal{P}$ to reveal $\mathsf{ICSig}(P_j, P_i, \mathcal{P}, \vec{S})$ and hence $\vec{S}$, where P_j is Signer.
4. "Party P_k completes the revelation of $\mathsf{ICSig}(P_j, P_i, \mathcal{P}, \vec{S})$ with output $\mathsf{Reveal}_k = \vec{S^\star}$ (resp. $\mathsf{Reveal}_k = \perp$)" to mean that P_k as a verifier has completed the protocol $\mathsf{RevealPublic}(P_j, P_i, \mathcal{P}, \vec{S}, \epsilon)$, where P_j is Signer and P_i is INT, with output $\mathsf{Reveal}_k = \vec{S^\star}$ (resp. $\mathsf{Reveal}_k = \perp$).
5. "Party P_i successfully/correctly revealed $\mathsf{ICSig}(P_j, P_i, \mathcal{P}, \vec{S})$ (and hence $\vec{S}$)" to mean that every honest verifier $P_k \in \mathcal{P}$ outputs $\mathsf{Reveal}_k = \vec{S}$ after completing the protocol $\mathsf{RevealPublic}(P_j, P_i, \mathcal{P}, \vec{S}, \epsilon)$, where P_i is INT and P_j is Signer.
6. "Party P_i failed to reveal $\mathsf{ICSig}(P_j, P_i, \mathcal{P}, \vec{S})$ (and hence $\vec{S}$)" to mean that every honest verifier $P_k \in \mathcal{P}$ outputs

$\mathsf{Reveal}_k = \perp$ after completing the protocol RevealPublic$(P_j, P_i, \mathcal{P}, \vec{S}, \epsilon)$, where P_i is INT and P_j is Signer.

4 Asynchronous Weak Commitment (AWC)

We now present our AWC scheme. For the ease of presentation, we will first present an AWC scheme which allows to commit and decommit a single secret. This will be followed by the description of the modifications required to make the scheme to deal with ℓ secrets concurrently. We will then compare our AWC scheme with the existing AWSS and W-SVSS schemes. Finally, we discuss an important interpretation of our AWC for committing and decommitting polynomials, instead of committing and decommitting a secret. The interpretation will be very helpful when we use our AWC scheme for designing our AVSS.

4.1 AWC Scheme for a Single Secret

We present an AWC scheme called AWC $-$ Single, consisting of a pair of protocols (Com, DeCom), which allows a Committer $\in \mathcal{P}$ to commit to a secret $s \in \mathbb{F}$ in a distributed fashion among the parties in $\mathcal{P}$. The high level idea of the protocol is as follows: During the protocol Com, Committer computes n Shamir shares [30] for the secret s, with threshold t. Specifically, Committer selects a random polynomial of degree at most t, subject to the condition that the constant term of the polynomial is the secret s. Let $Sh_1, \ldots, Sh_n$ be the n shares, which are the evaluation of the selected polynomial at n publicly known distinct values. Then Committer sends the ith share Sh_i to the party P_i and also gives his IC signature on Sh_i (as a Signer) to the party P_i (considering P_i as an INT). On receiving the IC signature on Sh_i from Committer, the party P_i then himself acts as a Signer and gives back P_i's IC signature on the *same* share Sh_i to Committer (considering Committer as an INT). On receiving back $2t+1$ IC signatures from at least $2t+1$ parties, Committer broadcasts the identities of these $2t+1$ parties (we denote these parties by the set WCORE) and every (honest) party terminates the protocol Com on receiving WCORE from Committer. The interpretation is that Committer has committed the secret to the parties in WCORE, where the committed secret is determined by the shares of the (honest) parties in WCORE. That is, the committed secret is the constant term of the polynomial, defined by the shares of the honest parties in WCORE. If the polynomial is of degree at most t, then we say that the committed secret is from $\mathbb{F}$, otherwise $\perp$ is the committed secret.

If Committer is *honest* then the protocol Com preserves the privacy of the secret s; this follows from the privacy of the Shamir secret sharing with threshold t and the secrecy property of the AICP. Moreover, Committer will eventually broadcast the set WCORE, as there are at least $2t+1$ honest parties in the set $\mathcal{P}$ who will give back their IC signatures on their respective shares to Committer and so each honest party will eventually terminate the protocol Com. Furthermore, the committed secret s will be from $\mathbb{F}$, as the shares of the honest parties in WCORE will define a polynomial of degree at most t. Now consider the case when Com is *corrupted*. In that case, he may not distribute "consistent" Shamir shares to the honest parties in WCORE. More specifically, the shares given to the honest parties in the set WCORE may not lie on a unique polynomial of degree at most t, so the committed secret may be $\perp$. However, we will show that this will not cause any problem later in the protocol DeCom; that is, if the corrupted Committer later tries to decommit $s^\star \neq \perp$, then no honest party will accept the decommitted value.

In the protocol DeCom, Committer reveals the IC signature of *all* the $2t+1$ parties in the set WCORE on their respective shares. So the participation of Committer is very crucial in the protocol DeCom, as otherwise, the honest parties will never participate in the protocol DeCom. This implies that if Committer is corrupted (and does not start revealing the signatures) then no (honest) party will participate (and hence terminate) in DeCom. Now if all the signatures (on the shares) are revealed correctly by Committer and the $2t+1$ shares lie on a unique polynomial of degree at most t, then the constant term of the polynomial is output as the decommitted secret; otherwise the parties output $\perp$.

It is easy to see that if Committer is *honest* then indeed every honest party will output s as the decommitted secret, where s was committed during Com. This is because Committer will successfully reveal the IC signature of the $2t+1$ parties in WCORE, on the corresponding Shamir shares; moreover all these shares will lie on the original polynomial of degree at most t. On the other hand, if Committer is *corrupted* and $s^\star$ is output as the decommitted secret then with high probability, $s^\star$ was committed during Com among the (honest) parties in WCORE. This is because a corrupted Committer will fail to reveal the IC signature of an *honest* party $P_i \in$ WCORE on an incorrect share Sh_i', which was not given to the ith party during Com. So if at all $s^\star$ is output as the decommitted secret, then it implies that during the protocol Com, the constant term of the polynomial defined by the shares of the honest parties in WCORE was $s^\star$.

Notice that the IC signatures given by Committer to the respective parties in WCORE are not used at all in the protocol Com and DeCom. In fact, it is enough for Committer to get the IC signatures from the parties in WCORE to design the AWC scheme. However, the signatures given by Committer to the parties in WCORE will play a very crucial role, when we will use the AWC scheme to design our AVSS scheme. The formal details of the protocol AWC $-$ Single are given in Fig 3.

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

$\mathsf{AWC} - \mathsf{Single}(\mathsf{Committer}, \mathcal{P}, s, \epsilon)$

$\mathsf{Com}(\mathsf{Committer}, \mathcal{P}, s, \epsilon)$

All instances of Gen and Ver in the following protocol are executed with error parameter $\epsilon' = \frac{\epsilon}{n}$ to bound the error probability of Com by ϵ.

GENERATING THE COMMITMENT INFORMATION : The following code is executed only by Committer:

1. Select a random polynomial $f(x)$ over $\mathbb{F}$ of degree at most t, such that $f(0) = s$ and for $i = 1, \ldots, n$, compute the ith share $Sh_i = f(i)$.
2. For $i = 1, \ldots, n$, give $\mathsf{ICSig}(\mathsf{Committer}, P_i, \mathcal{P}, Sh_i)$ to P_i by acting as a Signer and considering P_i as an INT.
3. For $i = 1, \ldots, n$, if $\mathsf{ICSig}(P_i, \mathsf{Committer}, \mathcal{P}, Sh_i')$ is received from P_i, such that $Sh_i' = Sh_i$ and the message (`Sign-Sent`, i, Committer) is received from the broadcast of P_i, then include P_i in a set dynamic set WCORE, which is initially $\emptyset$.
4. Wait till $|\mathsf{WCORE}| = 2t + 1$ and then broadcast the set WCORE and terminate.

SIGNING THE SHARES, VERIFYING Committer'S CLAIM AND TERMINATION : For $i = 1, \ldots, n$, every party $P_i \in \mathcal{P}$ (including Committer) executes the following code:

1. On receiving $\mathsf{ICSig}(\mathsf{Committer}, P_i, \mathcal{P}, Sh_i)$ from Committer, act as a Signer and give back $\mathsf{ICSig}(P_i, \mathsf{Committer}, \mathcal{P}, Sh_i)$ to Committer, considering Committer as an INT. In addition, broadcast the message (`Sign-Sent`, i, Committer) to notify that signature is given to Committer.
2. Wait to receive a set WCORE of size $2t+1$ from the broadcast of Committer. On receiving WCORE, wait to receive the message (`Sign-Sent`, j, Committer) from the broadcast of every $P_j \in$ WCORE. On receiving all these messages, terminate the protocol.

$\mathsf{DeCom}(\mathsf{Committer}, \mathcal{P}, s, \epsilon)$

All instances of RevealPublic in the following protocol are executed with error parameter $\epsilon' = \frac{\epsilon}{n}$ to bound the error probability of DeCom by ϵ.

DECOMMITTING THE SECRET : The following code is executed only by Committer:

1. Act as an INT and reveal $\mathsf{ICSig}(P_j, \mathsf{Committer}, \mathcal{P}, Sh_j)$, corresponding to each $P_j \in$ WCORE.

VERIFYING THE DECOMMITMENT AND TERMINATION : For $i = 1, \ldots, n$, every party $P_i \in \mathcal{P}$ (including Committer) executes the following code:

1. Act as a verifier and wait to complete the revelation of $\mathsf{ICSig}(P_j, \mathsf{Committer}, \mathcal{P}, Sh_j)$, corresponding to each $P_j \in$ WCORE.
2. If $\exists P_j \in$ WCORE, such that P_i completes the revelation of $\mathsf{ICSig}(P_j, \mathsf{Committer}, \mathcal{P}, Sh_j)$ with output $\mathsf{Reveal}_i = \perp$ then output $\perp$ and terminate.
3. If P_i completes the revelation of $\mathsf{ICSig}(P_j, \mathsf{Committer}, \mathcal{P}, Sh_j)$ with output $\mathsf{Reveal}_i = Sh_j$, then do the following:
   (a) If the points $\{(j, Sh_j) : P_j \in \mathsf{WCORE}\}$ lie on a unique polynomial, say $f^\star(x)$, of degree at most t, then output $s^\star = f^\star(0)$ and terminate.
   (b) If the points $\{(j, Sh_j) : P_j \in \mathsf{WCORE}\}$ do not lie on a unique polynomial of degree at most t, then output $\perp$ and terminate.

We now prove the properties of the protocol $\mathsf{AWC} - \mathsf{Single}$.

Lemma 5 (Termination) *Protocols* $(\mathsf{Com}, \mathsf{DeCom})$ *satisfy the termination condition of definition 3.*

PROOF: It is easy to see that if Committer is honest then eventually every honest party P_i will give $\mathsf{ICSig}(P_i, \mathsf{Committer}, \mathcal{P}, Sh_i')$ to Committer, where $Sh_i' = Sh_i$ and there are at least $2t+1$ such honest parties. So eventually, Committer will broadcast a set WCORE and by the properties of broadcast, every honest party will eventually receive the set. Moreover, the honest Committer also must have received the message (`Sign-Sent`, i, Committer) from the broadcast of every party $P_i \in$ WCORE, before including P_i in the set WCORE and so from the properties of broadcast every other honest party will also eventually receive these messages and will terminate the protocol Com. This proves the first requirement.

Let P_i be an honest party who has terminated Com. Thus P_i has received the set WCORE of size $2t + 1$ from the broadcast of Committer, along with the message (`Sign-Sent`, j, Committer) from the broadcast of every $P_j \in$ WCORE. Now from the properties of broadcast, every other honest party will also eventually receive this set and the corresponding messages and will terminate Com. This proves the second requirement.

Let Committer invokes the protocol DeCom. This implies that corresponding to each $P_j \in$ WCORE, Committer reveals the signature $\mathsf{ICSig}(P_j, \mathsf{Committer}, \mathcal{P}, Sh_j)$. Now eventually all these signatures will be revealed. That is, each honest party P_i will complete the revelation of all these signatures with some output, which may be either $\perp$ or some value. Now irrespective of these outputs (of the signature revelation), each honest party will either output a value from $\mathbb{F}$ or $\perp$ and terminate the protocol DeCom. This proves the third requirement. □

Lemma 6 (Correctness) *Protocols* $(\mathsf{Com}, \mathsf{DeCom})$ *satisfy the correctness condition of definition 3.*

PROOF: Let P_i be an *honest* party who terminated the protocol Com. This implies that P_i received the set WCORE of size $2t + 1$ from the broadcast of Committer, along with the messages (`Sign-Sent`, j, Committer) from the broad-

cast of every (honest) $P_j \in$ WCORE. This further implies that Committer has given the shares of the secret to the (honest) parties in WCORE; otherwise Committer will not receive the signature of those honest parties on the corresponding shares and those honest parties would not have broadcast the corresponding (`Sign-Sent, j,` Committer) message. Now out of the $2t+1$ parties in the set WCORE, at most t can be corrupted. We define the committed secret $\overline{s}$ as follows: if the shares of the honest parties in WCORE lie on a unique polynomial of degree at most t, say $\overline{f}(x)$, then $\overline{s} = \overline{f}(0)$, otherwise $\overline{s} = \perp$. Note that $\overline{s}$ is well defined, as there are at least $t+1$ honest parties in WCORE.

Now if Committer is *honest*, then during the protocol DeCom, Committer will successfully reveal all the signatures corresponding to every party in WCORE, except with probability at most ϵ. That is, corresponding to every *honest* $P_j \in$ WCORE, Committer (as an INT) will successfully reveal ICSig$(P_j, \text{Committer}, \mathcal{P}, Sh_j)$; this follows from **AICP-Correctness1**. On the other hand, corresponding to every *corrupted* $P_j \in$ WCORE, Committer (as an INT) will successfully reveal ICSig$(P_j, \text{Committer}, \mathcal{P}, Sh_j)$, except with probability $\epsilon' = \frac{\epsilon}{n}$; this follows from **AICP-Correctness2**. As there can be at most t corrupted parties in WCORE, except with probability at most $t\epsilon' \approx \epsilon$, all the $2t+1$ signatures (corresponding to the parties in WCORE) will be correctly revealed by Committer. Moreover, the points $\{(j, Sh_j) : P_j \in \text{WCORE}\}$ will define the original polynomial $f(x)$ of degree at most t, selected during Com. So each honest party will output $s = f(0)$. Furthermore, it is easy to see that $s = \overline{s}$, as $f(x) = \overline{f}(x)$. This proves the second requirement of the correctness property.

Now consider a *corrupted* Committer and let P_i be an *honest* party, who outputs $s^\star \in \mathbb{F}$ in the protocol DeCom. This implies that corresponding to every $P_j \in$ WCORE, party P_i completed the revelation of ICSig$(P_j, \text{Committer}, \mathcal{P}, Sh_j)$ with output $\text{Reveal}_i = Sh_j$. Moreover, the points $\{(j, Sh_j) : P_j \in \text{WCORE}\}$ lie on the polynomial $f^\star(x)$ of degree at most t, where $s^\star = f^\star(0)$. We first claim that every other *honest* party will also eventually complete the revelation of ICSig$(P_j, \text{Committer}, \mathcal{P}, Sh_j)$ with output Sh_j. This follows from the steps of the protocol RevealPublic. We next claim that except with probability ϵ, the polynomial $f^\star(x)$ is the same as the polynomial $\overline{f}(x)$. This follows from the fact that except with probability $\epsilon' = \frac{\epsilon}{n}$, a corrupted Committer will fail to correctly reveal ICSig$(P_j, \text{Committer}, \mathcal{P}, Sh'_j)$ on an arbitrary Sh'_j, corresponding to an *honest* $P_j \in$ WCORE (follows from **AICP-Correctness3**). Moreover, there are at least $t+1$ such honest P_j's in the set WCORE. So except with probability $(t+1)\epsilon' \approx \epsilon$, the signed shares corresponding to the honest parties in WCORE which are revealed by Committer are the same shares which were given to those honest parties during Com. Now there exists only a single polynomial of degree at most t, consistent with the shares of all the honest parties in WCORE. This proves the first requirement of the correctness property. □

Lemma 7 (Secrecy) *If* Committer *is honest then the information received by* Adv *till the end of* Com *is distributed independently of the secret* s.

PROOF: The proof follows from the properties of the Shamir secret sharing and the **AICP-Secrecy**. More specifically, without loss of generality, let $P_1, \ldots, P_t$ be under the control of Adv. So Adv will know t points (namely the shares) on the polynomial $f(x)$, which is a random polynomial of degree at most t. So from the view point of the adversary, for every possible secret $\overline{s}$, there exists a unique polynomial of degree at most t with $\overline{s}$ as the constant term and which is consistent with his t shares and so all possible secrets are equiprobable from his view point. Moreover, the adversary will not get any extra information about the remaining shares (of the honest parties) from the instances of AICP, which are used to generate the IC signature on these shares. That is, corresponding to every honest P_i, the adversary obtains no information about the share $Sh_i = f(i)$ during the instances Gen(Committer, $P_i, \mathcal{P}, Sh_i$), Ver(Committer, $P_i, \mathcal{P}, Sh_i$), Gen(P_i, Committer, $\mathcal{P}, Sh_i$) and Ver(P_i, Committer, $\mathcal{P}, Sh_i$), which are invoked during Com to generate Committer's and P_i's IC signature on Sh_i (this follows from Lemma 4). □

Theorem 3 *Protocols* (Com, DeCom) *is a* $(1-\epsilon)$*-AWC scheme. Protocol* Com *requires a private communication of* $\mathcal{O}(n^2 \log \frac{1}{\epsilon})$ *bits and broadcast of* $\mathcal{O}(n^2 \log \frac{1}{\epsilon})$ *bits. Protocol* DeCom *requires a broadcast of* $\mathcal{O}(n^2 \log \frac{1}{\epsilon})$ *bits.*

PROOF: In the protocol Com, at most $2n$ instances of Gen and Ver, each dealing with a single value are executed to generate the IC signatures. During DeCom, at most n instances of RevealPublic, each dealing with a single value are executed to reveal the signatures. The proof now follows from Theorem 2 by substituting $\ell = 1$ and from Lemma 5-7. □

4.2 AWC Scheme for ℓ Secrets

The AWC scheme presented in the last section allows to commit and later decommit a single secret. Now consider a scenario where a Committer would like to commit and later decommit a vector $\vec{S} = (s^1, \ldots, s^\ell)$, consisting of ℓ elements from $\mathbb{F}$, where $\ell > 1$. One simple way of doing this is to execute an instance of the Com and DeCom protocol on the behalf of each element $s^l \in \vec{S}$; this will require a private as well as broadcast communication of $\mathcal{O}(\ell n^2 \log \frac{1}{\epsilon})$ bits. However, we now show how to commit and decommit *all* the ℓ elements of $\vec{S}$ concurrently, so that it requires a private as well as broadcast communication of $\mathcal{O}((\ell n + n^2) \log \frac{1}{\epsilon})$ bits. So if $\ell = \Omega(n)$, which will be the case in

our AVSS scheme and the common coin protocol, then the broadcast communication of our protocol will be *independent* of ℓ. This will be a significant gain in the communication complexity, considering the fact that implementing broadcast through the A-cast protocol over a point to point network is an expensive operation[11].

The key idea here is that we extend the protocol Com and DeCom in a "natural" way, so as to deal with ℓ values concurrently. That is, the instances of Gen, Ver and RevealPublic in the Com and DeCom protocol are now invoked to deal with ℓ values concurrently, instead of a single value. For example, on the behalf of each party P_i, Committer will compute ℓ Shamir shares (one corresponding to each of the ℓ secrets). Now instead of executing ℓ different instances of Gen (each dealing with a single value) to give his IC signature on the ℓ ith share, Committer will execute a single instance of Gen to give his IC signature on all the ℓ ith share concurrently. The other steps are modified and extended similarly. The modified protocols are presented in Fig 4. The new scheme is called AWC $-$ Multiple, as it deals with multiple secrets. The properties of the modified scheme follow using the same arguments as for the earlier scheme and so we avoid giving the complete proofs to avoid repetition.

We state the following theorem, stating the communication complexity of the protocol AWC $-$ Multiple, whose proof follows from the properties of the protocol and the communication complexity of our AICP (Theorem 2).

Theorem 4 *In* AWC $-$ Multiple*, protocol* Com *requires a private communication of* $\mathcal{O}((\ell n+n^2)\log\frac{1}{\epsilon})$ *bits and broadcast of* $\mathcal{O}((\ell n+n^2)\log\frac{1}{\epsilon})$ *bits. Protocol* DeCom *requires a broadcast of* $\mathcal{O}((\ell n+n^2)\log\frac{1}{\epsilon})$ *bits.*

4.3 Comparison of Our AWC with the AWSS of [25] and W-SVSS of [1]

The best known AWSS scheme was presented in [25]. The scheme is an $(1-\epsilon)$-AWSS scheme and based on the idea of using a bi-variate polynomial to share a secret. Specifically, to share a secret s, a random bi-variate polynomial $F(x,y)$ with s as the constant term is used and each party receives n points on this polynomial (along with additional information like the IC signatures). So this approach inherently requires the distribution of $\Omega(n^2)$ elements from $\mathbb{F}$ to share a single secret. On the other hand, in our AWC scheme, to commit a single secret, only n elements from $\mathbb{F}$ need to be distributed, as the secret is committed using a univariate polynomial (ignoring the IC signatures). This automatically suggests a gain of $\Omega(n)$ in the communication complexity.

The weak-shunning VSS (W-SVSS) of [1], which may be considered as a "shunning" variant of AWSS, is also based on the idea of using a bi-variate polynomial of degree t in each variable to share the secret (however, it does not use any IC signature) and so it also requires distributing $\Omega(n^2)$ elements from the underlying field to share a single secret. Moreover, W-SVSS does not satisfy all the properties of AWSS. Specifically, if all the parties behave honestly during the protocol then we get the same properties as in an AWSS scheme; otherwise the protocol does not provide the properties of an AWSS, but ensures that there exists at least one *honest* party, who will identify at least one *corrupted* party, who behaved dis-honestly and whom the honest party will shun (ignore) from then onwards for the rest of the protocol execution. However, unlike the AWSS scheme of [25] and our AWC scheme, both of which are $(1-\epsilon)$-terminating, the W-SVSS scheme of [1] is al most surely terminating (i.e. terminates with probability one).

4.4 AWC for Sharing Polynomials

There is another interesting way to interpret the computation done in the protocol AWC $-$ Single and AWC $-$ Multiple; we will be using this interpretation while using our AWC for our AVSS. For simplicity, we focus on the protocol AWC $-$ Single (Fig 3); the same discussion even holds for the protocol AWC $-$ Multiple (Fig 4). Recall that in the protocol Com in AWC $-$ Single, in order to commit a secret s, Committer selected a polynomial $f(x)$ of degree at most t with s as the constant term and shared s through this polynomial by giving a point on the polynomial to each party as a share. We defined the committed value $\overline{s}$ as follows: we considered the polynomial $\overline{f}(x)$ defined by the shares of the *honest* parties in WCORE; if $\overline{f}(x)$ has degree at most t, then $\overline{s}=\overline{f}(0)\in\mathbb{F}$; otherwise $\overline{s}=\perp$. Moreover, if Committer is honest then $f(x)=\overline{f}(x)$ (see the proof of Lemma 6). We can view this entire computation (during the protocol Com) as follows:

- Committer has committed a *polynomial* $\overline{f}(x)$ among the (honest) parties in WCORE, where $|\mathsf{WCORE}|=2t+1$. Each (honest) party $P_j\in\mathsf{WCORE}$ has the share $Sh_j=\overline{f}(j)$ and the IC signature of Committer on the share Sh_j, namely $\mathsf{ICSig}(\mathsf{Committer},P_j,\mathcal{P},Sh_j)$. In addition, Committer will have the IC signature of each $P_j\in$ WCORE on the share Sh_j, namely $\mathsf{ICSig}(P_j,\mathsf{Committer},\mathcal{P},Sh_j)$. We call the signatures given by Committer (as a Signer) to the parties in WCORE (as an INT) as the *secondary signatures*, while the signatures given by the parties in WCORE (as a Signer) to Committer (as an INT) are called the *primary signatures*. The reason for distinguishing these two types of signatures will be clear when we use the AWC scheme in our AVSS scheme. Notice that if Committer is *honest*, then corresponding to *each* $P_j\in$ WCORE, the primary as well as the sec-

[11] Recall that an instance of A-cast requires a private communication of $\mathcal{O}(\ell n^2)$ bits to broadcast an ℓ bit message.

Fig. 4 AWC for ℓ secrets with $n = 3t + 1$.

$\mathsf{AWC} - \mathsf{Multiple}(\mathsf{Committer}, \mathcal{P}, \vec{S} = (s^1, \ldots, s^\ell), \epsilon)$

$\mathsf{Com}(\mathsf{Committer}, \mathcal{P}, \vec{S} = (s^1, \ldots, s^\ell), \epsilon)$

All instances of Gen and Ver in the following protocol are executed with error parameter $\epsilon' = \frac{\epsilon}{n}$ to bound the error probability of Com by ϵ.

GENERATING THE COMMITMENT INFORMATION : The following code is executed only by Committer:

1. For $l = 1, \ldots, \ell$, corresponding to the secret $s^l \in \vec{S}$, select a random polynomial $f_l(x)$ over $\mathbb{F}$ of degree at most t, such that $f_l(0) = s^l$.
2. For $i = 1, \ldots, n$, compute the vector of ith share $\vec{Sh}_i = (f_1(i), \ldots, f_\ell(i))$.
3. For $i = 1, \ldots, n$, give $\mathsf{ICSig}(\mathsf{Committer}, P_i, \mathcal{P}, \vec{Sh}_i)$ to P_i by acting as a Signer and considering P_i as an INT.
4. For $i = 1, \ldots, n$, if $\mathsf{ICSig}(P_i, \mathsf{Committer}, \mathcal{P}, \vec{Sh}'_i)$ is received from P_i, such that $\vec{Sh}'_i = \vec{Sh}_i$, and the message (`Sign-Sent`, i, Committer) is received from the broadcast of P_i, then include P_i in a dynamic set WCORE, which is initially $\emptyset$.
5. Wait till $|\mathsf{WCORE}| = 2t + 1$ and then broadcast the set WCORE and terminate.

SIGNING THE SHARES, VERIFYING Committer'S CLAIM AND TERMINATION : For $i = 1, \ldots, n$, every party $P_i \in \mathcal{P}$ (including Committer) executes the following code:

1. On receiving $\mathsf{ICSig}(\mathsf{Committer}, P_i, \mathcal{P}, \vec{Sh}_i)$ from Committer, act as a Signer and give back $\mathsf{ICSig}(P_i, \mathsf{Committer}, \mathcal{P}, \vec{Sh}_i)$ to Committer, considering Committer as an INT. In addition, broadcast the message (`Sign-Sent`, i, Committer) to publicly notify that signature is given to Committer.
2. Wait to receive a set WCORE of size $2t+1$ from the broadcast of Committer. On receiving WCORE, wait to receive the message (`Sign-Sent`, j, Committer) from the broadcast of every $P_j \in$ WCORE. On receiving all these messages, terminate the protocol.

$\mathsf{DeCom}(\mathsf{Committer}, \mathcal{P}, \vec{S}, \epsilon)$

All instances of RevealPublic in the following protocol are executed with error parameter $\epsilon' = \frac{\epsilon}{n}$ to bound the error probability of DeCom by ϵ.

DECOMMITTING THE SECRET : The following code is executed only by Committer:

1. Act as an INT and reveal $\mathsf{ICSig}(P_j, \mathsf{Committer}, \mathcal{P}, \vec{Sh}_j)$, corresponding to each $P_j \in$ WCORE.

VERIFYING THE DECOMMITMENT AND TERMINATION : For $i = 1, \ldots, n$, every party $P_i \in \mathcal{P}$ (including Committer) executes the following code:

1. Act as a verifier and wait to complete the revelation of $\mathsf{ICSig}(P_j, \mathsf{Committer}, \mathcal{P}, \vec{Sh}_j)$, corresponding to each $P_j \in$ WCORE.
2. If $\exists P_j \in$ WCORE, such that P_i completes the revelation of $\mathsf{ICSig}(P_j, \mathsf{Committer}, \mathcal{P}, \vec{Sh}_j)$ with output $\mathsf{Reveal}_i = \perp$ then output $\perp$ and terminate.
3. If corresponding to $P_j \in$ WCORE, party P_i completes the revelation of $\mathsf{ICSig}(P_j, \mathsf{Committer}, \mathcal{P}, \vec{Sh}_j)$ with output $\mathsf{Reveal}_i = \vec{Sh}_j$, then interpret $\vec{Sh}_j$ as $(f_1(j), \ldots, f_\ell(j))$ and do the following:
   (a) If for $l = 1, \ldots, \ell$, the points $\{(j, f_l(j)) : P_j \in \mathsf{WCORE}\}$ lie on a unique polynomial, say $f_l^\star(x)$, of degree at most t, then output $\vec{S^\star} = (f_1^\star(0), \ldots, f_\ell^\star(0))$ and terminate.
   (b) If $\exists l \in \{1, \ldots, \ell\}$, such that the points $\{(j, f_l(j)) : P_j \in \mathsf{WCORE}\}$ do not lie on a unique polynomial of degree at most t, then output $\perp$ and terminate.

ondary signature is on the *same* share Sh_j. Moreover, even if Committer is *corrupted*, the primary as well the secondary signature will be on the same share Sh_j, corresponding to *each honest* $P_j \in$ WCORE.

- If Committer is *honest*, then the polynomial $\overline{f}(x)$ will have degree at most t. However, if Committer is *corrupted* then the polynomial can have degree more than t.
- If Committer is *corrupted*, then the polynomial $\overline{f}(x)$ will be known to Adv. However, if Committer is honest, then Adv may know at most t points on $\overline{f}(x)$; moreover the information received by Adv will be distributed independently of the *secret* $\overline{s} = \overline{f}(0)$. So even though the committed polynomial $\overline{f}(x)$ will not be completely random, the secrecy of $\overline{f}(0)$ will be preserved if Committer is *honest* (see the proof of Lemma 7).

Also recall that during the protocol DeCom, to decommit the secret, Committer has to reveal *all* the *primary signatures* on the shares (the secondary signatures will be used later when we use the AWC as a black-box in our AVSS), corresponding to the parties in WCORE. If all the signatures are revealed correctly and the revealed shares lie on a polynomial of degree at most t, say $f^\star(x)$, then $s^\star = f^\star(0) \in \mathbb{F}$ is outputted as the decommitted secret; otherwise $\perp$ is outputted. Moreover, we have shown that if $f^\star(x)$ has degree at most t, then $f^\star(x)$ is the same polynomial, which was committed during the protocol Com (see Lemma 6). We can view this computation during the protocol DeCom, as if Committer decommits a polynomial $f^\star(x)$. If the polynomial has degree at most t, then all the (honest) parties output this polynomial; moreover the same polynomial was committed during Com. However, if the polynomial has degree more than t or the same polynomial was not committed during Com, then the parties output $\perp$.

We remark that the above idea of abusing the notion of "committing (decommitting) a secret" to "committing (de-

committing) a polynomial $f(x)$ of degree at most t" is not new and it is very well known and commonly used in the literature of WSS and VSS (for example, see[24, 19, 22]). This does not break the interface when AWC is further used as a black-box in our AVSS. The reason is that Committer still has to follow exactly the same steps "internally" to commit the polynomial $f(x)$, as he has to perform for committing a secret s. That is, for committing a secret s, Committer had to select a polynomial $f(x)$ of degree at most t, with the condition that $f(0) = s$; so s was the "actual" input of Committer for the protocol Com, while the polynomial $f(x)$ was the randomly generated information, which Committer used for committing s. Instead, we now view this $f(x)$ as the input of Committer and the rest of the protocol steps are the same.

Following the above discussion, in the rest of the paper, we will "abuse" the notion of committing and decommiting secrets (through AWC) and instead say that Committer commits and decommits polynomials (in the sense explained above) using the Com and DeCom protocols. More specifically, we will use the following notation:

Notation 3 (Notation for Using AWC to Commit/Decommit Polynomials) Recall that Committer can be any any party in the set $\mathcal{P}$. We say that:

1. "Committer commits ℓ polynomials $f_1(x), \ldots, f_\ell(x)$" to mean that Committer executes the protocol Com(Committer, $\mathcal{P}, (f_1(0), \ldots, f_\ell(0)), \epsilon$), where $f_1(x), \ldots, f_\ell(x)$ are used as the ℓ polynomials by Committer during the step 1 of GENERATING THE COMMITMENT INFORMATION in the protocol AWC − Multiple. If the honest parties terminate this protocol, then they will know a set WCORE of size $2t + 1$, such that each (honest) $P_j \in$ WCORE will have the secondary signature ICSig(Committer, $P_j, \mathcal{P}, \vec{Sh}_j$) and corresponding to each $P_j \in$ WCORE, Committer will have the primary signature ICSig(P_j, Committer, $\mathcal{P}, \vec{Sh}_j$). Here $\vec{Sh}_j = (f_1(j), \ldots, f_\ell(j))$ will be called the share of the committed polynomials $f_1(x)$, $\ldots, f_\ell(x)$.
2. "Committer decommits ℓ polynomials $f_1(x), \ldots, f_\ell(x)$" to mean that Committer executes the protocol DeCom(Committer, $\mathcal{P}, (f_1(0), \ldots, f_\ell(0)), \epsilon$), where during the step DECOMMITTING THE SECRET, Committer will reveal the primary signatures ICSig(P_j, Committer, $\mathcal{P}, \vec{Sh}_j$), corresponding to each $P_j \in$ WCORE, such that $\vec{Sh}_j = (f_1(j), \ldots, f_\ell(j))$. If the honest parties terminate the protocol, then they either output the polynomials $f_1(x), \ldots, f_\ell(x)$, if all these polynomials are of degree at most t and the same polynomials were committed by Committer earlier during the Com protocol; otherwise the parties output $\perp$.

5 Asynchronous Verifiable Secret Sharing (AVSS)

We now present our AVSS scheme. For the ease of presentation, we first present an AVSS scheme for sharing a single secret. This will help us to understand the underlying ideas. Later we will show how to modify the scheme so as to share ℓ secrets concurrently.

5.1 AVSS for Sharing a Single Secret

Before going into the details of the AVSS, we first closely look into our AWC scheme for sharing a single secret (namely the protocol AWC − Single) and see why it fails to satisfy the properties of AVSS. The first easy observation is that if Committer is *honest*, then the protocol AWC − Single provides all the properties of an AVSS (namely Com can be used as the sharing protocol, while DeCom can be used as the reconstruction protocol). Moreover, even if Committer is *corrupted*, the Com protocol provides all the properties of a sharing protocol of an AVSS; the problem arises during the DeCom protocol, as it may fail to satisfy two important requirements of a reconstruction protocol of an AVSS. More specifically, if Committer is *corrupted*, then the following two problems may arise during the DeCom protocol:

- The corrupted Committer may not initiate the protocol DeCom, in which case, the honest parties will never terminate the protocol. This will be a violation of the *third* requirement of the *termination* condition of AVSS, which requires that all the honest parties should terminate the reconstruction protocol, even if D (in this case Committer) is corrupted. This problem arises because the participation of Committer (namely the revelation of the primary signatures) is the key step in the DeCom protocol.
- The honest parties may output $\perp$ during the protocol DeCom, even if an $\overline{s} \in \mathbb{F}$ was committed by Committer during the Com protocol. This is because even though a corrupted Committer cannot successfully reveal (as an INT) the primary signatures of an *honest* $P_j \in$ WCORE on an "incorrect" share $Sh'_j \neq Sh_j$ (such that Sh'_j was not given as a share to P_j during the Com), the corrupted Committer can always successfully reveal the primary signatures of a *corrupted* $P_j \in$ WCORE on any arbitrary share Sh'_j of his choice; this is because in AICP, if Signer (in this case the party P_j) and INT (in this case the Committer) are *both* corrupted, then INT can successfully reveal the signature of Signer on any arbitrary value, which is "consistent" with the verification information of the verifiers. So if the revealed shares corresponding to the *corrupted* P_j's in WCORE do not lie on the polynomial defined by the shares of the honest P_j's in WCORE, then the parties end up outputting $\perp$, even though $\overline{s} \neq \perp$ was committed during Com. This will

be a violation of the *first* requirement of the *correctness* condition of AVSS, which demands the reconstruction of the committed value $\overline{s}$ during the reconstruction protocol.

To get away with these problems, we use the idea of sharing the secret using a bi-variate polynomial of degree t in each variable. This idea of sharing the secret using bi-variate polynomials is not new and has been widely used in the literature of VSS in the synchronous settings (for example, see [24,22,12,19,21] and their references). The same idea was also used in [1] to design a shunning-VSS (SVSS). We use the idea of bi-variate polynomials to design a $(1-\epsilon)$-AVSS. We would like to stress that even though the idea was also used in [1] to get an SVSS[12], our goal is different from theirs. Moreover, we use AWC (based on univariate polynomials) to design our AVSS, while [1] used W-SVSS (based on bi-variate polynomials) to design their SVSS. So even though the central idea of using the bi-variate polynomial is common in our AVSS as well as in the SVSS of [1], the protocol steps are completely different as both these primitives (AVSS and SAVSS) have different properties and accordingly, the "interface" to use AWC (resp. W-SVSS) as a black-box are also different. So our AVSS and SVSS of [1] are incomparable.

Before proceeding further, we provide a brief background about *symmetric* bi-variate polynomials, which are used in our protocol.

Bi-Variate Polynomials: A symmetric bi-variate polynomial $F(x,y)$ over $\mathbb{F}$ of degree t is a polynomial over two variables, each of degree *at most* t, where $F(x,y)$ has the following form:

$$F(x,y) = \sum_{i,j=0}^{t} r_{ij} x^i y^j \text{ and } F(l,m) = F(m,l), \forall l,m \in \mathbb{F},$$

which implies that $r_{ij} = r_{ji}$, for $i,j = 1,\ldots,t$. Notice that $F(x,y)$ has $(t+1)+t+\ldots+1 = \frac{(t+1)(t+2)}{2}$ coefficients. Let $f_i(x) \stackrel{def}{=} F(x,i)$, for $i = 1,\ldots,n$; then $f_i(x)$ is a univariate polynomial of degree at most t. Moreover, $f_i(j) = F(j,i) = f_j(i) = F(i,j)$, which follows from the symmetry of the bi-variate polynomial. Notice that the knowledge of $f_i(x)$ provides $t+1$ *distinct* points on the polynomial $F(x,y)$; i.e. given $f_i(x)$, one can compute $f_i(j) = F(j,i)$, for $j = 1,\ldots,t+1$. This immediately implies that given *any* $t+1$ *distinct* $f_i(x)$'s, one can efficiently compute $F(x,y)$, as the knowledge of $t+1$ such distinct $f_i(x)$'s provides $\frac{(t+1)(t+2)}{2}$ distinct points on $F(x,y)$, which are sufficient to interpolate $F(x,y)$.

An interesting property of the bi-variate polynomials, which will be used to prove the privacy of our AVSS is the following: let $s \in \mathbb{F}$ be the secret and $F(x,y)$ is a random, symmetric bi-variate polynomial of degree t, subject to the condition that $s = F(0,0)$. Then given only t distinct polynomials $f_i(x)$'s, where $f_i(x) = F(x,i)$, no information is revealed about s; intuitively this is because the knowledge of t such distinct $f_i(x)$'s provides $(t+1)+t+\ldots+2 = \frac{(t+1)(t+2)}{2} - 1$ points on $F(x,y)$, which is one less than the number of coefficients in $F(x,y)$. We will later formalize this intuition, while proving the properties of our AVSS. We now discuss the underlying ideas used in our AVSS. The description is quiet lengthy, but we feel that it is required to understand the issues that arise while designing the protocol and how we get rid of those issues step by step.

Informal Description of Our AVSS (Assuming Honest D): In our AVSS scheme, to share a secret s, the dealer D selects a random, symmetric bi-variate polynomial $F(x,y)$ of degree t, subject to the condition $F(0,0) = s$. For $i = 1,\ldots,n$, D sends the polynomial $f_i(x) = F(x,i)$ to the party P_i; we call $f_i(x)$ as the *share* of s for the party P_i. Moreover, for a share $f_i(x)$, we call $s_{ij} \stackrel{def}{=} f_i(j)$, as the jth *share-share* of the share $f_i(x)$, for $j = 1,\ldots,n$. So every party P_j will have a share-share of the share $f_i(x)$ of every other party P_i. This is because the jth share-share of $f_i(x)$ is $f_i(j)$, which is the same as $f_j(i)$, as $f_i(j) = f_j(i)$. So given his share $f_j(x)$, party P_j can compute the jth share-share $f_j(i) = f_i(j)$ of the ith share $f_i(x)$. This implies that Adv will have at most t share-shares of the share $f_i(x)$ of each *honest* P_i.

For simplicity, let us first assume that D is *honest*; then clearly the above distribution of information by D satisfies the *secrecy* property of AVSS, as Adv will have at most t shares, which does not provide him the complete information to know $F(0,0)$. During the reconstruction protocol, every party can reveal their share of s and given any $t+1$ *correct* shares, we can reconstruct $F(x,y)$ and hence s. However, we have to incorporate a mechanism, which would allow us to identify the correct shares; i.e. if a corrupted P_i reveals an incorrect share $f_i'(x)$, where $f_i'(x) \neq f_i(x)$, then the honest parties should be able to identify this. For this, we use the following idea: during the sharing protocol, on receiving his share $f_i(x)$, party P_i acts as a Committer and commits his share $f_i(x)$ using an instance of our AWC (this is where we use the notion of committing a polynomial through AWC). We denote the instance of Com (resp. DeCom) executed on behalf of the party P_i as Com_i (resp. DeCom_i). Notice that in Com_i, if Committer (namely P_i) is *honest*, then it reveals *no new* information about the share $f_i(x)$ to Adv. More specifically, Adv may learn at most t shares (points) of the committed polynomial $f_i(x)$ during Com_i; we call the shares of the committed polynomial $f_i(x)$ received during Com_i as the *AWC-shares* of $f_i(x)$, to distinguish it from the share-shares of $f_i(x)$. Now it is easy to see that the t AWC-

[12] Recall that SVSS does not provide all the properties of AVSS if the corrupted parties deviate from the protocol.

shares of $f_i(x)$ and the t share-shares of $f_i(x)$ obtained by Adv will be the same, namely the points $f_i(j)$, corresponding to the t P_j's under the control of Adv. So asking each party to commit his share $f_i(x)$ does not hamper the privacy of s, as now new information about $F(x, y)$ is revealed to Adv.

The honest parties will commit his share of s, however a *corrupted* party may commit an incorrect share of s. That is, if P_i is corrupted, then he may commit an incorrect share $f'_i(x)$ during Com_i, where $f'_i(x) \neq f_i(x)$. We get rid of this problem as follows: we ask a party P_j to participate in Com_i, *only if the jth AWC-share of the committed polynomial is the same as the jth share-share of the ith share*; otherwise P_j does not participate in Com_i at all. That is, during Com_i, if P_j receives the secondary signature $\mathsf{ICSig}(P_i, P_j, \mathcal{P}, Sh_j)$ from the Committer P_i, then P_j checks that Sh_j is the same as the jth share-share of $f_i(x)$ by checking $Sh_j \stackrel{?}{=} f_j(i)$, where $f_j(x)$ is the share of P_j (received from D). If the check passes, then only P_j performs the rest of the steps of the Com protocol (namely giving back the primary signature $\mathsf{ICSig}(P_j, P_i, \mathcal{P}, Sh_j)$, broadcasting the `Signature-Sent` message, etc) during the instance Com_i.

Notice that the above idea of *pre-checking* a condition before participating in an instance of Com will not cause any problem for the termination of the Com instances of the (honest) parties. This is because if P_i is *honest*, then the pre-checking will pass for every *honest* P_j during Com_i and since there are at least $2t + 1$ honest parties, eventually all the honest parties will participate in Com_i, corresponding to an honest Committer P_i. So each such Com_i will eventually terminate, as a WCORE of size $2t+1$ will be eventually constructed in such instances; we call the WCORE constructed during the instance Com_i as WCORE_i. The interesting feature of the above pre-checking is that it now ensures that if the instance Com_i, executed on the behalf of a *corrupted* P_i has terminated, then the committed polynomial is his share $f_i(x) = F(x, i)$. This is because there will be at least $t + 1$ honest parties in WCORE_i, who would have checked that the AWC-share of the committed polynomial is the same as the share-share of $f_i(x)$ and there exists a unique polynomial of degree at most t, namely $f_i(x)$, consistent with both the share-shares, as well as the AWC-shares of the honest parties in WCORE_i.

Next notice that during the sharing protocol, we cannot wait for the termination of *all* the n instances of the Com protocol, due to the fact that the instances corresponding to the t corrupted parties may never terminate. As a result, we have to terminate the sharing protocol, as soon as $2t + 1$ Com instances terminate, as otherwise it may lead to indefinite waiting. Now a subtle problem here is different parties may terminate different instances of Com at different time and it is essential that all the (honest) parties *agree* on the *same* $2t+1$ parties whose instances of Com will be eventually completed by all the parties. We solve this problem as follows: we ask D to keep "track" of all the Com instances and as soon as D completes $2t + 1$ such instances, he notifies this to everyone by broadcasting the identities of the committers of these instances and the respective WCORE_i sets.

More specifically, if D completes an instance Com_i, corresponding to the party P_i, he includes P_i in a set ShVCORE. That is, if D receives the set WCORE_i of size $2t + 1$ from the broadcast of P_i and the messages $(\texttt{Sign-Sent}, j, P_i)$ by all the P_j's in WCORE_i (which are conditions that need to be satisfied in order that D terminates the instance Com_i), then D includes P_i in the set ShVCORE. The interpretation is that ShVCORE is the set of parties whose instances of Com have been terminated by D. Now as soon as D includes $2t + 1$ parties in the set ShVCORE, he broadcasts ShVCORE, along with the WCORE_i sets corresponding to each $P_i \in$ ShVCORE, indicating that D has completed the Com instances of the $2t + 1$ parties in ShVCORE. The parties then verify that this is indeed the case. That is, each party waits to receive an ShVCORE, along with the corresponding WCORE_i's from the broadcast of D and then waits to himself complete the Com instances of each party in ShVCORE, before terminating the sharing protocol. This way, each party will eventually agree on the set of $2t + 1$ parties (along with the corresponding WCORE_i's), whose instances of Com will be completed by all the (honest) parties.

Once the honest parties terminate the sharing protocol, the reconstruction protocol may be as follows: we ask each party $P_i \in$ ShVCORE to decommit his committed share $f_i(x)$ by executing the instance DeCom_i. Notice that we cannot wait for the completion of *all* the $2t + 1$ instances of DeCom, corresponding to the $2t + 1$ parties in ShVCORE, as there may be t corrupted parties in ShVCORE, who may never execute their instance of DeCom. So during the reconstruction protocol, as soon as a party completes the DeCom instances of $t + 1$ parties from ShVCORE, with a univariate polynomial of degree t as the output, he takes those polynomials, reconstructs $F(x, y)$ using them, outputs $s = F(0, 0)$ and terminates. The property of DeCom ensures that if the output of an instance of DeCom is a polynomial of degree t, then the same polynomial was committed during the corresponding instance of Com.

Dealing with a Corrupted D: Unfortunately, all the discussion till now holds good only under the condition that D is *honest*; however if D is *corrupted*, then the only the above steps will not ensure all the properties of AVSS. This is because a corrupted D may distribute "inconsistent" shares to the honest parties; i.e. if D is corrupted, then the shares of *all* the honest parties may not lie on a *unique* bi-variate polynomial. More specifically, if there exists a unique, symmetric

bi-variate polynomial of degree t, say $\overline{F}(x, y)$, such that for every *honest* $P_i \in$ ShVCORE, it holds that $f_i(x) = \overline{F}(x, i)$ (we will often say that the share $f_i(x)$ *lies* on $\overline{F}(x, y)$ if this condition is satisfied), where $f_i(x)$ is the share of P_i, then we say that D has committed $\overline{s} = \overline{F}(0, 0)$ during the sharing protocol; otherwise we say that D has committed $\overline{s} = \perp$. Notice that $\overline{s}$ is well defined, as there will be at least $t + 1$ honest parties in ShVCORE, each holding a univariate polynomial of degree at most t as a share, which are sufficient to define a symmetric, bi-variate polynomial of degree t. Moreover, if D is *honest*, then $\overline{s} = s$, as the shares of all the parties in ShVCORE will lie on the original polynomial $F(x, y)$. Furthermore, just by following the steps discussed earlier, s will be reconstructed during the reconstruction protocol. However, if D is *corrupted* then the committed secret $\overline{s}$ may be $\perp$, but by following the steps discussed earlier, a value $s^\star$, where $s^\star \neq \perp$ may be reconstructed, which will violate the *correctness* requirement of AVSS. More specifically, if D is corrupted, then the shares of all the honest parties in ShVCORE may not lie on a unique bi-variate polynomial, which further implies that $f_i(j) \neq f_j(i)$ will hold, for at least one pair of *honest* parties (P_i, P_j). Now during the reconstruction protocol, depending upon which $t + 1$ instances of DeCom are first completed with univariate polynomials of degree t as the output, any bi-variate polynomial of degree t can be reconstructed and hence any secret can be reconstructed.

To deal with the above problem, it is enough if we can somehow allow for the decommitment of *all* the shares, committed by all the *honest* parties in ShVCORE. However it seems to be difficult because as mentioned earlier, we cannot afford to wait for the termination of all the DeCom instances, as it may cause problem for the termination condition. This is where the *secondary signatures* which were given during the Com protocol plays a crucial role. Recall that during the Com protocol, each party in WCORE holds the signature of Committer on the corresponding share, but it was not used during the DeCom protocol; we will now make use of those secondary signatures, while executing the DeCom instances in our reconstruction protocol.

Specifically, instead of just checking that there are *exactly* $2t+1$ parties in ShVCORE and there exists *some* WCOR E$_j$ of size *exactly* $2t + 1$ corresponding to each $P_j \in$ VCOR E, the parties check that there are *at least* $2t + 1$ parties in ShVCORE, i.e. $|\text{ShVCORE}| \geq 2t + 1$. In addition, corresponding to each $P_j \in$ ShVCORE, there should also exist at least $2t + 1$ parties in the set ShVCORE, who are also in WCORE$_j$, i.e. $|\text{ShVCORE} \cap \text{WCORE}_j| \geq 2t + 1$ should hold for each $P_j \in$ ShVCORE (why this is helpful will be clear in the sequel). This implies that while constructing ShVCORE, D has to ensure that the above conditions are satisfied by ShVCORE and only then be broadcasts ShVCORE during the sharing protocol. This further implies that D simply *should not terminate immediately* an instance Com$_j$ after receiving a WCORE$_j$ of size $2t + 1$ from P_j (and the corresponding `sign-sent` messages); rather D should keep "expanding" the set WCORE$_j$ in each instance Com$_j$ after receiving "new" `sign-sent` messages from new parties, which were not present in the set WCORE$_j$. Similarly, a party P_i *should not terminate immediately* an instance Com$_j$ after receiving a WCORE$_j$ of size $2t + 1$ (and the corresponding `sign-sent` messages); rather P_i should keep participating in Com$_j$, till he receives a ShVCORE from D, satisfying the new condition. This process of expansion of WCORE$_j$'s in the individual instances of Com should be continued by D, till he finds a ShVCORE of size at least $2t + 1$, satisfying the new condition.

On the first look, this process of expansion of the individual WCORE$_j$'s beyond its initial size of $2t + 1$ may look counter-intuitive; however this is very much required, otherwise even an honest D may fail to find an ShVCORE, satisfying the new conditions. More specifically, ideally if D is honest, then every honest party will be *eventually* included in the WCORE$_j$ of each honest P_j, but it is not known when will the honest parties will be included in WCORE$_j$, due to the asynchronous nature of the network. Now if D terminates an instance Com$_j$ immediately after receiving a WCOR E$_j$ of size $2t+1$, then it is not necessary that WCORE$_j$ contains only honest parties. So if D terminates *exactly* $2t + 1$ instances of Com with the corresponding (possibly different) WCORE$_j$'s of size *exactly* $2t + 1$, then the condition $|\text{ShVCORE} \cap \text{WCORE}_j| \geq 2t + 1$ may not be satisfied. Instead, if D keeps expanding each individual WCORE$_j$ and keeps checking for the above condition, then eventually he will find an ShVCORE of size *at least* $2t + 1$, satisfying the new condition. So the terminating condition for the sharing protocol is that each party should receive an ShVCORE of size at least $2t + 1$ and WCORE$_j$'s of size at least $2t + 1$ for each P_j in ShVCORE, so that the condition $|\text{ShVCORE} \cap \text{WCORE}_j| \geq 2t + 1$ is satisfied for every $P_j \in$ ShVCORE. As discussed above, if D is honest, then eventually this condition will be satisfied and the sharing protocol will terminate.

If the parties terminate the sharing protocol with an ShVCORE satisfying the above properties, then the reconstruction protocol proceeds as follows: we first ask each party $P_j \in$ ShVCORE to decommit his committed share $f_j(x)$ in the instance DeCom$_j$, for which P_j will reveal the *primary signatures* ICSig$(P_i, P_j, \mathcal{P}, f_j(i))$, corresponding to each $P_i \in$ WCORE$_j$ and we check whether all these revealed $f_j(i)$'s lie on a polynomial of degree at most t, say $\overline{f}_j(x)$. Recall that according to the protocol steps of DeCom, if $\overline{f}_j(x)$ is a polynomial of degree at most t, then $\overline{f}_j(x)$ would be considered as the output of DeCom$_j$ and by the properties of Com, the same polynomial was committed by P_j during the instance Com$_j$. However, we now perform *additional* check-

Fig. 5 AVSS with $n = 3t + 1$.

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

$\mathsf{Sh}(\mathsf{D}, \mathcal{P}, s, \epsilon)$

All instances of Com in the following protocol are executed with error parameter $\epsilon' = \frac{\epsilon}{n}$ to bound the error probability of Sh by ϵ.

DISTRIBUTION OF SHARES — The following code is executed only by D:

1. Select a random, symmetric bivariate polynomial $F(x, y)$ of degree t, such that $F(0, 0) = s$. For $i = 1, \ldots, n$, compute $f_i(x) \stackrel{def}{=} F(x, i)$.
2. For $i = 1, \ldots, n$, send the *share* $f_i(x)$ to the party P_i.

COMMITMENT OF THE SHARES — For $i = 1, \ldots, n$, every party $P_i \in \mathcal{P}$ (including D) executes the following code:

1. Wait to receive $f_i(x)$ from D.
2. If $f_i(x)$ has degree at most t, then act as a Committer and execute $\mathsf{Com}(P_i, \mathcal{P}, f_i(x), \epsilon')$[a] to commit the share $f_i(x)$. We call this instance of Com, initiated by P_i as Com_i and let WCORE_i be the instance of WCORE constructed in the instance Com_i.
3. For $j = 1, \ldots, n$, participate in the instance Com_j of Com, initiated by P_j. During the instance Com_j, if the secondary signature $\mathsf{ICSig}(P_j, P_i, \mathcal{P}, f_j(i))$ is received from P_j, then check whether $f_j(i) \stackrel{?}{=} f_i(j)$. If $f_j(i) = f_i(j)$, then only perform the rest of the steps of the protocol Com that P_i is supposed to perform in the instance Com_j. Otherwise, do not participate further in the instance Com_j.
4. For $j = 1, \ldots, n$, do not terminate and keep participating in the instance Com_j, even after receiving the set WCORE_j of size $2t + 1$ from the broadcast of Committer P_j.

ShVCORE CONSTRUCTION — The following code is executed only by D:

1. If a WCORE_j of size $2t+1$ is received from the broadcast of Committer P_j and the messages $(\texttt{Sign-Sent}, k, P_j)$ are received from the broadcast of every $P_k \in \mathsf{WCORE}_j$ in the instance Com_j, then include P_j in a dynamic set $\mathcal{T}$ (which is initially $\emptyset$). Do not terminate Com_j and keep participating in the instance Com_j.
2. If a new message $(\texttt{Sign-Sent}, k, P_j)$ is received from the broadcast of P_k during the instance Com_j, where $P_j \in \mathcal{T}$ and $P_k \notin \mathsf{WCORE}_j$, then update WCORE_j and include P_k in WCORE_j[b].
3. Keep updating $\mathcal{T}$ and the existing WCORE_js, corresponding to the P_js in $\mathcal{T}$, by performing the previous two steps, until there exists a set $\mathsf{ShVCORE} \subseteq \mathcal{T}$, such that $|\mathsf{ShVCORE}| \geq 2t + 1$ and $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t + 1$ holds for every $P_j \in \mathsf{ShVCORE}$.
4. On finding ShVCORE, broadcast the set ShVCORE and the set WCORE_j corresponding to each $P_j \in \mathsf{ShVCORE}$ and terminate.

VERIFICATION OF ShVCORE AND TERMINATION — For $i = 1, \ldots, n$, every party $P_i \in \mathcal{P}$ (including D) executes the following code:

1. Wait to receive ShVCORE and WCORE_j corresponding to each $P_j \in \mathsf{ShVCORE}$ from the broadcast of D. On receiving check if $|\mathsf{ShVCORE}| \geq 2t + 1$ and $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t + 1$ holds for every $P_j \in \mathsf{ShVCORE}$.
2. If the checking in the previous step passes, then corresponding to each $P_j \in \mathsf{ShVCORE}$, wait to receive the message $(\texttt{Sign-Sent}, k, P_j)$ from the broadcast of every party $P_k \in \mathsf{WCORE}_j$, during the instance Com_j. On receiving all these messages, terminate the protocol.

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

All instances of DeCom in the following protocol are executed with error parameter $\epsilon' = \frac{\epsilon}{n}$ to bound the error probability of Rec by ϵ.

DECOMMITTING THE SHARE — The following code is executed by every party $P_j \in \mathsf{ShVCORE}$:

1. Act as a Committer and decommit the share $f_j(x)$ committed during Com_j by executing $\mathsf{DeCom}(P_j, \mathcal{P}, f_j(x), \epsilon')$. Denote this instance of DeCom as DeCom_j.
2. If $P_j \in \mathsf{WCORE}_k$, corresponding to some $P_k \in \mathsf{ShVCORE}$, then reveal the secondary signature $\mathsf{ICSig}(P_k, P_j, \mathcal{P}, f_k(j))$, received during the instance Com_k from Committer P_k, by executing an instance of RevealPublic.

VERIFYING THE DECOMMITMENT, SECRET RECONSTRUCTION AND TERMINATION — For $i = 1, \ldots, n$, every party $P_i \in \mathcal{P}$ executes the following code:

1. Corresponding to every $P_j \in \mathsf{ShVCORE}$, participate in the instance DeCom_j, executed by P_j.
2. Corresponding to every $P_j \in \mathsf{ShVCORE}$, participate in the instances of RevealPublic, executed by P_j to reveal the secondary signatures $\mathsf{ICSig}(P_k, P_j, \mathcal{P}, f_k(j))$, corresponding to every $P_k \in \mathsf{ShVCORE}$ where $P_j \in \mathsf{WCORE}_j$.
3. Construct a set RecVCORE, which is initially $\emptyset$. Include $P_j \in \mathsf{ShVCORE}$ in RecVCORE if all the following holds:
   (a) A polynomial of degree at most t, say $\overline{f}_j(x)$ is obtained as the output at the end of DeCom_j.
   (b) Corresponding to every $P_k \in \mathsf{ShVCORE}$ where $P_j \in \mathsf{WCORE}_k$, party P_i completed the revelation of the secondary signature $\mathsf{ICSig}(P_k, P_j, \mathcal{P}, f_k(j))$ with output $\mathsf{Reveal}_i = f_k(j)$ and $\overline{f}_j(k) = f_k(j)$ holds.
4. Wait till $|\mathsf{RecVCORE}| = |\mathsf{ShVCORE}| - t$.
5. Corresponding to every $P_k \in \mathsf{ShVCORE}$, compute his share $\overline{f}_k(x)$ as follows:
   (a) If $P_k \in \mathsf{RecVCORE}$, then $\overline{f}_k(x)$ is the same as obtained at the end of DeCom_k.
   (b) If $P_k \notin \mathsf{RecVCORE}$, then $\overline{f}_k(x)$ is obtained by interpolating the points $\{(j, f_k(j))\}$, where $P_j \in \mathsf{RecVCORE}, P_j \in \mathsf{WCORE}_k$ and P_j revealed the secondary signature $\mathsf{ICSig}(P_k, P_j, \mathcal{P}, f_k(j))$.
6. If the shares $\{\overline{f}_k(x) : P_k \in \mathsf{ShVCORE}\}$ lie on a unique, symmetric bi-variate polynomial of degree t, say $\overline{F}(x, y)$, then output $\overline{s} = \overline{F}(0, 0)$ and terminate; otherwise output $\perp$ and terminate.

[a] See the last section for the interpretation of committing and decommitting a polynomial through a polynomial of degree t.

[b] This step denotes the "expansion" of WCORE_j beyond its initial size of $2t + 1$.

ing for P_j before taking $\overline{f}_j(x)$ as the output of DeCom_j. More specifically, along with the above steps, we also ask P_j to reveal the *secondary signatures* $\mathsf{ICSig}(P_k, P_j, \mathcal{P}, f_k(j))$, received from P_k during the instance Com_k, corresponding to *all* P_k's in $\mathsf{ShVCORE}$, where $P_j \in \mathsf{WCORE}_k$; moreover $f_k(j)$ should be the same as $\overline{f}_j(k)$, otherwise P_j is *discarded*. The idea is that if $P_j \in \mathsf{ShVCORE}$ and honest and if $P_j \in \mathsf{WCORE}_k$, corresponding to another $P_k \in \mathsf{ShVCORE}$, then P_j must have participated in the instance Com_k, otherwise P_j will be never included in WCORE_k. This further implies that during Com_k, P_j must have received the secondary signature $\mathsf{ICSig}(P_k, P_j, \mathcal{P}, f_k(j))$ from the $\mathsf{Committer}$ P_k and would have verified that $f_k(j) = \overline{f}_j(k)$ and only then would have continued with the rest of the steps in Com_k. So an honest $P_j \in \mathsf{ShVCORE}$ will be able to pass these additional checking during the reconstruction protocol and will not be discarded. The interesting feature is that once we have $|\mathsf{ShVCORE}| - t$ non-discarded parties, denoted by $\mathsf{RecVCORE}$, then apart from the shares committed by the *honest* parties in $\mathsf{RecVCORE}$, we will also know the shares committed by the *honest* parties in $\mathsf{ShVCORE} \setminus \mathsf{RecVCORE}$, through the secondary signatures revealed by the parties in $\mathsf{RecVCORE}$. This is because for each *honest* $P_k \in \mathsf{ShVCORE} \setminus \mathsf{RecVCORE}$, there will be at least $t+1$ parties, who were present in WCORE_k and who will be also present in $\mathsf{RecVCORE}$. This is ensured because during the sharing protocol there were at least $2t+1$ common parties in $\mathsf{ShVCORE}$ and WCORE_k; so clearly there will be at least $t+1$ common parties in $\mathsf{RecVCORE}$ and WCORE_k, as $\mathsf{RecVCORE} \subset \mathsf{ShVCORE}$, where $|\mathsf{RecVCORE}| = |\mathsf{ShVCORE}| - t$. Now these $t+1$ parties in WCORE_k (who are also in $\mathsf{RecVCORE}$) will correctly reveal the secondary signatures of P_k on the $t+1$ AWC-shares of $f_k(x)$, which are enough to reconstruct the share $f_k(x)$, which has degree at most t. This implies that even without waiting to terminate the DeCom instances of all the parties in $\mathsf{ShVCORE}$, we can reconstruct the shares of all the honest parties in $\mathsf{ShVCORE}$. We then check whether all the reconstructed shares lie on a unique symmetric bivariate polynomial of degree t and accordingly output either the constant term of the polynomial as the reconstructed value or $\perp$ as the reconstructed value. Since the shares of all the honest parties in $\mathsf{ShVCORE}$ are reconstructed correctly, no value other than the committed secret $\overline{s}$ will be reconstructed. Protocol $\mathsf{AVSS-Single}$ is formally presented in Fig 5.

We now prove the properties of the protocol $\mathsf{AVSS-Single}$.

Lemma 8 (Termination) *Protocols* $(\mathsf{Sh}, \mathsf{Rec})$ *satisfy the termination condition of definition 2.*

PROOF: If D is *honest*, then $f_i(j) = f_j(i)$ will hold for every pair (P_i, P_j) of honest parties, which implies that every honest party will eventually participate in the Com instance of every other honest party. This implies that D will eventually include every honest party P_i in the instance WCORE_j corresponding to every honest P_j. This is because D does not immediately terminate the instance Com_j after receiving a WCORE_j of size $2t+1$ from P_j in the instance Com_j. Now every honest party will be eventually included in the set $\mathcal{T}$ and so D will eventually find an $\mathsf{ShVCORE} \subseteq \mathcal{T}$, such that $|\mathsf{ShVCORE}| \geq 2t+1$ and $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t+1$ holds for every $P_j \in \mathsf{ShVCORE}$. This implies that D will eventually broadcast $\mathsf{ShVCORE}$ and WCORE_js corresponding to every $P_j \in \mathsf{ShVCORE}$. By the properties of broadcast, every honest party will eventually receive these sets from the broadcast of D. Moreover, every honest party will also eventually receive the desired $(\texttt{Sign-Sent}, \star, \star)$ messages, as D have received these messages while constructing $\mathsf{ShVCORE}$ and the corresponding WCOREs. So every honest party will eventually terminate the protocol Sh. This proves the first requirement.

Let P_{hon} be the first honest party who has terminated Sh. This implies that P_{hon} must have received the set $\mathsf{ShVCORE}$ and the sets WCORE_js corresponding to every $P_j \in \mathsf{ShVCORE}$ and verified that $|\mathsf{ShVCORE}| \geq 2t+1$ and $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t+1$. By the properties of broadcast, every other honest party will also eventually receive these sets and the verification will also pass for them. Party P_{hon} must have also received all the desired $(\texttt{Sign-Sent}, \star, \star)$ messages from the broadcast of the corresponding parties before terminating Sh. The properties of broadcast implies that every other honest party will also eventually receive the same $(\texttt{Sign-Sent}, \star, \star)$ messages and will eventually terminate the protocol Sh. This proves the second requirement.

If the honest parties terminate Sh, then they know $\mathsf{shVCORE}$ and there are at least $|\mathsf{ShVCORE}| - t$ honest parties in the set $\mathsf{ShVCORE}$. During the protocol Rec, these honest parties (in $\mathsf{ShVCORE}$) will honestly perform all the steps required during DECOMMITTING THE SHARE, namely they will decommit their share, which will be a univariate polynomial of degree at most t, except with probability at most ϵ' (follows from the correctness property of AWC); moreover they will be able to correctly reveal all the required secondary signatures, except with probability at most ϵ' (follows from the **AICP-Correctness2**). So even if the corrupted parties in $\mathsf{ShVCORE}$ do not perform their steps correctly during the Rec protocol, the honest parties will be eventually included in the set $\mathsf{RecVCORE}$, except with probability at most $|\mathsf{ShVCORE}| \cdot \epsilon' \approx \epsilon$. It is now easy to see that once the set $\mathsf{RecVCORE}$ is constructed, every honest party will eventually output either an $\overline{s}$ or $\perp$ and hence will terminate Rec. This proves the third requirement. □

Lemma 9 (Secrecy) *If* D *is honest then the information received by* Adv *till the end of* Sh *is distributed independently of the secret* s.

PROOF: Let $\mathcal{C}$ be the set of parties under the control of Adv, where $|\mathcal{C}| \leq t$ and $\mathsf{D} \notin \mathcal{C}$. So Adv will know the shares $f_i(x)$, where $P_i \in \mathcal{C}$. We first claim that throughout the protocol Sh, the adversary obtains no extra information other than these shares. This is because during the instance Com_i corresponding to an *honest* party P_i, Adv will obtain at most t AWC-shares of the share $f_i(x)$, which are already known to Adv, as these AWC-shares are the same as the t share-shares of $f_i(x)$, which Adv can compute from the shares of the t parties in $\mathcal{C}$. The secrecy property of Com ensures that the information revealed to Adv during Com_i is independent of $f_i(0)$ and hence no new information about the share $f_i(x)$ is revealed to Adv during Com_i. We now sow that given only the shares of the corrupted parties in $\mathcal{C}$, no information about the secret $s = F(0,0)$ is revealed to Adv. The proof follows from the properties of a symmetric bivariate polynomial of degree t, as given in [11]; for the sake of completeness, we recall the proof in the sequel.

To complete the proof, it is sufficient to show that from the view point of the adversary, for every possible secret $\overline{s} \in \mathbb{F}$, there are same number of symmetric bi-variate polynomials $\overline{F}(x,y)$ of degree t, with $\overline{s} = \overline{F}(0,0)$, such that $\overline{F}(x,y)$ is consistent with the shares received by Adv during Sh; i.e. $f_i(x) = \overline{F}(x,i)$ holds for every $P_i \in \mathcal{C}$. We proceed to do the same in the following.

Let $\overline{f}_i(x) \stackrel{def}{=} \overline{F}(x,i)$. Consider the polynomial

$$h(x) = \prod_{P_i \in \mathcal{C}} \left(\frac{-1}{i} \cdot x + 1 \right).$$

The polynomial $h(x)$ has degree at most t, where $h(0) = 1$ and $h(i) = 0$, for every $P_i \in \mathcal{C}$. Now define the bi-variate polynomial

$$Z(x,y) \stackrel{def}{=} h(x) \cdot h(y).$$

Note that $Z(x,y)$ is a symmetric bi-variate polynomial of degree t and $Z(0,0) = 1$ and $z_i(x) \stackrel{def}{=} Z(x,i) = 0$, for every $P_i \in \mathcal{C}$. Now if during the protocol Sh, D in reality has used the polynomial $F(x,y)$, then for every possible $\overline{s}$, the information (namely the shares) held by Adv is also consistent with the polynomial

$$\overline{F}(x,y) = F(x,y) + (\overline{s} - s) \cdot Z(x,y).$$

Indeed $\overline{F}(x,y)$ is a symmetric bi-variate polynomial of degree t and for every $P_i \in \mathcal{C}$,

$$\overline{f}_i(x) = \overline{F}(x,i) = f_i(x) + (\overline{s} - s) \cdot z_i(x) = f_i(x),$$

and

$$\overline{F}(0,0) = F(0,0) + (\overline{s} - s) \cdot Z(0,0) = s + \overline{s} - s = \overline{s}.$$

Thus there exists a one-to-one correspondence between the consistent polynomials for the shared secret s and those for $\overline{s}$ and so all secrets are equally likely from the view point of the adversary. □

Lemma 10 (Correctness) *Protocols* (Sh, Rec) *satisfy the correctness condition of definition 2.*

PROOF: Let P_{hon} be the first honest party to terminate the protocol Sh; this implies that P_{hon} has received the set shVC ORE and the corresponding WCORE_js from D and verified that $|\mathsf{ShVCORE}| \geq 2t+1$ and $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t+1$ for every $P_j \in \mathsf{ShVCORE}$. Notice that each party P_i in $\mathcal{H}$ has committed the same share $f_i(x)$, as received from D. We define the committed secret $\overline{s}$, committed by D as follows: if there exists a unique symmetric bi-variate polynomial of degree t, say $\overline{F}(x,y)$, such that $f_i(x) = \overline{F}(x,i)$ holds for every $P_i \in \mathcal{H}$ (recall that this means that the shares of the parties in $\mathcal{H}$ lie on $\overline{F}(x,y)$), then $\overline{s} = \overline{F}(0,0)$; otherwise $\overline{s} = \perp$. It is easy to see that if D is *honest*, then $\overline{s} = s$, as $\overline{F}(x,y) = F(x,y)$ in this case. We next show that each honest party upon terminating Rec will output only $\overline{s}$; we consider the following two cases, depending upon whether D is honest or corrupted:

1. D *is honest*: we first observe that if there exists a *corrupted* $P_j \in \mathsf{ShVCORE}$, then the share $\overline{f}_j(x)$ committed by P_j during Com_j is the same as $f_j(x)$, where $f_j(x) = F(x,y)$ and $F(x,y)$ is the polynomial selected by D. This is because each *honest* party P_i in WCORE_j must have checked that the AWC-share $\overline{f}_j(i)$ received from P_j during Com_j is the same as the ith share-share $f_i(j)$ of $f_j(x)$ received from D before participating in Com_j; i.e. P_i have checked that $\overline{f}_j(i) = f_i(j)$ before participating (exchanging the primary and secondary signatures) in Com_j. Moreover, there are at least $t+1$ honest P_is in WCORE_j, whose share-shares of $f_j(x)$ uniquely define the share $f_j(x)$ and so $\overline{f}_j(x) = f_j(x)$.
   We next claim that during the protocol Rec, the share $\overline{f_k}(x)$ computed on behalf of each $P_k \in \mathsf{ShVCORE}$ is the same as $f_k(x) = F(x,k)$, except with probability at most ϵ', which implies that $s = F(0,0)$ will be output, except with probability $|\mathsf{ShVCORE}| \cdot \epsilon \approx \epsilon'$. There are two cases:
   (a) $P_k \in \mathsf{RecVCORE}$: In this case, $\overline{f}_k(x)$ is the output of the instance DeCom_k. If P_k is *honest* then clearly $\overline{f}_k(x) = f_k(x)$, as the (correctness) property of DeCom ensures that if Committer is honest, then the polynomial (of degree at most t) committed by him during the Com protocol, will be obtained as the output during DeCom. On the other hand, if P_k is *corrupted*, then also $\overline{f}_k = f_k(x)$, except with probability ϵ'; this is because the (correctness) property of DeCom ensures that if Committer is corrupted and the output of DeCom is a polynomial of degree at most t, then except with probability ϵ', the same polynomial was committed during the Com protocol. Moreover, as discussed above the polynomial committed by a corrupted $P_k \in \mathsf{ShVCORE}$ during Com_k is the same as $f_k(x)$.

(b) $P_k \notin$ RecVCORE: In this case, $\overline{f}_k(x)$ is computed by interpolating the points $\{(j, f_k(j))\}$, where $P_j \in$ RecVCORE, $P_j \in \mathsf{WCORE}_k$ and P_j has (correctly) revealed the secondary signature $\mathsf{ICSig}(P_k, P_j, \mathcal{P}, f_k(j))$ on the AWC-share $f_k(j)$, which was given to P_j by P_k during the instance Com_k. Moreover the revealed $f_k(j)$ lies on the share $\overline{f}_j(x)$, where $\overline{f}_j(x)$ is computed as the share on the behalf of P_j during Rec; i.e. $f_k(j) = \overline{f}_j(k)$. We first notice that there will be at least $t+1$ such interpolating points $\{(j, f_k(j))\}$. This follows from the fact that $P_k \in$ ShVCORE implies that during Sh, $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_k| \geq 2t+1$ and at least $t+1$ of these common parties (which were present in ShVCORE as well as in WCORE_k) will be present in RecVCORE. Now we have already shown in the previous case that the share $\overline{f}_j(x)$ computed on the behalf of each $P_j \in$ RecVCORE is the same as the original share $f_j(x)$ except with probability ϵ'; i.e. $\overline{f}_j(x) = f_j(x) = F(x, j)$. Now the fact that for each interpolating point $(j, f_k(j))$ used to interpolate $\overline{f}_k(x)$ the relation $f_k(j) = \overline{f}_j(k)$ holds implies that $f_k(j) = f_j(k) = F(j, k)$ holds. So except with probability ϵ', $\overline{f}_k(x) = f_k(x)$.

2. D *is corrupted*: If $\overline{s} = \overline{F}(0,0)$, then the proof is exactly the same as for the case when D is honest. Now let us consider the case when $\overline{s} = \perp$, which implies that the shares of the parties in $\mathcal{H}$ do lie on a unique symmetric bi-variate polynomial of degree t. We show that except with probability at most ϵ', the share of each party P_k in $\mathcal{H}$ will be computed correctly during the protocol Rec and so except with probability at most $|\mathcal{H}| \cdot \epsilon' \approx \epsilon$, every honest party will output $\perp$, irrespective of the shares which are computed on behalf of the corrupted parties in ShVCORE.

   If $P_k \in$ RecVCORE, then the above claim is true, as in this case, the share $\overline{f}_k(x)$ computed on the behalf of P_k is obtained as the output of Com_k and the correctness property of AWC ensures that for an honest Committer, the committed polynomial will be obtained correctly during the DeCom. If $P_k \notin$ RecVCORE, then also the claim is true, as in this case the share $\overline{f}_k(x)$ computed on the behalf of P_k is obtained by interpolating the points $\{(j, f_k(j))\}$, where $P_j \in$ RecVCORE, $P_j \in \mathsf{WCORE}_k$ and P_j has (correctly) revealed the secondary signature $\mathsf{ICSig}(P_k, P_j, \mathcal{P}, f_k(j))$ on the AWC-share $f_k(j)$, which was given to P_j (as an INT) by P_k (as a Signer) during the instance Com_k. The **AICP-Correctness3** property ensures that all the revealed points $\{(j, f_k(j))\}$ indeed lie on the original polynomial $f_k(x)$, which was committed by P_k during Com_k and so $\overline{f}_k(x) = f_k(x)$. □

Theorem 5 *Protocols* $(\mathsf{Sh}, \mathsf{Rec})$ *is a* $(1-\epsilon)$*-AVSS scheme. Protocol* Sh *requires a private communication of* $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ *bits and broadcast of* $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ *bits. Protocol* Rec *requires a broadcast of* $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ *bits.*

PROOF: During the protocol Sh, D distributes n univariate polynomials of degree at most t as shares and n instances of Com are executed. During the protocol Rec, at most n instances of DeCom and at most n^2 instances of RevealPublic are executed. The proof now follows from Theorem 2 by substituting $\ell = 1$, Theorem 3 and from Lemma 8-10. □

5.2 AVSS for Sharing ℓ Secrets

Consider a scenario where D would like to share a vector of secrets $\vec{S} = (s^1, \ldots, s^\ell)$, consisting of ℓ elements from $\mathbb{F}$, where $\ell > 1$ and later the parties would like to reconstruct these secrets. One way of doing this is to execute an instance of the Sh and Rec protocol on the behalf of each element $s^l \in \vec{S}$; this will require a private as well as broadcast communication of $\mathcal{O}(\ell n^3 \log \frac{1}{\epsilon})$ bits. However, we now show how to share and reconstruct *all* the ℓ elements of $\vec{S}$ concurrently, so that it requires a private as well as broadcast communication of $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits. So if $\ell = \Omega(n)$, which will be the case in our common coin protocol, then the broadcast communication of our protocol will be *independent* of ℓ.

The underlying idea is to "extend" the AVSS scheme for sharing a single secret to deal with ℓ secrets *concurrently* in a "natural" way, similar to what was done earlier for the AWC. More specifically, D selects a random symmetric bi-variate polynomial $F_l(x, y)$ for sharing each $s^l \in \vec{S}$ and computes ℓ ith share $f_{1,i}(x), \ldots, f_{\ell,i}(x)$, where $f_{l,i}(x) \stackrel{def}{=} F_l(x, i)$ and distributes these shares. But now, instead of executing ℓ *different* instances of Com to commit ℓ ith share, party P_i executes a *single* instance of Com to commit all the ℓ polynomials (shares) concurrently. The rest of the protocol steps of $\mathsf{AVSS-Single}$ are modified in the same way, so as to deal with ℓ shares concurrently. The modified protocols are presented in Fig 6. The new scheme is called $\mathsf{AVSS-Multiple}$, as it deals with multiple secrets. The properties of the modified scheme follow using the same arguments as for the earlier scheme and so we avoid giving the complete proofs to avoid repetition.

We state the following theorem, stating the communication complexity of the protocol $\mathsf{AVSS-Multiple}$, whose proof follows from the properties of the protocol and the communication complexity of our AICP (Theorem 2) and AWC (Theorem 4).

Theorem 6 *In* $\mathsf{AVSS-Multiple}$*, protocol* Sh *requires a private communication of* $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ *bits and broadcast of* $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ *bits. Protocol* Rec *requires a broadcast of* $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ *bits.*

Fig. 6 AVSS with $n = 3t + 1$.

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

$\mathsf{Sh}(\mathsf{D}, \mathcal{P}, \vec{S} = (s^1, \ldots, s^\ell), \epsilon)$

All instances of Com in the following protocol are executed with error parameter $\epsilon' = \frac{\epsilon}{n}$ to bound the error probability of Sh by ϵ.

DISTRIBUTION OF SHARES — The following code is executed only by D:

1. For $l = 1, \ldots, \ell$, corresponding to the secret $s^l \in \vec{S}$, select a random, symmetric bivariate polynomial $F_l(x, y)$ of degree t, such that $F_l(0, 0) = s$. For $i = 1, \ldots, n$, compute $f_{l,i}(x) \stackrel{def}{=} F_l(x, i)$.
2. For $i = 1, \ldots, n$, send the *share* $f_{1,i}(x), \ldots, f_{\ell,i}(x)$ to the party P_i.

COMMITMENT OF THE SHARES — For $i = 1, \ldots, n$, every party $P_i \in \mathcal{P}$ (including D) executes the following code:

1. Wait to receive $f_{1,i}(x), \ldots, f_{\ell,i}(x)$ from D.
2. If $f_{1,i}(x), \ldots, f_{\ell,i}(x)$ have degree at most t, then act as a Committer and execute an instance of Com to commit the shares $f_{1,i}(x), \ldots, f_{\ell,i}(x)$. We call this instance of Com, initiated by P_i as Com_i and let WCORE_i be the instance of WCORE constructed in the instance Com_i.
3. For $j = 1, \ldots, n$, participate in the instance Com_j of Com, initiated by P_j. During the instance Com_j, if the secondary signature $\mathsf{ICSig}(P_j, P_i, \mathcal{P}, (f_{1,j}(i), \ldots, f_{\ell,j}(i)))$ is received from P_j, then check whether $f_{l,j}(i) \stackrel{?}{=} f_{l,i}(j)$, for $l = 1, \ldots, \ell$. If $f_{l,j}(i) = f_{l,i}(j)$ for all $l = 1, \ldots, \ell$, then only perform the rest of the steps of the protocol Com that P_i is supposed to perform in the instance Com_j. Otherwise, do not participate further in the instance Com_j.
4. For $j = 1, \ldots, n$, do not terminate and keep participating in the instance Com_j, even after receiving the set WCORE_j of size $2t + 1$ from the broadcast of Committer P_j.

ShVCORE CONSTRUCTION — The following code is executed only by D:

1. If a WCORE_j of size $2t + 1$ is received from the broadcast of Committer P_j and the messages $(\texttt{Sign-Sent}, k, P_j)$ are received from the broadcast of every $P_k \in \mathsf{WCORE}_j$ in the instance Com_j, then include P_j in a set $\mathcal{T}$ (which is initially $\emptyset$). Do not terminate Com_j and keep participating in the instance Com_j.
2. If a new message $(\texttt{Sign-Sent}, k, P_j)$ is received from the broadcast of P_k during the instance Com_j, where $P_j \in \mathcal{T}$ and $P_k \notin \mathsf{WCORE}_j$, then update WCORE_j and include P_k in WCORE_j.
3. Keep updating $\mathcal{T}$ and the existing WCORE_js, corresponding to the P_js in $\mathcal{T}$, by performing the previous two steps, until there exists a set $\mathsf{ShVCORE} \subseteq \mathcal{T}$, such that $|\mathsf{ShVCORE}| \geq 2t + 1$ and $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t + 1$ holds for every $P_j \in \mathsf{ShVCORE}$.
4. On finding ShVCORE, broadcast the set ShVCORE and the set WCORE_j corresponding to each $P_j \in \mathsf{ShVCORE}$ and terminate.

VERIFICATION OF ShVCORE AND TERMINATION — For $i = 1, \ldots, n$, every party $P_i \in \mathcal{P}$ (including D) executes the following code:

1. Wait to receive ShVCORE and WCORE_j corresponding to each $P_j \in \mathsf{ShVCORE}$ from the broadcast of D. On receiving check if $|\mathsf{ShVCORE}| \geq 2t + 1$ and $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t + 1$ holds for every $P_j \in \mathsf{ShVCORE}$.
2. If the checking in the previous step passes, then corresponding to each $P_j \in \mathsf{ShVCORE}$, wait to receive the message $(\texttt{Sign-Sent}, k, P_j)$ from the broadcast of every party $P_k \in \mathsf{WCORE}_j$, during the instance Com_j. On receiving all these messages, terminate the protocol.

$\mathsf{Rec}(\mathsf{D}, \mathcal{P}, \vec{S} = (s^1, \ldots, s^\ell), \epsilon)$

All instances of DeCom in the following protocol are executed with error parameter $\epsilon' = \frac{\epsilon}{n}$ to bound the error probability of Rec by ϵ.

DECOMMITTING THE SHARE — The following code is executed by every party $P_j \in \mathsf{ShVCORE}$:

1. Act as a Committer and decommit the shares $f_{1,j}(x), \ldots, f_{\ell,j}(x)$ committed during Com_j by executing an instance of DeCom. Denote this instance of DeCom as DeCom_j.
2. If $P_j \in \mathsf{WCORE}_k$, corresponding to some $P_k \in \mathsf{ShVCORE}$, then reveal the secondary signature $\mathsf{ICSig}(P_k, P_j, \mathcal{P}, (f_{1,k}(j), \ldots, f_{\ell,k}(j)))$, received during the instance Com_k from Committer P_k.

VERIFYING THE DECOMMITMENT, SECRET RECONSTRUCTION AND TERMINATION — For $i = 1, \ldots, n$, every party $P_i \in \mathcal{P}$ executes the following code:

1. Corresponding to every $P_j \in \mathsf{ShVCORE}$, participate in the instance DeCom_j, executed by P_j.
2. Corresponding to every $P_j \in \mathsf{ShVCORE}$, participate in the instances of RevealPublic, executed by P_j to reveal the secondary signatures $\mathsf{ICSig}(P_k, P_j, \mathcal{P}, (f_{1,k}(j), \ldots, f_{\ell,k}(j)))$, corresponding to every $P_k \in \mathsf{ShVCORE}$ where $P_j \in \mathsf{WCORE}_j$.
3. Construct a set RecVCORE, which is initially $\emptyset$. Include $P_j \in \mathsf{ShVCORE}$ in RecVCORE if all the following holds:
   (a) ℓ polynomials of degree at most t, say $\overline{f}_{1,j}(x), \ldots, \overline{f}_{\ell,j}(x)$ are obtained as the output at the end of DeCom_j.
   (b) Corresponding to every $P_k \in \mathsf{ShVCORE}$ where $P_j \in \mathsf{WCORE}_k$, party P_i completed the revelation of the secondary signature $\mathsf{ICSig}(P_k, P_j, \mathcal{P}, (f_{1,k}(j), \ldots, f_{\ell,k}(j)))$ with output $\mathsf{Reveal}_i = (f_{1,k}(j), \ldots, f_{\ell,k}(j))$. Moreover, for $l = 1, \ldots, \ell$, $\overline{f}_{l,j}(k) = f_{l,k}(j)$ holds.
4. Wait till $|\mathsf{RecVCORE}| = |\mathsf{ShVCORE}| - t$.
5. Corresponding to every $P_k \in \mathsf{ShVCORE}$, compute his shares $\overline{f}_{1,k}(x), \ldots, \overline{f}_{\ell,k}(x)$ as follows:
   (a) If $P_k \in \mathsf{RecVCORE}$, then $\overline{f}_{1,k}(x), \ldots, \overline{f}_{\ell,k}(x)$ are the same as obtained at the end of DeCom_k.
   (b) If $P_k \notin \mathsf{RecVCORE}$, then for $l = 1, \ldots, \ell$, the polynomial $\overline{f}_{l,k}(x)$ is obtained by interpolating the points $\{(j, f_{l,k}(j))\}$, where $P_j \in \mathsf{RecVCORE}$, $P_j \in \mathsf{WCORE}_k$ and P_j revealed the secondary signature $\mathsf{ICSig}(P_k, P_j, \mathcal{P}, (f_{1,k}(j), f_{\ell,k}(j)))$.
6. If for $l = 1, \ldots, \ell$, the shares $\{\overline{f}_{l,k}(x) : P_k \in \mathsf{ShVCORE}\}$ lie on a unique, symmetric bi-variate polynomial of degree t, say $\overline{F}_l(x, y)$, then output $\overline{S} = (\overline{F}_1(0, 0), \ldots, \overline{F}_\ell(0, 0))$ and terminate; otherwise output $\perp$ and terminate.

6 Existing Single Bit Common Coin and Our Multi-Bit Common Coin

In this section, we recall the description of the existing common coin protocol from [8] and state the properties that the protocol achieve. We then show that if we directly substitute our AVSS scheme $\mathsf{AVSS-Multiple}$ in this common coin protocol to achieve efficiency, then the resultant protocol fails to satisfy the properties of a common coin protocol. We then show the modifications that are required in the existing common coin protocol so that it can use our AVSS scheme $\mathsf{AVSS-Multiple}$ as a black-box. Interestingly, our analysis shows that infact by doing the modifications to the common coin protocol and by incorporating our AVSS scheme $\mathsf{AVSS-Multiple}$, we get a new common coin protocol which allows to generate $\Theta(n)$ common coins concurrently, instead of a single common coin (as was the case in the existing common coin protocol); moreover the $\Theta(n)$ coins are generated with significantly less communication complexity (a comparison will be provided at the end of the section).

6.1 Existing Common Coin Protocol

We recall the definition of the common coin and the construction of the common coin protocol following the description of [8]. In the following description, we assume that $(\mathsf{Sh}, \mathsf{Rec})$ is a given $(1-\epsilon)$-AVSS scheme.

Definition 6 (Common Coin [8]) Let π be an asynchronous protocol, where each party in $\mathcal{P}$ has a local random input and a binary output. We say that π is a $(1-\epsilon)-\mathsf{completing}, t-\mathsf{resilient}, p-\mathsf{common\ coin}$ protocol, if the following requirements hold for every possible input of the honest parties and every possible behaviour of Adv:

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

The Underlying Idea: The existing common coin protocol, referred as CC, consists of two stages. In the first stage, each party acts as a dealer and shares n random secrets, using n distinct instances of Sh, each with an allowed error parameter of $\epsilon' = \frac{\epsilon}{n^2}$. The ith secret shared by each party is actually "associated" with the party P_i. Once a party P_i terminates any $t+1$ instances of Sh, corresponding to the $t+1$ secrets associated with him, he broadcasts the identity of the dealers, who have shared those $t+1$ secrets. We say that these $t+1$ secrets are $\mathsf{attached}$ to P_i and later these $t+1$ secrets will be reconstructed to compute a "value", that will be associated with P_i.

Now during the second stage, after terminating the Sh instances of all the secrets attached to a party 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 the 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. 7. The protocol CC is a $(1-\epsilon)-\mathsf{completing}, t-\mathsf{resilient}, \frac{1}{4}-\mathsf{common\ coin}$ protocol, for a given ϵ, where $0 < \epsilon \leq 0.2$.

The proof that the protocol CC satisfies the properties of a common coin protocol is given in [8] and we do not recall it here. However, we do recall one of the lemmas used in [8] to prove the properties of the protocol CC. This is because in the next section, we will show that if we substitute the n instances of Sh executed by a party in the protocol CC by a single instance of the Sh protocol of our AVSS scheme $\mathsf{AVSS-Multiple}$ (to share all the n secrets concurrently), then this lemma may not be true any more; that is, the adversary in the modified common coin protocol may behave in such a way that the properties stated in the following lemma may not be true. For the sake of completeness, we also present the proof of the lemma.

Lemma 11 ([8]) *In the protocol* CC*, once an honest party* P_j *receives the message* $(\texttt{Attach}\ T_i\ \texttt{to}\ P_i)$ *from the broadcast of* P_i *and adds* P_i *in the set* $\mathcal{G}_j$*, then a unique value, say* V_i*, is fixed such that the following holds:*

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

PROOF: Once an *honest* P_j receives the message $(\texttt{Attach}\ T_i\ \texttt{to}\ P_i)$ and adds P_i in $\mathcal{G}_j$, a unique value V_i is fixed, where $V_i = (\sum_{P_k \in T_i} x_{ki}) \mod u$ and x_{ki} is the secret shared by P_k (as a D) during the instance Sh_{ki}. According to the protocol steps, eventually all the honest parties will participate in the instances Rec_{ki} and will reconstruct the secrets x_{ki}s at the end of 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 participates in the instances Rec_{ki}s only after it receives the message $(\texttt{Attach}\ T_i\ \texttt{to}\ P_i)$ from the broadcast of P_i. So the set T_i is fixed before any honest party starts participating in the instances Rec_{ki}s. The secrecy property of AVSS ensures that the corrupted parties will have no information about the value shared by any honest party until the value is reconstructed after executing the corresponding reconstruction protocol. Thus when the set T_i is fixed, the values that are shared by the corrupted parties

Fig. 7 Existing common coin protocol.

$\mathsf{CC}(\epsilon)$

All instances of Sh and Rec in the following protocol have error parameter $\epsilon' = \frac{\epsilon}{n^2}$, so as to bound the error probability of the protocol by ϵ. For $i = 1, \ldots, n$, every party $P_i \in \mathcal{P}$ executes the following code:

1. Initialize a Boolean flag $\mathsf{flag}_i = 0$.
2. For $j = 1, \ldots, n$, corresponding to the party P_j, choose a random value x_{ij} and act as a D to execute an instance of Sh, denoted by Sh_{ij}, to share x_{ij}.
3. For every $j, k \in \{1, \ldots, n\}$, participate in the instances Sh_{jk} and Sh_{kj}.
4. Construct a dynamic set $\mathcal{T}_i$, which is initially $\emptyset$. On terminating all the n instances $\mathsf{Sh}_{j1}, \ldots, \mathsf{Sh}_{jn}$ executed by P_j (as a D), include P_j to $\mathcal{T}_i$.
5. Wait till $|\mathcal{T}_i| = t + 1$. Then assign $T_i = \mathcal{T}_i$ and broadcast the message ($\texttt{Attach}\ T_i\ \texttt{to}\ P_i$).
   (We say that the secrets $\{x_{ji} | P_j \in T_i\}$ are attached to the party P_i).
6. Create a dynamic set $\mathcal{G}_i$, which is initially $\emptyset$. Add party P_j to $\mathcal{G}_i$ if
   (a) The message ($\texttt{Attach}\ T_j\ \texttt{to}\ P_j$) is received from the broadcast of P_j and
   (b) $T_j \subseteq \mathcal{T}_i$.
   Wait until $|\mathcal{G}_i| = 2t + 1$. Then assign $G_i = \mathcal{G}_i$ and broadcast the message ($P_i\ \texttt{Accepts}\ G_i$).
7. Create a dynamic set $\mathcal{S}_i$, which is initially $\emptyset$. Add party P_j to $\mathcal{S}_i$ if
   (a) The message ($P_j\ \texttt{Accepts}\ G_j$) is received from the broadcast of P_j and
   (b) $G_j \subseteq \mathcal{G}_i$.
   Wait until $|\mathcal{S}_i| = 2t + 1$. Then set $\mathsf{flag}_i = 1$ and let H_i be the current contents of $\mathcal{G}_i$.
8. Wait until $\mathsf{flag}_i = 1$. Then for every $P_j \in \mathcal{G}_i$, participate in the instance Rec_{kj}, corresponding to every $P_k \in T_j$, to reconstruct the secret x_{kj}, which was shared in the instance Sh_{kj}.
   (note that some parties may be included in $\mathcal{G}_i$ after flag_i has been set to 1. The corresponding instances of Rec are invoked immediately).
9. Let $u = \lceil 0.87n \rceil$[a]. Every party $P_j \in \mathcal{G}_i$ is associated with a value, say V_j, where V_j is the sum modulo u of all the secrets attached to P_j. That is, $V_j = (\sum_{P_k \in T_j} x_{kj}) \mod u$, where x_{kj} is obtained as the output in the instance Rec_{kj}.
10. Wait until the values associated with all the parties in H_i are computed. If there exits a party $P_j \in H_i$ such that $V_j = 0$, then output 0 and terminate. Otherwise output 1 and terminate.

[a] Here u is selected like this so as to ensure that the probability computations satisfy certain conditions while proving the properties of the protocol.

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 $[0, \ldots, u]$. □

6.2 An InCorrect Common Coin Protocol

In the protocol CC, each party invokes n instances of Sh (each sharing a single secret) to share n secrets. One can easily imagine that those n instances of Sh protocol could be replaced by a more efficient, single instance of the Sh protocol of our AVSS scheme $\mathsf{AVSS-Multiple}$, where all the n secrets can be shared concurrently. This would naturally lead to a more efficient common coin protocol. In the following, we do the same in the protocol WCC. But as the name suggests, we then show that this "direct" replacement without further modification will lead to an incorrect common coin protocol. Protocol WCC is given in Fig 8.

We next show that the protocol WCC may not satisfy the second part of Lemma 11. That is, during the protocol WCC, the adversary can behave in such a way that the unique value V_i, associated with an *honest* P_i, may not be distributed uniformly over $[0, \ldots, u]$. More specifically, Adv can decide the V_i for up to $t - 1$ honest parties and thus such V_i's are no longer random and uniformly distributed over $[0, \ldots, u]$. Consequently, Adv can enforce few honest parties to *always* output 0, while the remaining honest parties may output $\sigma \in \{0, 1\}$ with probability at least $\frac{1}{4}$. This will strictly violate the property of the common coin. We first discuss the underlying reason for the problem before specifying the exact adversarial behaviour.

The Reason for the Problem: Recall that in the protocol CC, every *honest* party P_k shares the secret x_{ki}, corresponding to a party P_i, using an "independent" instance Sh_{ki} of Sh and if the corresponding instance Rec_{ki} is invoked, then *only* the secret x_{ki} is reconstructed. So even after learning the secret x_{ki} that an honest P_k have selected for an *honest* P_i, the adversary Adv will not know the secrets x_{kj}s that the honest P_k have selected on the behalf of another honest P_j (unless the corresponding instance Rec_{kj} is invoked). However, in the protocol WCC, *all* the n secrets that an honest P_k has selected on behalf of the n parties are shared by a single instance of Sh. Now if we want to reconstruct *only* the secret x_{ki} that P_k has selected on behalf of a party P_i, then it is not possible to do so, without learning all the n secrets that P_k have shared. That is, if we invoke Rec_k to just reconstruct

Fig. 8 An incorrect common coin protocol obtained by using the instances of Sh and Rec protocol, capable of sharing and reconstructing n secrets concurrently.

Protocol WCC(ϵ)

All instances of Sh and Rec in the following protocol have error parameter $\epsilon' = \frac{\epsilon}{n}$, so as to bound the error probability of the protocol by ϵ. For $i = 1, \ldots, n$, every party $P_i \in \mathcal{P}$ executes the following code:

1. Initialize a Boolean flag $\mathsf{flag}_i = 0$.
2. For $j = 1, \ldots, n$, corresponding to the party P_j, choose a random value x_{ij} and act as a D and execute the protocol $\mathsf{Sh}(P_i, \mathcal{P}, \vec{S_i}, \epsilon')$, to share the vector of secrets $\vec{S_i} = (x_{i1}, \ldots, x_{in})$. We denote this instance of Sh by Sh_i.
3. For every $j \in \{1, \ldots, n\}$, participate in the instance Sh_j.
4. Construct a dynamic set $\mathcal{T}_i$, which is initially $\emptyset$. On terminating the instance Sh_j executed by P_j (as a D), include P_j to $\mathcal{T}_i$.
5. Wait till $|\mathcal{T}_i| = t + 1$. Then assign $T_i = \mathcal{T}_i$ and broadcast the message ($\texttt{Attach}\ T_i\ \texttt{to}\ P_i$).
(We say that the secrets $\{x_{ji} | P_j \in T_i\}$ are attached to the party P_i).
6. Create a dynamic set $\mathcal{G}_i$, which is initially $\emptyset$. Add party P_j to $\mathcal{G}_i$ if
(a) The message ($\texttt{Attach}\ T_j\ \texttt{to}\ P_j$) is received from the broadcast of P_j and
(b) $T_j \subseteq \mathcal{T}_i$.
Wait until $|\mathcal{G}_i| = 2t + 1$. Then assign $G_i = \mathcal{G}_i$ and broadcast the message ($P_i\ \texttt{Accepts}\ G_i$).
7. Create a dynamic set $\mathcal{S}_i$, which is initially $\emptyset$. Add party P_j to $\mathcal{S}_i$ if
(a) The message ($P_j\ \texttt{Accepts}\ G_j$) is received from the broadcast of P_j and
(b) $G_j \subseteq \mathcal{G}_i$.
Wait until $|\mathcal{S}_i| = 2t + 1$. Then set $\mathsf{flag}_i = 1$ and let H_i be the current contents of $\mathcal{G}_i$.
8. Wait until $\mathsf{flag}_i = 1$. Then for every $P_j \in \mathcal{G}_i$, participate in the instance Rec_k, corresponding to every $P_k \in T_j$, to reconstruct the vector of secrets $\vec{S_k} = (x_{k1}, \ldots, x_{kj})$, which was shared in the instance Sh_k.
(note that some parties may be included in $\mathcal{G}_i$ after flag_i has been set to 1. The corresponding instances of Rec are invoked immediately).
9. Let $u = \lceil 0.87n \rceil$. Every party $P_j \in \mathcal{G}_i$ is associated with a value, say V_j, where V_j is the sum modulo u of all the secrets attached to P_j. That is, $V_j = (\sum_{P_k \in T_j} x_{kj}) \mod u$, where x_{kj} is the jth element of the vector $\vec{S_k} = (x_{k1}, \ldots, x_{kj})$, which is obtained as the output in the instance Rec_k.
10. Wait until the values associated with all the parties in H_i are computed. If there exits a party $P_j \in H_i$ such that $V_j = 0$, then output 0 and terminate. Otherwise output 1 and terminate.

the secret x_{ki}, then in addition to x_{ki}, everyone (including Adv) learns all the n secrets $x_{k1}, \ldots, x_{kn}$ shared by P_k and this is the main cause of the problem. Specifically, the adversary Adv may schedule the delivery of the messages in such a way that he first learns all the secrets that an honest party has shared on the behalf of each honest party and then accordingly decide the secrets that the corrupted parties should share on behalf of the honest parties, so that finally the value associated with an honest party is no more a random value. Thus the adversary can completely "control" the final value associated with an honest P_i.

In a more detail, let P_i be an *honest* party. We describe a specific behavior of Adv in the protocol WCC, which would allow Adv to decide V_i to be 0 and thus make the honest P_i to output 0 (this strategy 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 adversarial strategy is given in Fig 9.

It is easy to see that by following the strategy given in Fig 9, the adversary can ensure that the $t+1$ parties in the set T_i computed by Adv indeed becomes the *first* $t + 1$ parties, whose instances of Sh are completed by P_i. And so those $t+1$ parties will be taken as the T_i by the party P_i. Moreover, after broadcasting the message ($\texttt{Attach}\ T_i\ \texttt{to}\ P_i$), party P_i will be included in the set $\mathcal{G}_i$ and also P_i will be eventually included in the set H_i. Later, $V_i = 0$ will be computed as the value associated with P_i and since $P_i \in H_i$, party P_i will output 0. So by following this strategy, Adv can ensure that the honest P_i outputs 0 in the protocol WCC.

The above problem can be eliminated if we can ensure that no corrupted party can ever share a secret on the behalf of any party *after* learning the secrets which some honest parties have shared. This is what we achieved in our common coin protocol presented in the next section.

6.3 A Multi-Bit Common Coin Protocol

In this section, we show how to amend the protocol WCC, so that it can handle the problem described in the previous section. Our new common coin protocol called MCC thus makes use of the Sh and Rec protocols of our AVSS scheme AVSS − Multiple. Interestingly, we also show that instead of outputting a *single* bit common coin (as in the protocol CC), protocol MCC can output $(n - 2t) = (t + 1) = \Theta(n)$ common coins, without requiring any additional communication from the parties. Thus instead of executing $(t + 1)$ independent instances of CC to generate $(t + 1)$ common coins, we can execute a single instance of our protocol MCC and generate $(t+1)$ common coins; moreover we achieve this at less cost. We first give the definition of *multi-bit common coin*,

Fig. 9 The adversarial strategy with respect to an honest P_i in the protocol WCC.

1. Let P_i be an *honest* party and P_j be a *corrupted* party. All corrupted parties participate in WCC honestly, except that P_j does not select the n secrets on behalf of the n parties and does not invoke the instance Sh_j.
2. Except for the instance Sh_i and the corresponding instance Rec_i, Adv (as a scheduler) delays all the messages sent to P_i and sent by P_i in every other instance Sh_k and the corresponding instances Rec_k. This prevents P_i to participate in any instance Sh_k and the corresponding Rec_k and hence to construct $\mathcal{T}_i$. However, this does not prevent P_i to be part of $\mathcal{T}_k$ for some P_k. The adversary Adv schedules the messages like this until the following happens:
   (a) $n-t-1$ honest parties (except P_i) and $t-1$ corrupted parties (except P_j) perform all the steps of WCC (honestly), construct the respective sets (namely $\mathcal{T}_k$s, $\mathcal{G}_k$s, $\mathcal{S}_k$s and H_ks), set flag_ks to 1 and start participating in the corresponding Rec instances. This way the n secrets selected by each of the $n-t-1$ honest parties (except P_i) and the $t-1$ corrupted parties will be revealed. (it is to be noted that the corrupted parties can reconstruct the secrets in each Rec_k by behaving honestly, even if the participation of the honest P_i in Rec_k is delayed).
   (b) Adv computes a set T_i of size $t+1$ containing the *corrupted* P_j and *any* t *honest* P_k's, whose Rec_k instances have terminated. So the secrets $\vec{S_k} = (x_{k1}, \ldots, x_{kn})$, shared by $P_k \in T_i$ are known to the adversary.
   (c) Adv selects x_{ji}, corresponding to P_j, such that $V_i = (\sum_{P_k \in T_i} x_{ki}) \mod u = 0$. Party P_j is then instructed to act as a D and invoke Sh_j, with x_{ji} being the secret selected on the behalf of P_i.
3. Once the above conditions are satisfied, Adv schedules the messages to and from P_i corresponding to every instance Sh_k, such that the set T_i computed by Adv (during the step 2(b)) indeed becomes the T_i for the party P_i (in the protocol WCC) and P_i broadcasts the message $(\texttt{Attach}\ T_i\ \texttt{to}\ P_i)$ and eventually includes P_i in the set $\mathcal{G}_i$.

which is a straightforward "extension" of definition 6 for ℓ bits.

Definition 7 (Multi-Bit Common Coin) Let π be an asynchronous protocol, where each party in $\mathcal{P}$ has a local random input and an ℓ bit output. We say that π is a $(1-\epsilon)-$ completing, $t-$ resilient, $p-$ multibit common coin protocol, with ℓ bits output, if the following requirements hold for every possible input of the honest parties and every possible behaviour of Adv:

- **Termination**: If all the honest parties participate in π, then with probability at least $(1-\epsilon)$, all the honest parties terminate the protocol.
- **Correctness**: For every possible value $\sigma \in \{(0, \ldots, 0(\ell \text{ times})), (1, \ldots, 1(\ell \text{ times}))\}$, with probability at least p all the honest parties output σ.

We now present our multi-bit common coin protocol MCC, which goes almost in the same line as the protocol WCC, except that we add few additional steps and modify some of the steps, due to which the corrupted parties are forced to share their secrets before learning anybody elses' secrets. In addition, we also modify the way in which a party computes his final output so that instead of outputting a single bit, he now outputs $\ell = t+1$ bits. We first discuss the high level idea of the protocol, specially the new steps that are added in the protocol WCC.

The High Level Idea of MCC : In the protocol MCC, we ensure that instead of a single value, $n-2t = t+1$ values are associated with each party P_i as follows: Recall that in the protocol WCC and CC, the value associated with P_i is decided based on the secrets shared by the $t+1$ parties in T_i on the behalf of P_i. More specifically, as in the worst case, there could be *only one* honest party in T_i, we added all the $t+1$ secrets shared by these parties on the behalf of P_i to get the uniformly random associated value. We observe that instead if we ensure that there are at least $t+1$ honest parties in T_i, then by applying the randomness extraction algorithm EXT, we can get $t+1$ random values associated with P_i. So in the protocol MCC, instead of ensuring that $|T_i| = t+1$, party P_i ensures that $|T_i| = 2t+1$, which implies that now P_i waits to terminate the Sh instances of $2t+1$ parties, instead of $t+1$ parties, to determine the values that will be associated with P_i. Notice that there are at least $2t+1$ honest parties in $\mathcal{P}$ and so an honest P_i will eventually find $2t+1$ parties whose instances of Sh will be terminated by him and thus an honest P_i will eventually terminate. Now out of the $2t+1$ parties in the set T_i, at least $t+1$ will be honest, who will share uniformly random values on the behalf of P_i; however, the identities of the honest parties in T_i will be unknown. So later, on reconstructing the secrets that the $2t+1$ parties in T_i have shared on the behalf of P_i, we apply the EXT algorithm on these $2t+1$ values and get $t+1$ random values, which will be associated with P_i.

Having discussed the idea behind how to get $t+1$ values associated with a party, we next discuss how we ensure that a corrupted party is not able to see the secrets shared by the honest parties, before sharing his secrets (as it was the case in WCC). In the protocol MCC, a party P_i simply does not add a party P_j in the set $\mathcal{T}_i$ after *himself* terminating the instance Sh_j; rather P_i waits to see that whether $n-t$ parties have terminated the instance Sh_j, before adding P_j to $\mathcal{T}_i$. This is achieved by asking every party to broadcast a "terminate" message (along with the identity of the dealer of the instance) once it terminates an instance of Sh and asking P_i to wait for $n-t$ such terminate messages corresponding to P_j. Next the construction of the sets $\mathcal{G}_i, \mathcal{S}_i$ and H_i is done in the same way as in the protocol WCC. Now what

follows is the *most crucial* step of MCC: after constructing the set $\mathcal{S}_i$, party P_i simply does not set his flag_i to 1 and starts participating in the corresponding Rec instances; instead P_i broadcasts a "reconstruct enable" message, indicating that he is "ready" to participate in the Rec instances. In addition, P_i starts participating in the corresponding Rec instances *only after* receiving the reconstruct enable messages from $n-t$ parties. Most importantly, once P_i starts participating in the Rec instances (after receiving $n-t$ reconstruct enable messages), he stops participating in the Sh instances of all the parties, which are not present in his $\mathcal{T}_i$ set at that stage. Thus, if $P_j \notin \mathcal{T}_i$, then P_i stops participating in the instance Sh_j and later resumes (participates) this instance only if P_j is included in $\mathcal{T}_i$.

The above *two* conditions, namely participating in the Rec instances only after receiving $n-t$ reconstruct enable messages, in conjunction with stopping participation in all the Sh_j instances where $P_j \notin \mathcal{T}_i$, ensures that a corrupted party is not able to select and share the secrets on the behalf of the honest parties after learning the secrets which the honest parties have shared on the behalf of the honest parties. Notice that the second condition, namely stopping the participation in the Sh_j instances of the parties not in $\mathcal{T}_i$ while participating in the Rec instances, may look counter intuitive. However, it is very much required to prevent the adversarial strategy discussed in the previous section (detailed proof is given in Lemma 13). Protocol MCC is presented in Fig 10.

We now proceed to prove the properties of the protocol MCC. Most of the properties follow from the properties of the protocol CC, whose proofs are given in [8]. For the sake of completeness, we will present them here. As in [8], while proving the properties, we assume that the following event E occurs: the invocations of Sh and Rec have been "properly" completed. This means that if an honest party has terminated an instance of Sh, then a vector $\overline{S}$ of n values is fixed, such that each honest party will eventually complete the corresponding instance of Rec and output $\overline{S}$. Moreover, if the dealer of this instance of Sh is honest, then $\overline{S}$ is the vector of n values, which he has shared on behalf of the n parties. It is easy to see that the event E occurs with probability at least $1 - n\epsilon' = 1 - \epsilon$.

Lemma 12 *Conditioned on the event E, all the honest parties terminate the protocol* MCC *in constant time.*

PROOF: We structure the proof in the following way. We first show that assuming every honest party has broadcasted the message `Reconstruct Enabled`, every honest party will terminate the protocol in constant time. Then we show that there exists at least one honest party who will broadcast the `Reconstruct Enabled` message. Consequently, we prove that if some honest party broadcasts the `Reconstruct Enabled` message, then eventually every other honest party will do the same.

So let us prove the first statement. Assuming every honest party has broadcasted the `Reconstruct Enabled` message, it will hold that eventually every *honest* party P_i will receive $n-t$ such messages from the broadcast of $n-t$ honest parties and will start participating in the Rec_k instances corresponding to each $P_k \in \mathcal{T}_i$. Now it clear that if a party P_k is included in the set $\mathcal{T}_i$ of an honest P_i, then P_k will be also eventually included in the set $\mathcal{T}_j$ of every other (honest) P_j. Hence if P_i participates in Rec_k, then eventually every other honest party will do the same. Now given that the event E occurs, all invocations of Rec terminate in constant time. Also the protocol for the broadcast terminates in constant time. This proves the first statement.

We next show that there exists at least one honest party, say P_i, who will broadcast the `Reconstruct Enabled` message. First notice that till P_i broadcasts the `Reconstruct Enabled` message, every honest party will keep participating in all the instances of Sh. By the termination property of Sh, every honest party will eventually terminate the Sh instance 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 the honest P_i, the set $\mathcal{T}_i$ will eventually contain at least $n-t$ parties and hence P_i will eventually broadcast the message $(\texttt{Attach}\ T_i\ \texttt{to}\ P_i)$. Similarly, every honest party P_j will be eventually included in the set $\mathcal{G}_i$ and so $\mathcal{G}_i$ will eventually contain at least $n-t$ parties and hence P_i will broadcast the message $(P_i\ \texttt{Accepts}\ G_i)$. Similarly, the set $\mathcal{S}_i$ will eventually be of size $n-t$ and hence P_i will broadcast the `Reconstruct Enabled` message.

Now we show that once the honest P_i broadcasts the `Reconstruct Enabled` message, every other honest party P_j will also eventually do the same. It is easy to see that every party that is included in $\mathcal{T}_i$ will be also eventually included in $\mathcal{T}_j$. 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 broadcast the `Reconstruct Enabled` message. □

We now prove the following important lemma, which is at the heart of MCC. The lemma shows that in the protocol MCC, the adversary strategy shown in Fig. 9 is not possible.

Lemma 13 *Let a corrupted party P_k is included in $\mathcal{T}_j$ of an honest P_j in the protocol* MCC. *Then the values shared by P_k in the instance Sh_k are completely independent of the values shared by the honest parties in their instances of* Sh.

PROOF: Let P_i be the *first honest* party who receives the `Reconstruct Enabled` message from at least $n-t$ parties and starts participating in the Rec instances, corresponding to each party in $\mathcal{T}_i$. To prove the lemma, we first assert that a *corrupted* party P_k will be never included in the set $\mathcal{T}_j$ of *any honest* P_j, if P_k executes the instance Sh_k (and hence

Fig. 10 Multi-Bit common coin protocol.

Protocol $\mathsf{MCC}(\epsilon)$

All instances of Sh and Rec in the following protocol have error parameter $\epsilon' = \frac{\epsilon}{n}$, so as to bound the error probability of the protocol by ϵ. For $i = 1, \ldots, n$, every party $P_i \in \mathcal{P}$ executes the following code:

1. For $j = 1, \ldots, n$, corresponding to the party P_j, choose a random value x_{ij} and act as a D and execute the protocol $\mathsf{Sh}(P_i, \mathcal{P}, \vec{S_i}, \epsilon')$, to share the vector of secrets $\vec{S_i} = (x_{i1}, \ldots, x_{in})$. Let this instance of Sh be denoted as Sh_i,
2. For $j = 1, \ldots, n$, participate in the instance Sh_j.
3. Upon terminating the instance Sh_j, broadcast the message $(P_i \texttt{ terminated } j)$.
4. Create a dynamic set $\mathcal{T}_i$, which is initially $\emptyset$. Upon receiving the message $(P_k \texttt{ terminated } j)$ from the broadcast of $n - t$ parties P_k, add P_j to $\mathcal{T}_i$.
5. Wait till $|\mathcal{T}_i| = 2t + 1$. Then assign $T_i = \mathcal{T}_i$ and broadcast the message $(\texttt{Attach } T_i \texttt{ to } P_i)$.
   (We say that the secrets $\{x_{ji} | P_j \in T_i\}$ are attached to the party P_i).
6. Create a dynamic set $\mathcal{G}_i$, which is initially $\emptyset$. Add party P_j to $\mathcal{G}_i$ if
   (a) The message $(\texttt{Attach } T_j \texttt{ to } P_j)$ is received from the broadcast of P_j and
   (b) $T_j \subseteq \mathcal{T}_i$.
   Wait until $|\mathcal{G}_i| = 2t + 1$. Then assign $G_i = \mathcal{G}_i$ and broadcast the message $(P_i \texttt{ Accepts } G_i)$.
7. Create a dynamic set $\mathcal{S}_i$, which is initially $\emptyset$. Add party P_j to $\mathcal{S}_i$ if
   (a) The message $(P_j \texttt{ Accepts } G_j)$ is received from the broadcast of P_j and
   (b) $G_j \subseteq \mathcal{G}_i$.
8. Wait until $|\mathcal{S}_i| = 2t + 1$. Then do the following:
   (a) Broadcast the message `Reconstruct Enabled`. Let H_i be the current contents of $\mathcal{G}_i$.
   (b) If a party $P_j \notin \mathcal{T}_i$, then stop participating in the instance Sh_j. Later resume participating in the instance Sh_j only if P_j is included in $\mathcal{T}_i$.
9. Wait to receive the `Reconstruct Enabled` message from the broadcast of at least $n - t$ parties. On receiving these messages, participate in the instance Rec_k, corresponding to every $P_k \in \mathcal{T}_i$, to reconstruct the secrets shared by P_k in the instance Sh_k.
   (When new parties are added to $\mathcal{T}_i$, party P_i participates in the corresponding Rec instances.)
10. Let $u = \lceil 0.87n(n - 2t) \rceil$. For every party $P_j \in \mathcal{G}_i$, associate $n - 2t = t + 1$ values $(V_{j1}, \ldots, V_{j(n-2t)})$ as follows:
    (a) Set $(v_{j1}, \ldots, v_{j(n-2t)}) = \mathsf{EXT}(X_j)$, where X_j is the vector of size $2t + 1$, consisting of the jth values, reconstructed during the instances Rec_k, where $P_k \in T_j$. That is, $X_j = \{x_{kj}\}$, where $P_k \in T_j$ and x_{kj} is the jth element of the n length vector, reconstructed in the instance Rec_k.
    (b) Set $V_{jl} = v_{jl} \mod u$, for $l = 1, \ldots, n - 2t$.
11. Wait until the $n - 2t$ values associated with all the parties in H_i are computed. Then do the following:
    (a) If there exits a party $P_j \in H_i$ such that $V_{jl} = 0$ for some $l \in \{1, \ldots, n - 2t\}$, then output $(0, \ldots, 0((n - 2t) \text{ times}))$ and terminate.
    (b) If $V_{jl} = 1$ for every party $P_j \in H_i$ and every $l \in \{1, \ldots, n - 2t\}$, then output $(1, \ldots, 1((n - 2t) \text{ times}))$ and terminate.

selects his secrets to be shared on behalf of the honest parties) *only after* P_i started participating in the Rec instances corresponding to the parties in $\mathcal{T}_i$. We prove this by contradiction.

So let P_i received the `Reconstruct Enabled` message from the parties in the set $\mathcal{B}_1$, where $|\mathcal{B}_1| \geq n - t$. Moreover, assume that P_k executes the instance Sh_k only after P_i received the `Reconstruct Enabled` message from the parties in $\mathcal{B}_1$ and started participating in the Rec instances corresponding to the parties in $\mathcal{T}_i$. Furthermore, assume that P_k is still included in the set $\mathcal{T}_j$ of some honest P_j. Now $P_k \in \mathcal{T}_j$ implies that P_j must have received the message $(P_m \texttt{ terminated } k)$ from the broadcast of at least $n - t$ P_ms, say $\mathcal{B}_2$, which implies that the (honest) parties in $\mathcal{B}_2$ have terminated the instance Sh_k and there are at least $t + 1$ such honest parties in $\mathcal{B}_2$. Now $|\mathcal{B}_1 \cap \mathcal{B}_2| \geq n - 2t$ and thus there exists at least *one honest* party, say P_α, who is present in $\mathcal{B}_1$ as well as in $\mathcal{B}_2$, as $n = 3t + 1$. This implies that the honest $P_\alpha \in \mathcal{B}_1$ must have terminated the instance Sh_k before broadcasting the `Reconstruct Enabled` message; otherwise $P_\alpha \in \mathcal{B}_2$ would have stopped participating in the instance Sh_k and would never broadcast the message $(P_\alpha \texttt{ terminated } k)$; this is because during the step 8(b) of the protocol, the honest P_α would have stopped participating in the instance Sh_k while broadcasting the `Reconstruct Enabled` message if P_α has not already terminated the instance Sh_k. This further implies that P_k must have executed the instance Sh_k (which the honest P_α have completed) *before* P_i started participating in the Rec instances. 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 the instance Sh_k before any honest party started participating in any Rec instance. Thus while choosing his own secrets for the instance Sh_k, the corrupted P_k will have no knowledge about the secrets shared by the honest parties in their instances of Sh. □

Lemma 14 *Let* $u = \lceil 0.87n(n-2t) \rceil$. *In the protocol* MCC, *once an honest party* P_j *receives the message* $(\texttt{Attach } T_i \texttt{ to } P_i)$ *from the broadcast of* P_i *and includes* P_i *in the set* $\mathcal{G}_j$, *then* $n - 2t$ *unique values, say* $V_{i1}, \ldots, V_{i(n-2t)}$ *are fixed such that the following holds:*

1. *Every honest party will associate $V_{i1}, \ldots, V_{i(n-2t)}$ with P_i, except with probability ϵ.*
2. *Each value $V_{i1}, \ldots, V_{i(n-2t)}$ 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 the step 10 of the protocol. We now prove the first part of the lemma. According to the lemma condition, $P_i \in \mathcal{G}_j$. This implies that $T_i \subseteq \mathcal{T}_j$. So the honest P_j will participate in the instance Rec_k, corresponding to each $P_k \in T_i$. Moreover, eventually $T_i \subseteq \mathcal{T}_m$ and $P_i \in \mathcal{G}_m$ will hold for *every other honest* party P_m. So, every other honest party will also eventually participate in the instance Rec_k corresponding to each $P_k \in T_i$. Now by the property of Rec, each honest party will eventually reconstruct $\vec{S_k} = (x_{k1}, \ldots, x_{ki}, \ldots, x_{kn})$ at the completion of Rec_k, except with probability ϵ'. Thus, with probability $1 - (n-t)\epsilon' \approx 1 - \epsilon$, every honest party will correctly have the vector X_i during the step 10 and hence will associate the values $V_{i1}, \ldots, V_{i(n-2t)}$ with P_i.

We now prove second part of the lemma. By Lemma 13, 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 T_i contains $n-t$ parties and hence at least $n-2t$ honest parties and every honest partys' shared secrets are uniformly distributed and mutually independent. This implies that in the vector X_i, there are at least $n-2t$ uniformly random values and so from the properties of EXT, the values $v_{i1}, \ldots, v_{i(n-2t)}$ computed from X_i will be completely random. Finally, since each V_{il} is computed as v_{il} modulo u, the values $V_{i1}, \ldots, V_{i(n-2t)}$ will be uniformly distributed over $[0, \ldots, u]$. □

Lemma 15 *In the protocol* MCC*, once an honest party broadcasts the message* `Reconstruct Enabled`*, there exists a set M of size $|M| \geq \frac{n}{3}$, such that the following holds:*

1. *For every party $P_j \in M$, some honest party has received the message* (`Attach` T_j `to` P_j) *from the broadcast of P_j.*
2. *When any honest party P_j broadcasts the message* `Reconstruct Enabled`*, then it will hold that $M \subseteq H_j$.*

PROOF: Let P_i be the *first honest party* to broadcast the message `Reconstruct Enabled`. Then let M be the set of parties P_k, who belongs to the set $\mathcal{G}_l$ of at least $t+1$ parties P_l, who are present in the set $\mathcal{S}_i$, when P_i broadcasted the `Reconstruct Enabled` message. We claim that this set M has all the properties as stated in the lemma.

It is clear that $M \subseteq H_i$. Thus the party P_i must have received the message (`Attach` T_j `to` P_j) from the broadcast of every $P_j \in M$. So this proves the first part of the lemma.

An *honest* P_j broadcasts the message `Reconstruct Enabled` only when $\mathcal{S}_j$ contains $2t+1$ parties. Now note that $P_k \in M$ implies that P_k belongs to $\mathcal{G}_l$ of at least $t+1$ parties P_l, who are present in $\mathcal{S}_i$. 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 $\mathcal{G}_l \subseteq \mathcal{G}_j$. This implies that $P_k \in M$ belongs to $\mathcal{G}_j$ before the party P_j broadcasted the `Reconstruct Enabled` message. Since H_j is the instance of $\mathcal{G}_j$ at the time when P_j broadcasted the `Reconstruct Enabled` message, 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 belongs to H_j, thus proving the second part of the lemma.

To complete the lemma, it remains to show that $|M| \geq \frac{n}{3}$, for which we use a counting argument. Let $m = |\mathcal{S}_i|$ at the time when P_i broadcasted the `Reconstruct Enabled` message. 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 $k, l \in \{1, \ldots, n\}$ if and only if the following holds: (a) P_i has received the message (P_l `Accepts` G_l) from the broadcast of P_l and included P_l in $\mathcal{S}_i$ before broadcasting the `Reconstruct Enabled` message and (b)$P_k \in G_l$. The remaining entries (if any) of Λ_i are left blank. Then M is the set of parties P_k such that the 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 *Let $\epsilon \leq 0.2$ and assume that all the honest parties have terminated the protocol* MCC*. Then for every possible value $\sigma \in \{(0, \ldots, 0(\ell \text{ times})), (1, \ldots, 1(\ell \text{ times}))\}$, with probability at least $\frac{1}{4}$ all the honest parties output σ, where $\ell = (n-2t)$.*

PROOF: By Lemma 14, for every P_i that is included in the $\mathcal{G}_j$ of some *honest* party P_j, there exists some fixed (yet unknown) $n-2t$ unique values, say $V_{i1}, \ldots, V_{i(n-2t)}$, that are

distributed uniformly and independently over $[0, \ldots, u]$ and all the honest parties will associate $V_{i1}, \ldots, V_{i(n-2t)}$ with P_i. Since the parties have terminated the protocol MCC, this implies that the event E occurs. This further implies that with probability at least $(1-\epsilon)$, all the honest parties will agree on the values associated with every party, as this depends upon whether the instances of Sh and Rec have completed properly. Now we consider two cases:

- We show that the probability of outputting $\sigma = (0, \ldots, 0$ (ℓ times)) by all the honest parties is at least $\frac{1}{4}$. Let M be the set of parties guaranteed by Lemma 15. Clearly if $V_{jl} = 0$ for some $P_j \in M$ and some $l \in \{1, \ldots, \ell\}$ and if all the honest parties associate V_{jl} (as the lth value) with P_j, then clearly all the honest parties will output $(0, \ldots, 0(\ell$ times)). The probability that for at least one party $P_j \in M$, $V_{jl} = 0$ for some $l \in \{1, \ldots, \ell\}$ is $1 - (1 - \frac{1}{u})^{\ell |M|}$. Now $u = \lceil 0.87n(n-2t) \rceil$ and $\ell = (n-2t)$. Also $|M| \geq \frac{n}{3}$. Therefore for all $n > 4$, we have $1 - (1 - \frac{1}{u})^{\ell |M|} \geq 0.316$. So, the probability that all the honest parties output $\sigma = (0, \ldots, 0(\ell$ times)) is at least $\geq 0.316 \times (1-\epsilon) \geq 0.25 = \frac{1}{4}$.
- We show that the probability of outputting $\sigma = (1, \ldots, 1$ (ℓ times)) by all the honest parties is at least $\frac{1}{4}$. It is obvious that if *no* party P_j has $V_{jl} = 0$ for any $l \in \{1, \ldots, \ell\}$ and if all honest parties associate V_{jl} with P_j, then all the honest parties will output 1. As $u = \lceil 0.87n(n-2t) \rceil$ and $\ell = (n-2t)$, the probability of this event is at least $(1 - \frac{1}{u})^{\ell n} \cdot (1-\epsilon) \geq e^{-1.15} \cdot 0.8 \geq 0.25 = \frac{1}{4}$. □

Theorem 7 *For every ϵ, where $0 < \epsilon \leq 0.2$, protocol* MCC *is a $(1-\epsilon)$-completing, t-resilient, $\frac{1}{4}$-multibit common coin protocol, with $(n-2t) = \Theta(n)$ bits output. Conditioned on the event that all the honest parties terminate the protocol, they do so in constant time. The protocol requires a private communication of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits and broadcast of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits.*

PROOF: In the protocol MCC, each party executes an instance of Sh to share n secrets and the corresponding instance of Rec is executed to reconstruct the n secrets. So the communication complexity of MCC follows from Theorem 6 by substituting $\ell = n$. The theorem now follows from Lemma 12-16. □

We end this section by comparing the protocols CC and MCC.

Comparison of CC **and** MCC : The protocol CC is a common coin protocol with one bit output. If we use our Sh and Rec protocols of the AVSS scheme AVSS − Multiple (which is the most efficient AVSS scheme) in the protocol CC, then it would require a private communication, as well as broadcast communication of $\mathcal{O}(n^5 \log \frac{1}{\epsilon})$ bits. This is because there will be $\Theta(n^2)$ instances of Sh and Rec, each dealing with a single secret (i.e. $\ell = 1$). This further implies that if we execute $t+1$ independent instances of the protocol CC, we can get a $t+1$ bit output common coin protocol, where the private as well as the broadcast communication will be $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits. Now comparing this with the complexity figures in Theorem 7, we see that we get a saving of $\Theta(n^2)$ by using the protocol MCC, instead of executing $t+1$ instances of CC. The saving comes by asking every party to use a single instance of Sh to share n secrets concurrently and then associating $n-2t$ values with a party from the $n-t$ values attached to the party.

7 Existing Voting Protocol

In this section, we recall the existing vote protocol from [8], which will be required for the construction of our ABA protocol. For the sake of completeness, the proofs of the properties of the protocol are presented in **APPENDIX B**.

Informally, the voting protocol does "whatever can be done deterministically" to reach agreement. In a Voting protocol, every party has a single bit as input. The protocol tries to find out whether there is a detectable majority for some value among the inputs of the parties. In the protocol, each party's output can have *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. The Output $(\Lambda, 0)$ stands for "non-distinct majority".

The voting protocol will have the following properties:

1. If all the honest parties have the same input σ, then all the 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 voting protocol consists of three "stages", each having a similar structure. The protocol called VOTE is presented in Fig 11. In the protocol, party P_i has the input bit x_i.

The properties of the protocol VOTE are stated in the following lemmas, whose proofs are available in [8]. For the sake of completeness, the proofs are recalled in **APPENDIX B**.

Lemma 17 ([8]) *All the honest parties terminate the protocol* VOTE *in constant time.*

Lemma 18 ([8]) *If all the honest parties have the same input σ, then all the honest parties will output $(\sigma, 2)$.*

Fig. 11 Existing vote protocol.

Protocol VOTE

For $i = 1, \ldots, n$, every party $P_i \in \mathcal{P}$ executes the following code:

1. On input x_i, broadcast $(\texttt{input}, P_i, x_i)$.
2. Create a dynamic set $\mathcal{A}_i$, which is initially $\emptyset$. Add (P_j, x_j) to $\mathcal{A}_i$ if $(\texttt{input}, P_j, x_j)$ is received from the broadcast 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 broadcast $(\texttt{vote}, P_i, A_i, a_i)$.
4. Create a dynamic set $\mathcal{B}_i$, which is initially $\emptyset$. Add (P_j, A_j, a_j) to $\mathcal{B}_i$ if $(\texttt{vote}, P_j, A_j, a_j)$ is received from the broadcast 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 broadcast $(\texttt{re-vote}, P_i, B_i, b_i)$.
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 broadcast 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 19 ([8]) *If some honest party outputs $(\sigma, 2)$, then every other honest party will eventually output either $(\sigma, 2)$ or $(\sigma, 1)$.*

Lemma 20 ([8]) *If some honest party outputs $(\sigma, 1)$ and no honest party outputs $(\sigma, 2)$ then every other honest party will output either $(\sigma, 1)$ or $(\Lambda, 0)$.*

The communication complexity of the protocol VOTE is stated in the following theorem.

Theorem 8 *Protocol* VOTE *requires a broadcast of $\mathcal{O}(n^2 \log n)$ bits.*

PROOF: In the protocol, each party may broadcast A, B and C sets, each containing the identity of $n - t$ parties. Since the identity of each party can be represented by $\log(n)$ bits, clearly the protocol requires broadcast of $\mathcal{O}(n^2 \log n)$ bits. □

□

8 Multi-Bit ABA Protocol

Once we have the $(n-2t)$ bit common coin protocol MCC and the VOTE protocol, we can design our multi-bit ABA protocol to reach agreement on $(n - 2t)$ bits concurrently, by extending the idea used in [8]. We first informally discuss the underlying idea used in [8] for reaching agreement on a *single* bit by using the CC protocol along with the protocol VOTE; the same idea is extended in a "natural" way in our protocol for reaching agreement on $(n-2t)$ bits concurrently.

The Underlying Idea for Agreement on a Single Bit : The ABA protocol (for a single bit) proceeds in "iterations", where in each iteration every party computes a "modified input" value. In the first iteration, the modified input of a party P_i is his private input bit (for the ABA protocol) x_i. In each iteration, the parties execute an instance of the protocol VOTE and CC *sequentially*, that is, a party participates in the instance of CC only after terminating the instance of VOTE (the reason for this provision will be clear while proving the properties of the ABA protocol). If a party outputs $(\sigma, 1)$ in the instance of the VOTE protocol, implying that he finds a "distinct majority" for the value σ, then he sets his modified input for the next iteration to σ, irrespective of the value which is going to be output in the instance of CC; otherwise, he sets his modified input for the next iteration to be the output of the CC protocol, which is invoked by all the parties in each iteration, irrespective of whether the output of the CC protocol is used or not by the parties for setting the modified inputs for the next iteration. Once a party outputs $(\sigma, 2)$ in an instance of the VOTE protocol, implying that he finds an "overwhelming majority" for the value σ, then he broadcasts σ. Finally, once a party receives σ from the broadcast of $t + 1$ parties, he outputs σ and terminates.

Extending the Idea for $(n-2t)$ Bits : In our multi-bit ABA protocol, we extend the above idea as follows: during the first iteration, the "modified input" for each party will consists of his private $(n - 2t)$ input bits, so each party will have $(n-2t)$ modified input bits. Then in each iteration, the parties execute $(n-2t)$ parallel instances of the VOTE protocol (one instance on behalf of each bit), followed by a single instance of the MCC protocol (on behalf of all the $(n - 2t)$ bits). Note that a party participates in the instance of MCC only after terminating all the $(n - 2t)$ instances of the VOTE protocol. The way a party sets his modified bits for the next iteration, depending upon the outcome of the VOTE and MCC protocol is exactly the same as explained above, the only difference being that now this is done parallely for $(n - 2t)$ bits, instead of a single bit.

More specifically, after completing the iteration k, each party sets the lth bit of his modified input for the $(k + 1)$th iteration as follows, for each $l \in \{1, \ldots, n - 2t\}$: if $(\sigma_l, 1)$

Fig. 12 Multi-bit ABA protocol to reach agreement on $(n-2t) = t+1$ bits.

MABA(ϵ)

Each instance of MCC in the following protocol is executed with an error parameter $\epsilon' = \frac{\epsilon}{4}$ to bound the error probability of the protocol by ϵ. For $i = 1, \ldots, n$, each party $P_i \in \mathcal{P}$ executes the following code:

1. Set $r = 0$. On having the input $(x_{i1}, \ldots, x_{i(n-2t)})$, set $v_{1l} = x_{il}$, for $l = 1, \ldots, n-2t$ and set the $\mathsf{flag}_l = 0$.[a]
2. Repeat until terminating:
   (a) Set $r = r+1$. Participate in $(n-2t)$ instances of the VOTE protocol, with v_{rl} as the input for the l^{th} instance of the VOTE protocol, for $l = 1, \ldots, (n-2t)$. Set (y_{rl}, m_{rl}) as the output of the l^{th} instance of the VOTE protocol.
   (b) Wait to terminate all the $(n-2t)$ instances of the VOTE protocol, executed in the previous step. Then invoke MCC(ϵ') and wait until its termination with output $(c_{r1}, \ldots, c_{r(n-2t)})$.
   (c) For every $l \in \{1, \ldots, n-2t\}$, such that $\mathsf{flag}_l = 0$, do the following:
      i. If $m_{rl} = 2$, then set $v_{(r+1)l} = y_{rl}$, broadcast (`Terminate with output` $v_{(r+1)l}, l$) and participate in only one more instance of VOTE protocol corresponding to the l^{th} bit with $v_{(r+1)l}$ as the input.[b]
      Participate in only one more instance of MCC if (`Terminate with output` $v_{(r+1)l}, l$) is broadcasted for all $l = 1, \ldots, n-2t$.[c]
      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 output` σ_l, l) from the broadcast of at least $t+1$ parties, for some value σ_l, output σ_l as the l^{th} bit, terminate all the computation regarding l^{th} bit and set $\mathsf{flag}_l = 1$.
   (e) If $\mathsf{flag}_l = 1$ for every $l = 1, \ldots, n-2t$ then terminate the protocol with output $(\sigma_1, \ldots, \sigma_{n-2t})$.

[a] Here $\mathsf{flag}_1, \ldots, \mathsf{flag}_{n-2t}$ are the (local) Boolean flags to indicate whether the agreement on the lth bit has been achieved.
[b, c] The purpose of these restrictions is to prevent the parties from participating in an unbounded number of iterations before enough (`Terminate with output` σ_l, l) broadcasts are completed.

is obtained as the output of the lth instance of VOTE during the kth iteration, then the lth bit is set as σ_l; otherwise the lth bit is set as the lth output bit obtained at the end of MCC protocol during the kth iteration. This process is repeated till $(\sigma_l, 2)$ is obtained as the output of the lth instance of VOTE during some iteration, in which case, a party stops all computations related to the lth bit and broadcasts σ_l. Our multi-bit ABA protocol, called MABA, is presented in Fig 12.

We now proceed to prove the properties of the protocol MABA; most of the proofs follow from the properties of the single bit ABA protocol provided in [8], but for the sake of completeness we provide them here.

Lemma 21 *In the protocol* MABA*, if all the honest parties have the same input* $(\sigma_1, \ldots, \sigma_{n-2t})$*, then all the honest parties terminate and output* $(\sigma_1, \ldots, \sigma_{n-2t})$.

PROOF: The proof follows from the fact that if all the honest parties have the same input $(\sigma_1, \ldots, \sigma_{n-2t})$, then by Lemma 18, during the first iteration every honest party will output $(y_{1l}, m_{1l}) = (\sigma_l, 2)$ upon terminating the lth instance of the VOTE protocol and will consequently broadcast (`Terminate with output` σ_l, l). □

Lemma 22 *If some honest party terminates the protocol* MABA *with output* $(\sigma_1, \ldots, \sigma_{n-2t})$*, then all the honest parties will eventually terminate* MABA *with output* $(\sigma_1, \ldots, \sigma_{n-2t})$.

PROOF: To prove the lemma, it is enough to show that for every $l = 1, \ldots, n-2t$, if an honest party outputs σ_l as the lth bit, then all the honest parties will also eventually output σ_l as the lth bit. We first claim that if an honest party broadcasts (`Terminate with output` σ_l, l), then eventually every other honest party will also do the same. Let k be the first iteration when an *honest* party P_i broadcasts (`Terminate with output` σ_l, l); we show that every other honest party will broadcast the same either in the kth iteration or in the $(k+1)$th iteration. Since the honest P_i has broadcasted (`Terminate with output` σ_l, l) during the kth iteration, it implies that $y_{kl} = \sigma_l$ and $m_{kl} = 2$, which further implies that P_i has obtained $(\sigma_l, 2)$ as the output of the lth instance of the VOTE protocol, invoked during the kth iteration. So by Lemma 19, every other honest party P_j will output either $(\sigma, 2)$ or $(\sigma, 1)$ during this instance of the VOTE protocol. In case P_j outputs $(\sigma, 2)$, the it will broadcast (`Terminate with output` σ_l, l) during the kth iteration itself. Furthermore every honest P_j will execute the lth instance of the VOTE during the $(k+1)$th iteration with input $v_{(k+1)l} = \sigma_l$. So clearly, during the $(k+1)$th iteration, every honest party will have the same input σ_l for the lth instance of VOTE. Therefore by Lemma 18, every honest party will output $(\sigma_l, 2)$ during this instance of the VOTE protocol. Thus all the honest parties broadcast (`Terminate with output` σ_l, l) either during the kth iteration or during the $(k+1)$th iteration.

Now suppose that an honest party outputs σ_l as the lth bit, so at least one honest party must have broadcasted (`Terminate with output` σ_l, l). Consequently, all the honest parties will also broadcast the same. So eventually, every honest party will receive (`Terminate with output` σ_l, l) from the broadcast of $(n-t)$ parties and (`Terminate with output` $\overline{\sigma_l}, l$) from the broadcast of at most t cor-

rupted parties. Therefore every honest party will output σ_l as the lth bit. □

Lemma 23 *If all the honest parties have initiated and completed some iteration k, then with probability at least $\frac{1}{4}$, all the honest parties will have the same modified inputs $\{v_{(k+1)l} : l \in \{1,\ldots,n-2t\}$ and $\mathsf{flag}_l = 0\}$ for the $(k+1)$th iteration*[13].

PROOF: For every $l \in \{1,\ldots,n-2t\}$ where $\mathsf{flag}_l = 0$, the modified input $v_{(k+1)l}$ for the $(k+1)$th iteration is set based on the outcome of either the lth instance of the VOTE protocol (during the kth iteration) or the outcome of the MCC protocol (during the kth iteration). For all such $v_{(k+1)l}$s, we have two possible cases:

- For all ls for which *all* the honest parties executed step 2(c)-(iii) in iteration k for setting $v_{(k+1)l}$, it holds that $v_{(k+1)l}$ is set to the lth bit of the (local) output obtained in the instance of MCC. The property of MCC ensures that with probability at least $\frac{1}{4}$ all the honest parties obtain the same ($(n-2t)$ bit) output after completing MCC and hence with the same probability, all the honest parties will have the same value for all the $v_{(k+1)l}$s, which are set to the lth bit of the output of MCC.
- For all ls for which *some* honest party has set $v_{(k+1)l} = \sigma_l$ for some $\sigma_l \in \{0,1\}$, either in step 2(c)-(i) or step 2(c)-(ii) of iteration k, it holds that no honest party will set $v_{(k+1)l} = \overline{\sigma_l}$ in step 2(c)-(i) or step 2(c)-(ii) (this follows from Lemma 20). Moreover, the probability that for all such ls, all the honest parties have σ_l as the lth bit of their (local) output of MCC is the same as the probability that all the honest parties have the same ($(n-2t)$ bit) output at the end of MCC, which is at least $\frac{1}{4}$. Now the parties start executing MCC, only after the termination of VOTE. Hence the outcome of VOTE is *fixed*, before MCC is invoked. Thus the corrupted parties can not force the output of VOTE to prevent agreement. Hence with probability at least $\frac{1}{4}$, all the honest parties will set $v_{(k+1)l} = \sigma_l$. □

Now before proceeding further, let us define the following event C_k: let C_k be the event that each honest party completes all the iterations it initiated, up to (and including) the kth iteration. That is, for each iteration $1 \leq r \leq k$ and for each party P, if P initiated iteration r then it computes $v_{(r+1)l}$, for every $l \in \{1,\ldots,n-2t\}$, for which $\mathsf{flag}_l = 0$. Let C denote the event that C_k occurs for all k.

Lemma 24 *Conditioned on the event C, all the honest parties terminate the protocol* MABA *in constant expected time.*

PROOF: First notice that in order to terminate MABA, each honest party must set $\mathsf{flag}_l = 1$, for all $l \in \{1,\ldots,n-2t\}$. We first claim that for each $l \in \{1,\ldots,n-2t\}$, all the honest parties will set $\mathsf{flag}_l = 1$ within constant time, after the *first* instance when some honest party broadcasts $(\texttt{Terminate with output}\ \sigma_l, l)$, for some $\sigma_l \in \{0,1\}$. So let the first instance when an honest party broadcasts $(\texttt{Terminate with output}\ \sigma_l, l)$ occurs during the iteration r_l. This implies that all the honest parties participated in the lth instance of the VOTE and in the instance of MCC of all the iterations upto iteration $r_l + 1$. From the proof of Lemma 22, it follows that all the honest parties will broadcast $(\texttt{Terminate with output}\ \sigma_l, l)$ by the end of iteration $r_l + 1$. All these instances of broadcast complete in constant time. Moreover, each honest party will set $\mathsf{flag}_l = 1$ after completing $t+1$ of these broadcasts and thus, after the first instance when some honest party broadcasts $(\texttt{Terminate with output}\ \sigma_l, l)$, all the honest parties will set $\mathsf{flag}_l = 1$ in constant time.

Now let r_{max} be the maximum among $r_1,\ldots,r_{n-2t}$. It is easy to see that all the honest parties will set $\mathsf{flag}_l = 1$, for all $l \in \{1,\ldots,n-2t\}$ in constant time after the iteration $r_{max}+1$. This is because by the iteration $r_{max}+1$, all the honest parties must have broadcasted $(\texttt{Terminate with output}\ \sigma_l, l)$, for every $l \in \{1,\ldots,n-2t\}$ (this follows from the definition of r_{max}) and all these broadcasts will terminate in constant time.

Let the random variable τ_{max} count r_{max}; if $\tau_{max} = \infty$, then the honest parties will not terminate MABA, as the honest parties will wait to set flag_{max} to 1. Conditioned on the event C, all the honest parties terminate each iteration in constant time. To complete the proof, it is enough to show that $E(\tau_{max}|C)$ is constant. We have

$$Prob(\tau_{max} > k|C_k) \leq Prob(\tau_{max} \neq 1|C_k) \times \ldots \times Prob(\tau_{max} \neq k|C_k \cap \tau_{max} \neq 1 \ldots \cap \tau_{max} \neq k-1).$$

From Lemma 23, after completing an iteration k, all the honest parties will have the same modified inputs for the $(k+1)$th iteration, except with probability at most $\frac{3}{4}$. This implies that each of the k multiplicands on the right hand side of the above equation is at most $\frac{3}{4}$ and thus $Prob(\tau_{max} > k|C_k) \leq (\frac{3}{4})^k$. Now it follows via a simple calculation that $E(\tau_{max}|C) \leq 16$. □

Lemma 25 *In the protocol* MABA, $Prob(C) \geq (1-\epsilon)$.

PROOF: As in Lemma 24, let r_{max} be the maximum among $r_1,\ldots,r_{n-2t}$. We have

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

From the proof of Lemma 24, we have $Prob(\tau_{max} > k|C_k) \leq (\frac{3}{4})^k$. We will now bound the term $Prob(\overline{C_{k+1}}|C_k \cap \tau_{max} \geq$

[13] Note that here the probability $\frac{1}{4}$ is not for the each individual l, but instead for all ls simultaneously.

k). If all the honest parties execute the kth iteration and complete the instance of MCC during the kth iteration, then all the honest parties complete the kth iteration (as the instances of VOTE will always complete in each iteration). Now the instance of MCC is invoked with error parameter $\epsilon' = \frac{\epsilon}{4}$. Thus with probability at least $1 - \frac{\epsilon}{4}$, all the honest parties complete the instance of MCC during the kth iteration. Therefore, for each k, $Prob(\overline{C_{k+1}}|C_k \cap \tau_{max} \geq k) \leq \frac{\epsilon}{4}$. So we get

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

Theorem 9 (Multi-Bit ABA) *Let $n = 3t + 1$. Then for every $0 < \epsilon \leq 0.2$, protocol* MABA *is a $(1 - \epsilon)$-terminating, multi-bit ABA protocol with $(n - 2t)$ bit output. The protocol requires a private communication as well as broadcast of $\mathcal{O}(\mathsf{R}n^4 \log \frac{1}{\epsilon})$ bits, where* R *is the expected running time of the protocol. Given that the parties terminate, they do so in constant expected time (i.e.* $\mathsf{R} = \mathcal{O}(1)$*).*

PROOF: In each iteration of the protocol MABA, one instance of MCC and $(n-2t)$ instances of VOTE are executed, which requires a total private as well as broadcast communication of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits. Moreover, from the proof of Lemma 24, there will be (expected) constant number of such iterations. The theorem now follows from Lemma 21-25. $\square$

9 Conclusion and Open Problems

We have presented a $(1-\epsilon)$-terminating unconditional ABA protocol with optimal resilience, which significantly improves the communication complexity of the best known $(1 - \epsilon)$-terminating ABA protocol of [9]. Our protocol also improves the communication complexity of the almost surely terminating ABA protocol of [1] (though the ABA protocol of [1] has a stronger property of being almost surely terminating). The key factors that have contributed to the gain in the communication complexity of our ABA protocol are

- Using a shorter route *AICP* $\rightarrow$ *AWC* $\rightarrow$ *AVSS* to get our AVSS scheme and to introduce the new primitive AWC, which can be designed more efficiently that AWSS, the commonly used primitive in the AVSS of [9] and in the SVSS of [1].
- Improving each of the underlying building blocks, so as to deal with multiple values concurrently.
- Modifying the existing common coin protocol to make it compatible to use our AVSS scheme (sharing multiple secrets concurrently) and to generate $\Theta(n)$ common coins concurrently.

An interesting open problem is to further improve the communication complexity of the 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].

Acknowledgements: We would like to sincerely thank the anonymous referees for their comments which helped us to significantly improve the overall presentation of the paper.

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. H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, 2004.
3. 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.
4. C. H. Bennett, G. Brassard, C. Crépeau, and U. M. Maurer. Generalized privacy amplification. *IEEE Transactions on Information Theory*, 41(6):1915–1923, 1995.
5. C. H. Bennett, G. Brassard, and J. Robert. Privacy amplification by public discussion. *SIAM J. Comput.*, 17(2):210–229, 1988.
6. 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.
7. 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.
8. R. Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute, Israel, 1995.
9. 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. Full version available at http://citeseerx.ist.psu/viewdoc/summary?doi=10.1.1.8.8120.
10. 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.
11. R. Cramer and I. Damgård. *Multiparty Computation, an Introduction*. Contemporary Cryptography. Birkhuser Basel, 2005.
12. 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.
13. 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.

14. D. Dolev and R. Reischuk. Bounds on information exchange for Byzantine Agreement. *Journal of ACM*, 32(1):191–204, 1985.
15. 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.
16. P. Feldman and S. Micali. An optimal probabilistic protocol for synchronous Byzantine Agreement. *SIAM Journal of Computing*, 26(4):873–933, 1997.
17. M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *JACM*, 32(2):374–382, 1985.
18. M. Fitzi. *Generalized Communication and Security Models in Byzantine Agreement*. PhD thesis, ETH Zurich, 2002.
19. 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.
20. 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.
21. 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.
22. 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.
23. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
24. 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.
25. 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.
26. M. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *JACM*, 27(2):228–234, 1980.
27. 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.
28. 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.
29. 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.
30. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

APPENDIX A: Communication Complexity Analysis of the AVSS and ABA of [9]

The communication complexity analysis of the AVSS and ABA protocol of [9] 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 [9] given in Canetti's Thesis [8]. To bound the error probability by ϵ, all the communication and computation in the protocol of [9] 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 $\kappa = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

To begin with, in the ICP protocol of [9], Signer gives $\mathcal{O}(\kappa)$ field elements to INT and $\mathcal{O}(\kappa)$ field elements to Verifier. Though the ICP protocol of [8] is presented with a *single* Verifier, it is executed with n verifiers in the protocol A-RS. In order to execute ICP with n verifiers, Signer 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 [8] 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 [30], the authors in [9] 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 [9] involves a private communication of $\mathcal{O}(n^2\kappa^2)$ bits. In addition to this, the A-RS-Share protocol also involves broadcast 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 broadcast of $\mathcal{O}(n^2 \log(n))$ bits.

Proceeding further, the authors in [9] designed an AWSS scheme using their A-RS protocol. 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 [30]) of the secret and instantiate n instances of the ICP protocol for each of the n shares. Now each individual party A-RS-Share (as a D) 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 broadcast 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 requires a private communication of $\mathcal{O}(n^5\kappa^3)$ bits and broadcast $\mathcal{O}(n^5\kappa\log(n))$ bits.

The authors in [9] 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 of private communication and $\mathcal{O}(n^2\kappa^2\log(n))$ bits of broadcast communication.

Finally using their Two&Sum AWSS-Share and AWSS-Rec protocol, the authors in [9] 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 reconstruct $\mathcal{O}(n^3\kappa)$ field elements by executing instances of AWSS-Rec. So in total, the AVSS-Share protocol of [9] involves a private communication of $\mathcal{O}(n^9\kappa^4)$ bits and broadcast of $\mathcal{O}(n^9\kappa^2\log(n))$ bits. The AVSS-Rec protocol involves n instances of AWSS-Rec, resulting in a private communication of $\mathcal{O}(n^6\kappa^3)$ bits and broadcast 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 [9] is $\mathcal{O}(n^{11}\kappa^4)$ bits of private communication and $\mathcal{O}(n^{11}\kappa^2\log(n))$ bits of broadcast communication. Now in the ABA protocol of [9], common coin protocol is called for $\mathsf{R} = \mathcal{O}(1)$ expected time. Hence the ABA protocol of [9] involves a private communication of $\mathcal{O}(n^{11}\kappa^4)$ bits and broadcast of $\mathcal{O}(n^{11}\kappa^2\log(n))$ bits. As mentioned earlier, $\kappa = \mathcal{O}(\log\frac{1}{\epsilon})$. Thus the ABA protocol of [9] involves a private communication of $\mathcal{O}(n^{11}\log(\frac{1}{\epsilon})^4)$ bits and broadcast of $\mathcal{O}(n^{11}\log(\frac{1}{\epsilon})^2\log(n))$ bits.

APPENDIX B: Proofs for the Protocol VOTE

Lemma 17 [8] *All the honest parties terminate the protocol* VOTE *in constant time*.

PROOF (SKETCH): Every honest party P_i will broadcast his input x_i. As there are at least $n-t$ honest parties, from the properties of broadcast, 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 [8] *If all the honest parties have the same input σ, then all the honest parties will output $(\sigma, 2)$.*

PROOF: Consider an honest party P_i. If all the honest parties have the same input σ, then at most t (corrupted) parties may broadcast $\overline{\sigma}$ as their input. Therefore, it is easy to see that *every* $P_k \in \mathcal{B}_i$ must have broadcasted his vote $b_k = \sigma$. Hence the honest P_i will output $(\sigma, 2)$. □

Lemma 19 [8] *If some honest party outputs $(\sigma, 2)$, then every other honest party will output either $(\sigma, 2)$ or $(\sigma, 1)$.*

PROOF: Let an *honest* P_i outputs $(\sigma, 2)$. This implies that every $P_j \in B_i$ had broadcasted vote $a_j = \sigma$. As $|B_i| = 2t+1$, it implies that for every other *honest* party P_j, it holds that $|B_i \cap B_j| \geq t+1$ and so P_j is bound to broadcast re − vote $b_j = \sigma$ and hence will output either $(\sigma, 2)$ or $(\sigma, 1)$. □

Lemma 20 [8] *If some honest party outputs $(\sigma, 1)$ and no honest party outputs $(\sigma, 2)$ then every other honest party will output either $(\sigma, 1)$ or $(\Lambda, 0)$.*

PROOF: Assume that an *honest* party P_i outputs $(\sigma, 1)$. This implies that all the parties $P_j \in C_i$ had broadcasted 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 broadcast 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-voted as σ, there must be at least $t+1$ parties who have broadcasted 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 broadcast 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 $(\Lambda, 0)$. □