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

Asynchronous Byzantine Agreement with Optimal Resilience

Arpita Patra · Ashish Choudhury · C. Pandu Rangan

Received: date / Accepted: date

Abstract We present an efficient, optimally-resilient Asynchronous Byzantine Agreement (ABA) protocol involving $n = 3t+1$ parties over a completely asynchronous network, tolerating a computationally unbounded Byzantine adversary, capable of corrupting at most t out of the n parties. In comparison with the best known optimally-resilient ABA protocols of Canetti and Rabin (STOC 1993) and Abraham, Dolev and Halpern (PODC 2008), our protocol is significantly more efficient in terms of the communication complexity.

Our ABA protocol is built on a new statistical asynchronous verifiable secret sharing (AVSS) protocol with optimal resilience. Our AVSS protocol significantly improves the communication complexity of the only known statistical and optimally-resilient AVSS protocol of Canetti et al. Our AVSS protocol is further built on an asynchronous primitive called asynchronous weak commitment (AWC), while the AVSS of Canetti et al. is built on the primitive called asynchronous weak secret sharing (AWSS). We observe that AWC has weaker requirements than AWSS and hence it can be designed more efficiently than AWSS.

The common coin primitive is one of the most important building blocks for the construction of an ABA protocol. In this paper, we extend the existing common coin protocol to make it compatible with our new AVSS protocol that shares multiple secrets simultaneously. As a byproduct, our new common coin protocol is more communication efficient than all the existing common coin protocols.

Keywords Byzantine agreement, Computationally unbounded, Secret sharing, Common coin.

Few of the results in this paper appeared in PODC 2009 [22] and PODC 2012 [9]. The work in this paper was initiated when the first two authors were PhD students at Department of Computer Science and Engineering, IIT Madras.

Arpita Patra
Department of Computer Science
University of Bristol, United Kingdom
E-mail: arpita.patra@bristol.ac.uk,arpitapatra10@gmail.com

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 [23] 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 out of the n parties may act in any arbitrary manner to make the remaining parties disagree. The BA problem has been investigated extensively in various models (see for example [20, 15, 7, 2] and their references). Studying the BA problem in an asynchronous setting is interesting, since an asynchronous network models a real-life network like the Internet more appropriately than a synchronous network. The problem of asynchronous BA (called ABA) has received 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 (but finitely) delayed. The inherent difficulty in designing an asynchronous protocol is that it is impossible to 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 (to avoid endless waiting) and the communication from t (potentially slow but honest) parties may have to be ignored; this poses new challenges for constructing asynchronous protocols.

Compared to the synchronous BA protocols, the best known ABA protocols involve huge communication complexity (more on this later). This paper aims to present 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. It was shown in [23] that a BA (and 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* ABA protocol. Fisher, Lynch and Paterson's seminal impossibility result on deterministic ABA protocols [14] implies that any (randomized) ABA protocol must have non-terminating runs, where some honest[2] party(ies) may not output any value and thus may not terminate at all. An ABA protocol is called $(1-\epsilon)$-terminating [8,7], if the honest parties terminate the protocol with probability[3] 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**: The maximum number of corruptions t, that the protocol can tolerate.
- **Communication Complexity (CC)**: The total number of bits communicated by the honest parties in the protocol. The communication complexity has two parts: the *private* communication which is done over the point-to-point secure channels[4] and the *broadcast* communication which is done in order to send a message (publicly) to everyone. The broadcast primitive in the asynchronous setting is implemented using Bracha's asynchronous broadcast protocol [6] (see Section 2.6).
- **Expected Running Time (ERT)**: We consider the expected running time (ERT) R of an ABA protocol, *conditioned on the event that all the (honest) parties terminate*; this notion of expectancy is weaker than the usual notion of expectation, where the expectancy is over all possible events. An $(1 - \epsilon)$-terminating ABA protocol may have non-terminating runs, where the (usual) expected running time will be infinite. Thus, our measure of ERT, also followed in [8,7], is with respect to the executions where the parties terminate (more on this later).

Based on the above parameters, the best known ABA protocols are summarized in Table 1.

Table 1 Summary of the best known ABA protocols; $poly(x)$ stands for polynomial in x and AST stands for almost-surely terminating.

Ref.	Type	Resilience	CC	ERT (R)
[6]	AST	$t < n/3$	$\mathcal{O}(2^n)$	$\mathcal{O}(2^n)$
[12,13]	AST	$t < n/4$	$poly(n)$	$\mathcal{O}(1)$
[8,7]	$(1-\epsilon)$	$t < n/3$	$poly(n, \frac{1}{\epsilon})$	$\mathcal{O}(1)$
[1]	AST	$t < n/3$	$poly(n)$	$\mathcal{O}(n^2)$

Over a period of time, the techniques and the design approaches of ABA protocols have evolved spectacularly. Rabin [25] 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 [6] presented a simple implementation of the common coin protocol, whose success probability is $\Theta(2^{-n})$. Feldman and Micali [12,13] came up with a common coin protocol that has a constant success probability. The essence of [12] is the reduction of implementing the common coin to that of designing an *Asynchronous Verifiable Secret Sharing* (AVSS) protocol. Informally, an AVSS protocol is a two phase protocol (sharing and reconstruction) carried out among the parties. 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 shared secret in the reconstruction phase, while preserving the secrecy of s until the reconstruction phase. Following [12,13], almost all the optimally-resilient ABA protocols including the protocols of [8,1], followed the same approach of reducing the ABA problem to that of AVSS. In this paper, we follow the same approach too.

1.2 Our Contribution

We present an optimally-resilient, $(1 - \epsilon)$-terminating ABA protocol with a private and broadcast communication of $\mathcal{O}($ R $n^4 \log \frac{1}{\epsilon})$ bits for reaching agreement on $t + 1 = \Theta(n)$ bits[5] *concurrently*; where R is the ERT of the protocol. So the (expected) *amortized* communication complexity of our protocol, for reaching agreement on a single bit, is $\mathcal{O}($R $n^3 \log \frac{1}{\epsilon})$ bits of private, as well as broadcast communication. Moreover, conditioned on the event that our ABA protocol

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

[2] A party is called honest if it is not under the control of Adv.

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

[4] We assume that every pair of parties are directly connected by a secure and authentic channel.

[5] For $n = 3t + 1$, we have $t = \Theta(n)$.

terminates, it does so in a constant expected time; i.e. $\mathsf{R} = \mathcal{O}(1)$. In Table 2, we compare our ABA protocol with the best known optimally-resilient ABA protocols of [8, 1].

Table 2 Comparison of our optimally-resilient ABA protocol with the best known optimally-resilient ABA protocols; AST stands for almost-surely terminating. The communication complexity is the expected communication complexity as it depends on the ERT of the protocol. While the protocols of [8, 1] are designed for single bit inputs, our protocol handles multiple bit inputs. In this table, we consider the amortized communication complexity of our protocol for single bit inputs.

Ref.	Type	R	CC
[8]	$(1-\epsilon)$	$\mathcal{O}(1)$	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 Paper	$(1-\epsilon)$	$\mathcal{O}(1)$	Private– $\mathcal{O}(\mathsf{R}n^3\log\frac{1}{\epsilon})$ Broadcast– $\mathcal{O}(\mathsf{R}n^3\log\frac{1}{\epsilon})$

On one hand, communication-complexity wise our ABA protocol improves over the ABA protocol of [8] by a large margin, while keeping all other properties in place (namely $(1-\epsilon)$-terminating and a constant expected running time). 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 communication complexity when $\log\frac{1}{\epsilon} < n^5\log n$.
2. Our ABA protocol has a constant ERT whereas the ABA protocol of [1] has $\mathcal{O}(n^2)$ ERT.

On the negative side, our protocol is $(1-\epsilon)$-terminating, whereas the protocol of [1] is almost-surely terminating. We now briefly discuss the approaches used in the ABA protocols of [8], [1] and the current article.

• The ABA protocol of Canetti et al. [8, 7] uses the reduction of [12, 13] from ABA to AVSS; i.e. an AVSS protocol with $n = 3t+1$ parties is constructed first. The authors in [8] followed 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 and ICP, A-RS and AWSS stand for *Information Checking Protocol*, *Asynchronous Recoverable Sharing* and *Asynchronous Weak Secret Sharing* respectively. The final AVSS protocol is highly communication intensive as well as 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[6]. The protocol allows a dealer to share a single secret and all the (honest) parties terminate the protocol with probability at least $(1-\epsilon)$.

[6] The exact communication complexity analysis of the AVSS (and the ABA) protocol of [8] was not done earlier. For the sake of completeness, we carry out the same in APPENDIX A.

• The ABA protocol of [1] followed the same reduction from ABA to AVSS as in [8], except that a variant of AVSS called *shunning* (asynchronous) VSS (SVSS) is used in place of AVSS. SVSS is a weaker notion of AVSS: if all the parties behave correctly, then SVSS satisfies all the properties of AVSS without any error; else it enables some honest party to identify at least one corrupted party, whom the honest party "shuns" from then onwards. In [1], an SVSS scheme is constructed building on a primitive called *weak* SVSS (W-SVSS). Notably W-SVSS is the "shunning" variant of the AWSS primitive, where the latter served as a building block for the AVSS protocol of [8, 7].

The use of SVSS instead of AVSS in generating the common coin causes the ABA protocol of [1] to have $\mathcal{O}(n^2)$ ERT (intuitively this is because in the worst case it requires $\mathcal{O}(n^2)$ executions of the SVSS, so that every honest party shuns every corrupted party). The SVSS protocol allows the dealer to share a single secret and requires a private communication as well as broadcast of $\mathcal{O}(n^4\log(n))$ bits.

• Although we follow the same approach of designing our ABA protocol from an AVSS protocol, we depart from the standard practice at several places:

1. We design a communication efficient, optimally-resilient AVSS protocol (i.e. with $n = 3t+1$), taking a "shorter" route, namely ICP $\rightarrow$ AWC $\rightarrow$ AVSS, rather than following the route suggested in [8]. Here AWC stands for *Asynchronous Weak Commitment*.
2. While the AVSS protocols of [8] as well as of [1] (the shunning variant) used AWSS as the building block, we replace AWSS by AWC, a new primitive introduced in this paper. We find that AWC has "weaker" requirements than AWSS and can be designed more efficiently than the existing AWSS protocols (the details are elaborated in Section 2.3.3 and Section 4.3). Specifically, while the existing AWSS and W-SVSS are based on the idea of using *bivariate* polynomials of degree t in each variable, we design an AWC protocol based on Shamir secret sharing [27] and therefore using *univariate* polynomials of degree t; this immediately implies a gain of $\Theta(n)$ in the communication complexity.
3. We extend the existing notion of ICP [8] to deal with multiple verifiers simultaneously, instead of a single verifier (see Section 2.4). Informally, ICP allows to authenticate data in the presence of a computationally unbounded adversary and it serves as the "starting point" of our AVSS as well as the AVSS protocol of [8]. Our multiple-verifier ICP readily fits in our AWC protocol. In [8], the (single-verifier) ICP was implicitly extended for multiple verifiers when it was used in the higher level primi-

tives. Presenting ICP for multiple verifiers thus provides greater conceptual clarity when it is plugged into the next level primitive which is AWC in our case. Furthermore, our multiple-verifier ICP achieves better communication complexity than the existing single-verifier ICP extended to deal with multiple verifiers.

4. We design each of the above building blocks (i.e. ICP, AWC, AVSS) to deal with multiple values concurrently (unlike the existing protocols which are designed to handle only a single value). This leads to a significant gain in the communication complexity over executing multiple instances of the protocols dealing with a single value.

Combining the above results, we obtain an AVSS protocol that significantly improves over the only known optimally-resilient (statistical) AVSS of [8] and is of independent interest. 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.

5. Finally, we make several changes to the existing common coin protocol. The best known common coin protocol of [13,7] employs AVSS sharing a single secret. Informally, in the common coin protocol of [13], each party is asked to act as a dealer and share n random secrets using n separate instances of an AVSS protocol. One can improve the communication complexity if the n instances of the AVSS (dealing with a single secret) are replaced by a single instance of an AVSS protocol which allows the dealer to share all the n secrets concurrently. When plugging our new AVSS protocol (sharing multiple secrets concurrently) for the same, we noticed that this "trivial" substitution leads to an "incorrect" common coin protocol. So we bring forth several modifications to the existing common coin protocol that allow us to use our AVSS protocol for sharing multiple secrets concurrently. As a result, our new common coin protocol is more communication efficient than the existing common coin protocol of [13,7,8]. Interestingly, the new common coin protocol is a *multi-bit* protocol, which allows the parties to generate $t + 1 = \Theta(n)$ random and independent common coins concurrently further leading to reach agreement on $t + 1$ bits concurrently.

Our multi-bit common coin protocol leads to an ABA protocol with an amortized private and broadcast communication of $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits for reaching agreement on a single bit.

Organization of the Paper: In the next section, we describe the asynchronous network model and formally define ABA, AVSS, AWC, AWSS, asynchronous ICP (AICP), single and multi-bit common coin, followed by the description of the existing tools. 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 [22] and the existing W-SVSS scheme of [1]. In Section 5, we present our AVSS scheme. In Section 6, we recall the existing common coin protocol from [7] and present our multi-bit common coin protocol. In Section 7, we recall the existing voting protocol from [7], which together with the multi-bit common coin protocol implies our ABA protocol presented in Section 8.

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. We assume that each pair of parties are directly connected by a secure and authentic channel and t out of the n parties can be under the influence of a computationally unbounded Byzantine (active) adversary, denoted as Adv. Moreover, we assume $n = 3t + 1$. The adversary Adv, completely dictates the parties under its control and can force them to deviate from the honest behavior in any arbitrary manner during the execution of a protocol. The parties not under the influence of Adv are called honest or uncorrupted.

The communication channels are asynchronous, having arbitrary, but 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. 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 [7], we consider a protocol execution in the asynchronous model as a sequence of *atomic steps*, where a single party is *active* in each such step. A party gets 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 is controlled by 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.

Running Time of an Asynchronous Protocol [8]. 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).

Let C be 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 specific choice of the 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 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 the input $\vec{x}$ and adversary Adv and conditioned on the event C. That is:

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

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 next present the definition of ABA and the other related primitives. We note that all computation and communication in our protocols are done over a finite field $\mathbb{F}$. Our ABA protocol is $(1-\epsilon)$-terminating, for a given $\epsilon > 0$. Looking ahead, to bound the error probability of our ABA by ϵ, we select a finite field $\mathbb{F} = GF(2^\kappa)$ for the minimum κ such that $\epsilon \geq \frac{4n^5(n+t-1)}{2^\kappa - n}$. Without loss of generality, we assume that n is polynomial in $\frac{1}{\epsilon}$; i.e. $n = poly(\frac{1}{\epsilon})$. So each element of $\mathbb{F}$ can be represented by $\mathcal{O}(\kappa) = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

2.1 Asynchronous Byzantine Agreement (ABA)

Definition 1 (ABA [8]) Let π be an asynchronous protocol executed among the parties in $\mathcal{P}$, where each party has a (private) binary input and a binary output. We say that π is a $(1-\epsilon)$-terminating ABA protocol for a single bit, for an allowed error parameter ϵ (with $\epsilon > 0$) if the following hold 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 at least $(1-\epsilon)$, all the honest parties eventually terminate the protocol.
2. **Correctness**: All the honest parties who terminate the protocol hold identical bit as the output. Moreover, if all the honest parties had the same input bit, say ρ, then all the honest parties output ρ upon termination.

The above definition can be easily extended for ℓ bits, where $\ell > 1$ and we call such a protocol a *multi-bit* ABA protocol.

2.2 Asynchronous Verifiable Secret Sharing (AVSS)

An AVSS protocol consists of two sub-protocols: a sharing protocol (called Sh) where a special party called dealer shares a secret among the parties and a reconstruction protocol (called Rec) where the parties reconstruct the secret from their shares without any "help" from the dealer. Formally:

Definition 2 (Asynchronous Verifiable Secret Sharing (AVSS) [8]) Let $(\mathsf{Sh}, \mathsf{Rec})$ 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 $(\mathsf{Sh}, \mathsf{Rec})$ is a $(1-\epsilon)$-AVSS scheme[7], for an allowed error parameter ϵ (where $\epsilon > 0$), if the following requirements hold for every possible 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 every *honest* party eventually terminates Sh.
  3. If all the honest parties 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[8]:
  1. If D is *honest*, then $\overline{s} = s$. Moreover, all the honest parties output $\overline{s}$ upon terminating Rec (i.e. $\overline{s}$ is the reconstructed secret), with probability at least $(1-\epsilon)$.
  2. Even if D is *corrupted*, all the honest parties output $\overline{s}$ upon terminating Rec, with probability at least $(1-\epsilon)$. This property is also called the *strong commitment* property.
- **Secrecy**: Let VIEW(s) be the random variable, 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 VIEW(s) is identically distributed for all $s \in \mathbb{F}$.

We conclude with the following note:

Note 1 An alternate definition of VSS/AVSS requires that D's committed secret $\overline{s}$ belongs to $\mathbb{F}$, instead of $\mathbb{F} \cup \{\perp\}$ [17, 19]. However, Definition 2 is "equivalent" to this alternate definition, since in Definition 2 we can say that $\overline{s} \in \mathbb{F}$, by fixing a default value in $\mathbb{F}$, which may be output in case the Rec protocol completes with a $\perp$ output. A VSS satisfying the definition of [17, 19] is required when VSS serves as the building block for secure multi-party computation (MPC) [3]. On the other hand, VSS (AVSS) satisfying our definition is enough for the construction of (asynchronous) BA

[7] Such schemes are also called statistical AVSS.

[8] We often say that D has committed/shared $\overline{s}$ during Sh; the interpretation of $\overline{s} = \perp$ will be clear during the description of our AVSS.

protocols. We also note that our definition of VSS was used in [21] to study the round complexity of VSS. □

Definition 2 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 AWSS and AWC

In this section, we first define AWSS, a "weaker" primitive than AVSS; we note that AWSS was used in [8] to design a $(1-\epsilon)$-AVSS and a "shunning variant" of AWSS was used in [1] to construct a shunning AVSS. We then introduce a new asynchronous primitive called AWC, which has weaker requirements than AWSS. We propose this new primitive as a "replacement" for AWSS in the construction of our $(1-\epsilon)$-AVSS. The section concludes with a comparison between AWSS and AWC, where we argue that indeed AWC has weaker requirements than AWSS.

2.3.1 Asynchronous Weak Secret Sharing (AWSS)

An AWSS protocol consists of two sub-protocols: a sharing protocol (WSh), where the dealer shares a secret and a reconstruction protocol (WRec), where the parties reconstruct the shared secret. The **Termination** and the **Secrecy** conditions of AVSS (see Definition 2) remain the same for AWSS, except that Sh is replaced by WSh and Rec is replaced by WRec. The **Correctness** condition of AVSS also remains the same for AWSS for the case when D is *honest*; however, the **Correctness** condition is "weakened" in AWSS for the case when D 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[9] of WRec.

Notice that the only difference between AVSS and AWSS is for the case when D is *corrupted* and the shared value $\overline{s}$ is *not equal* to $\perp$; in this case while the parties in WRec may reconstruct $\perp$, the parties in Rec will always reconstruct $\overline{s}$.

2.3.2 Asynchronous Weak Commitment (AWC)

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. Later during the decommitment protocol, Committer "decommits/opens" the committed secret and the parties verify whether this was the secret committed earlier and accordingly either accept or reject the secret. Thus AWC can be viewed as a distributed version of the classical two-party commitment schemes [24]. 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 $s^\star$ (the secret to be decommitted)[10] and all the parties upon terminating Decom either output $s^\star$ or $\perp$. 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 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 every *honest* party eventually terminates Com.
  3. If Committer invokes Decom and all the honest parties participate in Decom, then the following requirements hold:
     (a) If Committer is *honest*, then all the honest parties eventually terminate Decom.
     (b) If Committer is *corrupted* and some honest party terminates Decom, then all the honest parties eventually terminate 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 that the following requirements hold[11] :
  1. If $s^\star = \overline{s}$, then with probability at least $(1-\epsilon)$, all the honest parties output $s^\star$ (i.e. $s^\star$ is accepted as the decommitted secret), upon terminating Decom.
  2. If $s^\star \neq \overline{s}$, then with probability at least $(1-\epsilon)$, all the honest parties output $\perp$ (i.e. $s^\star$ is rejected as the decommitted secret[12]) upon terminating Decom.
- **Secrecy**: Let VIEW(s) be the random variable, denoting the view of Adv during an execution of Com, where Committer has the input s. If Committer is *honest* during Com and has not begun executing Decom, then VIEW(s) is identically distributed for all $s \in \mathbb{F}$.

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

2.3.3 Comparison between AWSS and AWC

The sharing protocol of AWSS and the commitment protocol of AWC achieves the *same* outcome, namely a distributed commitment to a unique value $\overline{s}$, which remains pri-

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

[10] The value of $s^\star$ is defined by the information disclosed by Committer during Decom.

[11] We say that Committer has committed $\overline{s}$ during Com; the interpretation of $\overline{s} = \perp$ will be clear during the description of our AWC.

[12] For an *honest* Committer, $s^\star = \overline{s}$ will always hold; moreover $\overline{s} = s$ and hence $\overline{s} \in \mathbb{F}$ will also hold. The condition $s^\star \neq \overline{s}$ may happen only for a *corrupted* Committer.

vate if D and Committer are honest. However, there is a subtle difference between the reconstruction protocol of AWSS and the decommitment protocol of AWC. Specifically, the difference is in the role that D and Committer plays respectively to ensure the termination of the respective protocols. The reconstruction protocol of an AWSS 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 in the reconstruction protocol. On the other hand, the decommitment protocol demands a special role from Committer to enforce 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 "more" communication/distribution of information to the parties may be called for during the sharing protocol of an AWSS scheme, as compared to the commitment protocol of an AWC scheme. The intuition is that the "additional" information may enable the honest parties to reconstruct the shared secret during the reconstruction protocol, even without the participation of the dealer. The intuition turns out to be right as we are able to design an AWC protocol that is more efficient than the known AWSS protocols (see Section 4.3).

Finally we note that both AWSS and AWC provides the same type of commitment. Namely if Committer is *honest* then it will decommit the committed secret and thus the secret will be accepted by the honest parties. On the other hand if Committer is *corrupted*, then it cannot commit $\overline{s} \in \mathbb{F}$ and later decommit a *different* secret $s^\star \in \mathbb{F}$. Such an attempt by a corrupted Committer (in co-operation with the corrupted parties) will cause the honest parties to output $\perp$.

2.4 Asynchronous Information Checking Protocol (AICP)

An 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. [26]. As described in [26, 8, 11], an ICP is executed among three parties: a Signer, an intermediary INT and a Verifier. Informally, an ICP consists of two stages, implemented by different protocol(s):

- **Signature-generation stage**: it consists of two phases. In the first phase, Signer computes its 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 for Verifier and hands it to Verifier. In the second phase, INT (in co-operation with Signer and Verifier) confirms whether indeed the received signature is a "valid" signature.
- **Signature-revelation stage**: here INT reveals the signature $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathsf{Verifier}, s)$, claiming that it has received it from Signer. Verifier then verifies the signature, using the verification information and either accepts or rejects the signature (and hence s).

An IC signature may be considered as the information-theoretically secure variant of digital signatures. It provides properties like unforgeability and non-repudiation; in addition, it provides information-theoretic security of the secret. That is, if Signer and INT are *honest*, then at the end of signature-generation stage, a *corrupted* Verifier does not learn any information about s in the information-theoretic sense.

We extend the notion of ICP in two directions: Firstly, we consider *multiple* verifiers, where each party in $\mathcal{P}$ acts as a Verifier. Looking ahead, the multiple-verifier ICP concept readily fits in our AWC protocol; note that Signer and INT can be any two parties from the set $\mathcal{P}$. Secondly, instead of a *single* secret, we consider ICP that can deal with *multiple* secrets concurrently; later this allows to achieve better communication complexity, than executing multiple instances of ICP, dealing with a single secret. Our ICP is executed in the asynchronous setting and thus we call it AICP.

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

1. **Generation Phase**: it is initiated by Signer, having a secret input $\vec{S} = (s^1, \ldots, s^\ell) \in \mathbb{F}^\ell$, on which Signer wants to give its signature to INT. 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**: it is initiated by INT, who interacts with Signer and the verifiers to ensure that he holds a secret $\vec{S}$ and a corresponding authentication information, which will be later accepted by every (honest) verifier in $\mathcal{P}$ during the revelation phase. The secret $\vec{S}$ and the corresponding authentication information, which INT holds at the end of this phase[13] is called Signer's *IC signature* on $\vec{S}$, denoted as $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$.
3. **Revelation Phase**: it is carried out by INT and the verifiers in $\mathcal{P}$, where they interact among themselves. Here INT reveals $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$ and each verifier locally verifies $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$ with respect to its verification information and possibly let its 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 $\mathsf{Reveal}_i = \perp$

[13] This may be different from the secret and authentication information that INT received from Signer during the generation phase if Signer is *corrupted*; the details will be cleared during the description of our AICP.

(indicating that P_i is not convinced that INT obtained $\vec{S}$ from Signer). Accordingly, we say that 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), with Signer having a private input $\vec{S} \in \mathbb{F}^\ell$ for the protocol Gen, is called a $(1-\epsilon)$-AICP, for an allowed error parameter ϵ (where $\epsilon > 0$), if the following requirements[14] hold for every possible Adv:

1. **AICP-Correctness1:** If Signer and INT are *honest*, then each *honest* verifier $P_i \in \mathcal{P}$ outputs $\mathsf{Reveal}_i = \vec{S}$ at the end of RevealPublic.
2. **AICP-Correctness2:** If Signer is *corrupted* and INT is *honest*, holding $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$ at the end of Ver, then with probability at least $(1-\epsilon)$, all honest verifiers output $\mathsf{Reveal}_i = \vec{S}$ at the end of RevealPublic.
3. **AICP-Correctness3:** If Signer is *honest* and INT holds $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$ 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 $\vec{S}$. More specifically, let VIEW$(\vec{S})$ denote the random variable, describing the view of the adversary during an execution of Gen and Ver, where the secret input of Signer is $\vec{S}$. If Signer and INT are honest and INT has not executed RevealPublic, then VIEW$(\vec{S})$ is identically distributed for all $\vec{S} \in \mathbb{F}^\ell$.

2.5 Single and Multi-bit Common Coin

We now give the definition of common coin protocol, followed by the extension to multi-bit common coin protocol.

Definition 5 (Common Coin [7]) Let π be an asynchronous protocol, where each party in $\mathcal{P}$ has a random input and a binary output. We say that π is a $(1-\epsilon)$-*completing,* t-*resilient,* p-*common coin* protocol, for a given ϵ, where $\epsilon > 0$, if the following requirements hold for every possible input of the honest parties and every possible 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.

Definition 6 (Multi-Bit Common Coin) Let π be an asynchronous protocol, where each party in $\mathcal{P}$ has a random input and an output of ℓ bits. We say that π is a $(1-\epsilon)$-*completing,* t-*resilient,* p-*multi-bit common coin* protocol, with an output of ℓ bits, for a given error parameter ϵ, where $\epsilon > 0$, if the following requirements hold for every possible input of the honest parties and every possible 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 $l = 1, \ldots, \ell$, all the honest parties output σ_l as the lth bit with probability at least p for every $\sigma_l \in \{0,1\}$. [15]

2.6 Existing Tools

Asynchronous Broadcast: This primitive, called A-cast, was introduced and implemented by Bracha [6] with $3t+1$ parties. Formally, A-cast is defined as follows:

Definition 7 (A-cast [8]) Let π be an asynchronous protocol initiated by a Sender $\in \mathcal{P}$, 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 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 $m^\star = m$.

In Fig. 1, we recall the Bracha's A-cast protocol from [7]; the protocol incurs a private communication of $\mathcal{O}(\ell n^2)$ bits to broadcast an ℓ bit message [7].

In the rest of the paper, we use the following terminologies while using the A-cast protocol:

Terminology 1 (Terminologies for Using the A-cast Protocol). We say that:

1. "P_i *broadcasts* m" to mean that P_i acts as a Sender and invokes an instance of A-cast to broadcast m.
2. "P_j *receives* m *from the broadcast of* P_i" to mean that P_j (as a receiver) completes the execution of P_i's broadcast (namely the instance of the A-cast protocol where P_i is Sender), with m as the output.

Randomness Extraction [5,4]: In our common coin protocol, we use a well known method for "extracting" randomness with information-theoretic security. The setting is as follows: let $a_1, \ldots, a_N \in \mathbb{F}$, such that *at least* K out of these N values are selected uniformly at random from

[14] A closer look reveals that **AICP-Correctness2** and **AICP-Correctness3** are analogous to the non-repudiation and unforgeability property respectively of traditional digital signatures.

[15] Thus, the probability p is associated with each *individual* bit.

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.

Every $P_i \in \mathcal{P}$ (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.

$\mathbb{F}$; however, the exact identities of those K values are not known. The goal is to compute K values from $a_1, \ldots, a_N$, say $b_1, \ldots, b_K$, each of which is uniformly distributed over $\mathbb{F}$. This is achieved as follows: 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 $b_1 = f(N), \ldots, b_K = f(N + K - 1)$ (note that, we require $|\mathbb{F}| \geq N + K$ to make the technique work; in our protocols, N, K and $|\mathbb{F}|$ will be such that this relationship will be true). 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, which is a $(1 - \mu)$-AICP[16], where $\mu = \frac{n(\ell+t-1)}{|\mathbb{F}|-\ell}$ and ℓ is the number of secrets on which the signature is generated. Let the secret input of Signer be $\vec{S} = (s^1, \ldots, s^\ell) \in \mathbb{F}^\ell$. The underlying idea behind the protocol is as follows: During the generation phase, Signer selects a polynomial $F(x)$ of degree at most $\ell+t-1$, 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 publicly known distinct elements from $\mathbb{F}$. The polynomial $F(x)$ is given to INT. In addition, Signer gives the value of $F(x)$ at a random *evaluation-point* α_i (different from all β_j's) to each verifier P_i. During the revelation phase, INT 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.

An AICP with the aforementioned generation and revelation phase, and with a void verification phase, already 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 $F^\star(x)$, different from $F(x)$, and still remain unnoticed by an honest verifier P_i. The above AICP further satisfies the **AICP-Secrecy** property, as the degree of $F(x)$ is at most $\ell + t - 1$ and at most t points on $F(x)$ will be disclosed to Adv; 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 $\ell + t - 1$, there exists a polynomial that will be "consistent" with those t random points and *any* ℓ secrets. As we disclose below, the above AICP, however, does not achieve the **AICP-Correctness2**. Specifically if Signer is *corrupted*, then he might give $F(x)$ to INT, but evaluations of a different polynomial $\overline{F}(x)$ to each honest verifier, where $\overline{F}(x) \neq F(x)$. A verification phase, allowing interaction among Signer, INT and the verifiers is thus incorporated to bar Signer from doing the above. 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 $\ell + t - 1$. 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 are given in Fig. 2.

We now prove the properties of the protocol MultiVerifierAICP. We begin with the following two supporting claims.

Claim 1. *Let $F(x), R(x)$ be two polynomials of degree at most $\ell+t-1$ 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 $\frac{1}{|\mathbb{F}|-1}$, 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

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

further implying $(d - e)F(\alpha_i) = (d - e)v_i$ or $F(\alpha_i) = v_i$, which is a contradiction. As d is random, the condition $B(\alpha_i) = dv_i + r_i$ holds with probability at most $\frac{1}{|\mathbb{F}|-1}$. □

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.

[16] We use μ instead of ϵ to denote the error probability of our AICP, as we reserve ϵ to represent the error probability of our ABA protocol.

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

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

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

The following steps are executed only by Signer:

1. Select a random polynomial $F(x)$ over $\mathbb{F}$ of degree at most $\ell + t - 1$, such that $F(\beta_i) = s^i$, for $i = 1, \ldots, \ell$, where $\beta_1, \ldots, \beta_\ell$ are publicly known distinct elements from $\mathbb{F}$. In addition, select a random polynomial $R(x)$ over $\mathbb{F}$ of degree at most $\ell + t - 1$.
2. 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 $\alpha_i \in \mathbb{F} \setminus \{\beta_1, \ldots, \beta_\ell\}$. Send $F(x), R(x)$ to INT and (α_i, v_i, r_i) to the verifier P_i, where $v_i = F(\alpha_i)$ and $r_i = R(\alpha_i)$.

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

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 $dv_i + r_i = B(\alpha_i)$ for every verifier $P_i \in \mathcal{R}$, then Signer broadcasts the message `OK`.
   Otherwise Signer broadcasts the polynomial $F(x)$.
4. INT and the verifiers do the following, depending upon the broadcast of Signer:
   (a) If `OK` is received from the broadcast of Signer then:
      i. INT sets $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S}) = F(x)$, where $F(x)$ was received from Signer during Gen.
      ii. For $i = 1, \ldots, n$, verifier P_i sets (α_i, v_i) as its verification information, where α_i and v_i was received by P_i from Signer during Gen.
   (b) If a polynomial $\overline{F}(x)$ of degree at most $\ell + t - 1$ is received from the broadcast of Signer, then:
      i. INT sets $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S}) = \overline{F}(x)$.
      ii. For $i = 1, \ldots, n$, verifier P_i computes $v_i = \overline{F}(\alpha_i)$ and sets (α_i, v_i) as its verification information.

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

1. INT broadcasts $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$.
2. For $i = 1, \ldots, n$, verifier P_i does the following:
   (a) Wait to receive $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$ from the broadcast of INT. Upon receiving, interpret $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$ as a polynomial $F^\star(x)$ of degree at most $\ell + t - 1$.
   (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 $\mathsf{Reveal}_i = \vec{S^\star}$, 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 $\mathsf{Reveal}_i = \perp$.

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 broadcast the polynomial $B(x)$ during Ver, where $B(x) \stackrel{def}{=} dF(x) + R(x)$ and so $dv_i + r_i = B(\alpha_i)$ will hold for every $P_i \in \mathcal{R}$. □

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* $\mathsf{Reveal}_i = \vec{S^\star} = \vec{S}$.

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 broadcast $\mathsf{ICSig} = F(x)$ (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* Signer *is corrupted and* INT *is honest, holding* $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$ *at the end of* Ver*, then except with probability at most* $\frac{n}{|\mathbb{F}|-1}$*, all honest verifiers will output* $\mathsf{Reveal}_i = \vec{S}$ *at the end of* RevealPublic.

PROOF: We first claim that except with probability at most $\frac{1}{|\mathbb{F}|-1}$, an *honest* verifier $P_i \in \mathcal{R}$ will broadcast the `Accept` message during RevealPublic, in response to $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$ broadcasted by the honest INT; so except with probability at most $|\mathcal{R}| \cdot \frac{1}{|\mathbb{F}|-1} \leq \frac{n}{|\mathbb{F}|-1}$, all the honest ver-

ifiers in $\mathcal{R}$ will broadcast Accept. 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 now prove our claim by considering 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 claim is true, as INT will set $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S}) = \overline{F}(x)$ as the IC signature and each *honest* verifier $P_i \in \mathcal{R}$ will set $v_i = \overline{F}(\alpha_i)$ as its verification information at the end of Ver. During RevealPublic, the honest INT will broadcast $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S}) = F^\star(x) = \overline{F}(x)$ 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 $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S}) = F^\star(x) = F(x)$ during RevealPublic. Now we have the following cases depending on the relationship between $(F(x), R(x))$ held by INT and the tuple (α_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 Accept, 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 Accept, except with probability at most $\frac{1}{|\mathbb{F}|-1}$, 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 $R(\alpha_i) \neq r_i$, then there exists a unique, non-zero 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 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 *holds* $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$ *at the end of* Ver*, then the probability that an honest* P_i *outputs* $\mathsf{Reveal}_i = \vec{S^\star}$ *at the end of* RevealPublic*, where* $\vec{S^\star} \neq \vec{S}$ *is at most* $\frac{n(\ell+t-1)}{|\mathbb{F}|-\ell}$.

PROOF: First of all we have to consider a *corrupted* INT, as an honest INT always reveals $\mathsf{ICSig}(\mathsf{Signer}, \mathsf{INT}, \mathcal{P}, \vec{S})$. To make an honest verifier $P_i \in \mathcal{P}$ output $\mathsf{Reveal}_i = \vec{S^\star}$ 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, that evaluates to the elements of $\vec{S^\star}$ at $x = \beta_1, \ldots, \beta_\ell$. We now claim that if INT does so, then except with probability at most $\frac{\ell+t-1}{|\mathbb{F}|-\ell}$, an *honest* verifier $P_i \in \mathcal{R}$ will broadcast the Reject message during RevealPublic; so except with probability at most $|\mathcal{R}| \cdot \frac{\ell+t-1}{|\mathbb{F}|-\ell} \leq \frac{n(\ell+t-1)}{|\mathbb{F}|-\ell}$, all the honest verifiers in $\mathcal{R}$ will broadcast Reject. 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), this implies that no honest verifier in the set $\mathcal{P}$ will output $\vec{S^\star}$. We now prove our claim by considering the following two cases, based on the broadcast of the honest Signer 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. So clearly the condition *C2* will not hold for any honest verifier $P_i \in \mathcal{R}$ during RevealPublic. An honest verifier $P_i \in \mathcal{R}$ can therefore broadcast the Accept message under the condition *C1*, namely when $F^\star(\alpha_i) = v_i$ holds. 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 that $F^\star(\alpha_i) = v_i = F(\alpha_i)$ holds is the same as INT can correctly guess α_i, which is at most $\frac{\ell+t-1}{|\mathbb{F}|-\ell}$ (since $F^\star(x)$ and $F(x)$ can have the same value for at most $\ell + t - 1$ 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 $P_i \in \mathcal{R}$. 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 $\frac{\ell+t-1}{|\mathbb{F}|-\ell}$:
   (a) $F^\star(\alpha_i) = v_i = F(\alpha_i)$: As discussed above, this can happen with probability at most $\frac{\ell+t-1}{|\mathbb{F}|-\ell}$.
   (b) $B(\alpha_i) \neq dv_i + r_i$ *and* Signer *broadcasted the* OK *message during* Ver: This is impossible because if $B(\alpha_i) \neq dv_i + r_i$, then an honest Signer would have broadcasted $F(x)$, instead of OK during Ver. □

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* $\vec{S} = (s^{(1)}, \ldots, s^{(\ell)})$.

PROOF: If Signer and INT are honest, then Signer will broadcast the OK message during Ver. Without loss of generality, let the verifiers $P_1, \ldots, P_t \in \mathcal{P}$ be under the control of Adv. 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 $i = 1, \ldots, t$. Further, the adversary Adv can compute $B(\beta_1), \ldots, B(\beta_\ell)$, from which it 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 $\ell + t - 1$ and the two polynomials are independent of each other. It is easy to see that $d, F(\alpha_1), \ldots, F(\alpha_t), R(\alpha_1), \ldots R(\alpha_t), dF(\beta_1)$

$+ R(\beta_1), \ldots, dF(\beta_\ell) + R(\beta_\ell)$ have 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 $s^1, \ldots, s^\ell$, there exists polynomials $\overline{R}(x)$ and $\overline{F}(x)$ of degree $\ell + t - 1$ with $r_1 = \overline{R}(\alpha_1), \ldots, r_t = \overline{R}(\alpha_t)$, $v_1 = \overline{F}(\alpha_1), \ldots, v_t = \overline{F}(\alpha_t)$ and $\overline{F}(\beta_1) = s^1, \ldots, \overline{F}(\beta_\ell) = s^\ell$, such that $dv_1 + r_1 = d\overline{F}(\beta_1) + \overline{R}(\beta_1), \ldots, dv_\ell + r_\ell = d\overline{F}(\beta_\ell) + \overline{R}(\beta_\ell)$ will hold. □

Theorem 1 *Protocols* $(\mathsf{Gen}, \mathsf{Ver}, \mathsf{RevealPublic})$ *constitute a* $(1-\mu)$*-AICP, where* $\mu = \frac{n(\ell+t-1)}{|\mathbb{F}|-\ell}$. *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: During 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 causing 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 and separately Signer broadcast a polynomial represented by $\ell + t$ field elements, thus incurring a broadcast of $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ bits. In the protocol RevealPublic, INT broadcasts ICSig that consists of $\ell + t$ field elements. Each verifier in $\mathcal{R}$ broadcasts `Accept`/`Reject` message. So RevealPublic involves broadcast of $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ bits. The proof now follows from Lemmas 1-4 and the fact that $\frac{n}{|\mathbb{F}|-1} < \frac{n(\ell+t-1)}{|\mathbb{F}|-\ell}$. □

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

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

1. "P_i *gives* $\mathsf{ICSig}(P_i, P_j, \mathcal{P}, \vec{S})$ *to* P_j" to mean that P_i as a Signer executes the protocol $\mathsf{Gen}(P_i, P_j, \mathcal{P}, \vec{S})$, considering P_j as an INT, to give its IC signature on $\vec{S}$.
2. "P_i *receives* $\mathsf{ICSig}(P_j, P_i, \mathcal{P}, \vec{S})$ *from* P_j" to mean that P_i as an INT has completed the protocol $\mathsf{Ver}(P_j, P_i, \mathcal{P}, \vec{S})$ and holds $\mathsf{ICSig}(P_j, P_i, \mathcal{P}, \vec{S})$, where P_j is Signer.
3. "P_i *reveals* $\mathsf{ICSig}(P_j, P_i, \mathcal{P}, \vec{S})$" to mean that P_i as an INT executes $\mathsf{RevealPublic}(P_j, P_i, \mathcal{P}, \vec{S})$ for revealing $\mathsf{ICSig}(P_j, P_i, \mathcal{P}, \vec{S})$ and hence $\vec{S}$, where P_j is Signer, while the verifiers in $\mathcal{P}$ participate in the instance of RevealPublic.
4. "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 RevealPublic $(P_j, P_i, \mathcal{P}, \vec{S})$, where P_j is Signer and P_i is INT, with output $\mathsf{Reveal}_k = \vec{S^\star}$ (resp. $\mathsf{Reveal}_k = \perp$).
5. "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 $\mathsf{RevealPublic}(P_j, P_i, \mathcal{P}, \vec{S})$, where P_i is INT and P_j is Signer.
6. "P_i *failed to reveal* $\mathsf{ICSig}(P_j, P_i, \mathcal{P}, \vec{S})$ (and hence $\vec{S}$)" to mean that each honest verifier P_k outputs $\mathsf{Reveal}_k = \perp$ after completing $\mathsf{RevealPublic}(P_j, P_i, \mathcal{P}, \vec{S})$, where P_i is INT and P_j is Signer.

4 Asynchronous Weak Commitment (AWC)

We present an AWC scheme, which is a $(1-\delta)$-AWC, where $\delta = \frac{n^2(\ell+t-1)}{|\mathbb{F}|-\ell}$ and ℓ is the number of secrets committed in the scheme. For the ease of presentation, we first present an AWC scheme which allows to commit and decommit a single secret (i.e. $\ell = 1$). This is followed by the modifications required to deal with ℓ secrets concurrently. Next, we compare our AWC scheme with the existing AWSS and W-SVSS schemes. We conclude with an important interpretation of our AWC on committing and decommitting polynomials (instead of committing and decommitting a secret as it is projected now). The interpretation plays a crucial role when AWC is plugged in the AVSS scheme.

4.1 AWC Scheme for a Single Secret

We present an AWC scheme called AWC-Single, consisting of a pair of protocols $(\mathsf{Com}, \mathsf{Decom})$, which allows a Committer $\in \mathcal{P}$ to commit 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 [27] 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 of s, which are nothing but n distinct evaluations of the polynomial. Then Committer sends the ith share Sh_i to the party P_i. On receiving the share Sh_i from Committer, party P_i "acknowledges" by signing Sh_i and giving $\mathsf{ICSig}(P_i, \mathsf{Committer}, \mathcal{P}, Sh_i)$ to Committer.

To avoid endless waiting (due to asynchronicity), on receiving the signatures from $n-t = 2t+1$ parties, Committer broadcasts the identities of these $2t+1$ parties (we denote these parties by the set WCORE) and every (honest) party terminates Com on receiving WCORE from Committer. At this stage, Committer has committed a secret determined by the shares of the (honest) parties in WCORE. Specifically, let $\overline{f}(x)$ be the polynomial, defined by the shares[17] of the

[17] A set of shares $\mathcal{S} = \{(i, Sh_i)\}$ is said to *define a polynomial*, say $p(\cdot)$, if $p(i) = Sh_i$ holds for every index i in the set $\mathcal{S}$; we also say that

honest parties in WCORE. Then, the committed secret, say $\overline{s}$, is the constant term of $\overline{f}(x)$ (thus belongs to $\mathbb{F}$) if the polynomial is of degree at most t, else $\overline{s} = \perp$.

If Committer is *honest* then the protocol Com preserves the privacy of the secret s; this follows from the privacy of Shamir secret-sharing and the secrecy property of the AICP. Furthermore, protocol Com terminates since every honest party in the set of at least $2t + 1$ honest parties will be eventually included in WCORE. The committed secret $\overline{s}$ is the same as the secret input s of the honest Committer. Now consider the case when Committer is *corrupted*. In that case, it 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 $\overline{s}$ may be $\perp$. However, as discussed in the sequel, during the protocol Decom, a corrupted Committer cannot decommit any $s^\star$ different from the committed secret $\perp$.

In the protocol Decom, Committer reveals the signatures of *all* the $2t + 1$ parties in the set WCORE on the corresponding shares. So the participation of Committer is very crucial in the protocol Decom, as otherwise, the honest parties will never participate and terminate the protocol Decom. A corrupted Committer might thus choose to let Decom never complete. 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 $\perp$. The interpretation is that Committer wants to decommit the secret, say $s^\star$, which is the constant term of the polynomial, defined by the $2t + 1$ shares (corresponding to the parties in WCORE), that are revealed by Committer (along with the corresponding signatures). If these shares define a polynomial of degree at most t, then $s^\star$ is the constant term of the polynomial (thus belongs to $\mathbb{F}$); otherwise $s^\star = \perp$.

It is easy to see that an *honest* Committer will successfully reveal the required signatures, leading to a decommitment of s. On the other hand, a *corrupted* Committer can *only* successfully reveal the signature on the *same* share Sh_i, which it had given to an *honest* party $P_i \in$ WCORE during Com (follows from the **AICP-Correctness3** property of AICP). This implies that if $s^\star \in \mathbb{F}$ is the decommitted secret then with high probability, $s^\star$ is the same as secret $\overline{s}$, committed during Com. That is, the constant term of the polynomial defined by the shares of the honest parties in WCORE (namely $\overline{s}$) in Com is $s^\star$.[18]

Without violating the properties of the protocol, we add few *additional* steps in the protocol Com. These additional steps do not play any role for the AWC scheme, but they are crucial to our AVSS scheme (described in the next section) that is built on AWC scheme. Firstly, while distributing Sh_i to party P_i, the Committer signs the share and gives $\mathsf{ICSig}(\mathsf{Committer}, P_i, \mathcal{P}, Sh_i)$ to P_i. To distinguish these signatures from the ones described before, we use the following notations: **(1)** The signatures $\mathsf{ICSig}(P_i, \mathsf{Committer}, \mathcal{P}, Sh_i)$, given by the share-holders P_i to Committer are called *primary signatures*. **(2)** The signatures $\mathsf{ICSig}(\mathsf{Committer}, P_i, \mathcal{P}, Sh_i)$, given by Committer to the share-holders P_i are called *secondary signatures*. Secondly, $(\texttt{Sign-Sent}, \star, \star)$ and $(\texttt{Sign-Received}, \star, \star)$ messages are broadcasted by the share-holders and Committer respectively, after giving and receiving the primary signatures. The formal details of AWC-Single are given in Fig 3.

We now prove the properties of AWC-Single.

Lemma 5 (Termination) *Protocols* $(\mathsf{Com}, \mathsf{Decom})$ *satisfy the termination condition of Definition 3 without any error.*

PROOF: If Committer is *honest* then eventually it will receive $\mathsf{ICSig}(P_i, \mathsf{Committer}, \mathcal{P}, Sh_i')$ from every honest P_i, where $Sh_i' = Sh_i$ and there are at least $2t + 1$ such honest parties. So eventually, Committer will broadcast a set WCORE and the corresponding $(\texttt{Sign-Received}, \star, \star)$ messages and by the properties of broadcast, every honest party will eventually receive the set and the messages. Moreover, the honest Committer also must have received the message $(\texttt{Sign-Sent}, i, \mathsf{Committer})$ 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 $2t + 1$ and the required $(\texttt{Sign-Received}, \star, \star)$ messages from the broadcast of Committer, along with the message $(\texttt{Sign-Sent}, j, \mathsf{Committer})$ 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.

Now we show that if Committer is honest then every party will terminate Decom. The claim is trivially true since an honest Committer will invoke the protocol Decom and reveal the signature $\mathsf{ICSig}(P_j, \mathsf{Committer}, \mathcal{P}, Sh_j)$ corresponding to each $P_j \in$ WCORE. Irrespective of whether Committer has revealed the signatures successfully or not, every party will terminate the protocol eventually.

It is left to show that if Committer is corrupted and some honest party, say P_i, terminates Decom, then every other

the shares in $\mathcal{S}$ *lie* on the polynomial $p(\cdot)$. Note that the degree of $p(\cdot)$ will be at most $|\mathcal{S}| - 1$.

[18] Note that there are at least $t+1$ honest parties in WCORE and thus the shares of the honest parties in WCORE uniquely define a single polynomial of degree at most t.

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

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

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

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 ICSig(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 ICSig(P_i, 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 dynamic set WCORE, which is initialized to $\emptyset$. In addition, broadcast the message (Sign-Received, i, Committer) to publicly notify the receipt of signature from the party P_i.
4. Wait till $|\text{WCORE}| = 2t + 1$ and then broadcast the set WCORE.

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 ICSig(Committer, $P_i, \mathcal{P}, Sh_i$) from Committer, act as a Signer and give ICSig(P_i, 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, along with the messages (Sign-Received, j, Committer), corresponding to each $P_j \in$ WCORE, from the broadcast of Committer. On receiving WCORE and these messages, 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.

Decom(Committer, $\mathcal{P}$, s)

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

1. Act as an INT and reveal ICSig(P_j, 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 ICSig(P_j, 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 ICSig(P_j, Committer, $\mathcal{P}, Sh_j$) with output $\text{Reveal}_i = \perp$ then output $\perp$ and terminate.
3. If P_i completes the revelation of ICSig(P_j, Committer, $\mathcal{P}, Sh_j$) with output $\text{Reveal}_i = Sh_j$ for every $P_j \in$ WCORE, then do the following:
   (a) If the points $\{(j, Sh_j) : P_j \in \text{WCORE}\}$ 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 $\{(j, Sh_j) : P_j \in \text{WCORE}\}$ do not lie on a unique polynomial of degree at most t, then output $\perp$ and terminate.

honest party eventually terminates the protocol. From the protocol steps, it follows that P_i must have completed the revelation of ICSig(P_j, Committer, $\mathcal{P}, Sh_j$), corresponding to every $P_j \in$ WCORE with some output Reveal_i. From the protocol steps of RevealPublic, every other honest party will also eventually complete the revelation of these ICSigs and hence will terminate the protocol Decom. This concludes the proof of termination. □

Lemma 6 (Correctness) *Protocols* (Com, Decom) *satisfy the correctness condition of Definition 3 with probability at least* $(1 - \delta)$*, where* $\delta = \frac{n^2 t}{|\mathbb{F}|-1}$.

PROOF: First we show that if some honest party terminates Com, then there exists a *committed* secret $\overline{s} \in \mathbb{F} \cup \{\perp\}$. The secret $\overline{s}$ is defined 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 and the condition that some honest party terminated Com ensures that every honest party in WCORE has received its share from Committer. We next define $s^\star$, the input for Committer during the protocol Decom (namely the value for decommitting): $s^\star$ is the constant term of the polynomial, defined by the $2t+1$ shares, which are revealed by Committer during the protocol Decom; if the polynomial has degree at most t, then $s^\star \in \mathbb{F}$, otherwise $s^\star = \perp$. Now we consider the following two cases (depending upon the behaviour of Committer):

- Committer *is honest*: In this case, $s^\star = \overline{s} = s$. This is because, the $2t + 1$ shares which Committer reveals during Decom are the points on the original polynomial $f(x)$, selected by Committer during Com. In this case, Committer will successfully reveal the signature corresponding to every party in WCORE, except with probability at most $t\mu \leq \delta$, where $\mu = \frac{nt}{|\mathbb{F}|-1}$ (for every *honest* party in WCORE, the claim is true without any error due to **AICP-Correctness1**; for every *corrupted* party in WCORE, the claim is true due to **AICP-Correctness2**, except with probability μ; the rest follows from the fact

that there can be at most t corrupted parties in WCORE). So except with probability at most δ, all the honest parties will output s upon terminating Decom. This proves the first requirement.

– Committer *is corrupted*: we show that if Committer tries to decommit $s^\star$, where $s^\star \neq \overline{s}$, then except with probability at most δ, all the honest parties will output $\perp$ upon terminating Decom. To decommit $s^\star \neq \overline{s}$, it must be the case that during Decom, Committer reveals Sh'_j as the share on the behalf of at least one *honest* party $P_j \in$ WCORE, such that Sh'_j is different from the share Sh_j, which Committer has given to P_j during the Com protocol. However, if Committer does so, then except with probability at most μ, Committer will fail to reveal $\mathsf{ICSig}(P_j, \mathsf{Committer}, \mathcal{P}, Sh'_j)$ due to **AICP-Correctness3**. Now there are at least $t+1$ honest parties in WCORE and so except with probability at most $|\mathsf{WCORE}| \cdot \mu \leq \delta$, Committer will fail to successfully reveal the IC signature on any incorrect share on the behalf of any honest party in WCORE. This proves the second 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 Shamir secret-sharing and the **AICP-Secrecy**. More specifically, Adv gets to know t points (namely the shares) on the random polynomial $f(x)$ of degree at most t from the corrupted parties. From the view-point of Adv, for every possible secret $\overline{s}$, there exists a unique polynomial of degree at most t which is consistent with the t shares that Adv knows and the secret $\overline{s}$. So all possible secrets are equi-probable from Adv's point of view. Moreover, Adv does not get any extra information about the shares of the honest parties from the corresponding instances of AICP. That is, corresponding to every honest P_i, the adversary obtains no information about the share $Sh_i = f(i)$ during the instances $\mathsf{Gen}(\mathsf{Committer}, P_i, \mathcal{P}, Sh_i), \mathsf{Ver}(\mathsf{Committer}, P_i, \mathcal{P}, Sh_i), \mathsf{Gen}(P_i, \mathsf{Committer}, \mathcal{P}, Sh_i)$ and $\mathsf{Ver}(P_i, \mathsf{Committer}, \mathcal{P}, Sh_i)$, which are invoked in Com to generate Committer's and P_i's IC signature on Sh_i (this follows from Lemma 4). □

Theorem 2 *Protocols* $(\mathsf{Com}, \mathsf{Decom})$ *is a* $(1-\delta)$*-AWC scheme for a single secret, where* $\delta = \frac{n^2 t}{|\mathbb{F}|-1}$. *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, $2n$ 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 1 by substituting $\ell = 1$ and from Lemmas 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 let the secret be a vector $\vec{S} = (s^1, \ldots, s^\ell)$, consisting of ℓ elements from $\mathbb{F}$, where $\ell > 1$. We can execute one "dedicated" instance of the Com and Decom protocol for 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 when AWC is plugged in our AVSS scheme and the common coin protocol) then the broadcast communication of our protocol will be $\mathcal{O}(\ell n \log \frac{1}{\epsilon})$ bits, instead of $\mathcal{O}(\ell n^2 \log \frac{1}{\epsilon})$ bits. This is 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 very expensive[19].

We extend the protocol Com and Decom in a "natural" way to deal with ℓ values concurrently. Firstly, Committer computes ℓ Shamir-sharings, one corresponding to each $s^l \in \vec{S}$. Secondly the instances of Gen, Ver and RevealPublic in the Com and Decom protocol are invoked to deal with ℓ values concurrently, instead of a single value. The modified protocols are presented in Fig 4. The new scheme is called AWC-Multiple, as it deals with multiple secrets. The properties of AWC-Multiple follow using the same arguments as for the earlier scheme: more specifically, the argument for termination is exactly the same as in Lemma 5. The correctness property follows similarly. Specifically, the vector of secret committed by Committer is defined by the vector of Shamir-shares of the honest parties in WCORE, while the vector of secret decommitted by Committer is defined by the vector of Shamir-shares corresponding to the parties in WCORE revealed by Committer (along with the signatures). Finally the secrecy is argued as follows. We observe that each element of $\vec{S}$ is independently Shamir-shared with threshold t; moreover the vector of Shamir-shares of the honest parties remain private during the generation of primary and secondary signatures on them, thanks to the secrecy property of AICP. We state the following theorem; the communication complexity of AWC-Multiple follows from the properties of the protocol and the communication complexity of our AICP (Theorem 1).

Theorem 3 *Protocols* $(\mathsf{Com}, \mathsf{Decom})$ *is a* $(1-\delta)$*-AWC scheme for ℓ secrets, where* $\delta = \frac{n^2(\ell+t-1)}{|\mathbb{F}|-\ell}$. *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.*

[19] 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 $n = 3t + 1$.

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

Com(Committer, $\mathcal{P}, \vec{S}$)

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 $S\vec{h}_i = (f_1(i), \ldots, f_\ell(i))$.
3. For $i = 1, \ldots, n$, give ICSig(Committer, $P_i, \mathcal{P}, S\vec{h}_i$) to P_i by acting as a Signer and considering P_i as an INT.
4. For $i = 1, \ldots, n$, if ICSig(P_i, Committer, $\mathcal{P}, S\vec{h}'_i$) is received from P_i, such that $S\vec{h}'_i = S\vec{h}_i$, 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 initialized to $\emptyset$. In addition, broadcast the message (`Sign-Received`, i, Committer) to publicly notify the receipt of signature from the party P_i.
5. Wait till $|\text{WCORE}| = 2t + 1$ and then broadcast the set WCORE.

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 ICSig(Committer, $P_i, \mathcal{P}, S\vec{h}_i$) from Committer, act as a Signer and give ICSig(P_i, Committer, $\mathcal{P}, S\vec{h}_i$) 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 $2t + 1$, along with the messages (`Sign-Received`, j, Committer), corresponding to each $P_j \in$ WCORE, from the broadcast of Committer. On receiving WCORE and the messages, 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.

Decom(Committer, $\mathcal{P}, \vec{S}$)

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

1. Act as an INT and reveal ICSig(P_j, Committer, $\mathcal{P}, S\vec{h}_j$), 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 ICSig(P_j, Committer, $\mathcal{P}, S\vec{h}_j$), corresponding to each $P_j \in$ WCORE.
2. If $\exists P_j \in$ WCORE, such that P_i completes the revelation of ICSig(P_j, Committer, $\mathcal{P}, S\vec{h}_j$) with output $\text{Reveal}_i = \perp$ then output $\perp$ and terminate.
3. If P_i completes the revelation of ICSig(P_j, Committer, $\mathcal{P}, S\vec{h}_j$) with output $\text{Reveal}_i = S\vec{h}_j$ for every $P_j \in$ WCORE, then interpret $S\vec{h}_j$ as $(f_1(j), \ldots, f_\ell(j))$ and do the following:
   (a) If for $l = 1, \ldots, \ell$, the points $\{(j, f_l(j)) : P_j \in \text{WCORE}\}$ 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 $\{(j, f_l(j)) : P_j \in \text{WCORE}\}$ do not lie on a unique polynomial of degree at most t, then output $\perp$ and terminate.

4.3 Comparison of Our AWC with the AWSS of [22] and W-SVSS of [1]

The best known AWSS scheme was presented in [22]. The scheme is a $(1-\epsilon)$-AWSS scheme and is based on the idea of using a bivariate polynomial to share a secret. Specifically, to share a secret s, a random bivariate 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 secret, only n elements from $\mathbb{F}$ need to be distributed, as the secret is shared using a univariate polynomial. This clearly 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 based on the idea of using a bivariate 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; else the protocol ensures that there exists at least one *honest* party, who will shun (ignore communication from) at least one *corrupted* party from then onwards for the rest of the protocol execution. Property wise, W-SVSS is incomparable to AWC.

4.4 AWC for Sharing Polynomials

An interesting interpretation of the computation done in the protocol AWC-Single and AWC-Multiple is presented be-

low; this interpretation is very crucial for understanding how the AWC is used later in our AVSS scheme. For simplicity, we present the discussion with respect to AWC-Single (Fig 3); the discussion for the protocol AWC-Multiple (Fig 4) will follow similarly. 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); furthermore the view of the adversary will be independent of $f(0)$ (see the proof of Lemma 7). We can recast the entire computation (during the protocol Com) in terms of Committer committing a polynomial $f(x)$ instead of a secret s. Essentially, we now consider $f(x)$ instead of s as the input of Committer, while rest of the protocol steps remain the same. Similarly, we can recast the computation in Decom in terms of Committer decommitting a polynomial. Namely, if the polynomial $f^\star(x)$ reconstructed in Decom has degree at most t, then all the (honest) parties output this polynomial; moreover the same polynomial was committed during Com. 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 (decommitting) a polynomial $f(x)$ of degree at most t" is not new and it is commonly used in the literature of WSS and VSS (for example, see [21, 16, 19]).

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 terminologies:

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

1. "Committer *commits* $f_1(x), \ldots, f_\ell(x)$" to mean that Committer invokes Com(Committer, $\mathcal{P}$, ($f_1(0), \ldots, f_\ell(0)$)), where Committer uses $f_1(x), \ldots, f_\ell(x)$ for generating the Shamir-shares of $s^1, \ldots, s^\ell$ respectively 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 has the secondary signature ICSig(Committer, $P_j, \mathcal{P}, \vec{Sh}_j$) and Committer has the primary signature ICSig(P_j, Committer, $\mathcal{P}, \vec{Sh}_j$), corresponding to every $P_j \in$ WCORE; here $\vec{Sh}_j = (f_1(j), \ldots, f_\ell(j))$ is called the j^{th} *share* of the committed polynomials $f_1(x), \ldots, f_\ell(x)$.
2. "Committer *decommits* $f_1(x), \ldots, f_\ell(x)$" to mean that Committer invokes Decom(Committer, $\mathcal{P}$, ($f_1(0), \ldots, f_\ell(0)$)), where during the step DECOMMITTING THE SECRET in AWC-Multiple, Committer reveals the primary signatures ICSig(P_j, Committer, $\mathcal{P}, \vec{Sh}_j$), corresponding to each $P_j \in$ WCORE, where $\vec{Sh}_j = (f_1(j), \ldots, f_\ell(j))$. If the honest parties terminate the protocol, then they either output $f_1(x), \ldots, f_\ell(x)$, if these polynomials are of degree at most t and the same polynomials were committed earlier by Committer, during the Com protocol; otherwise the parties output $\perp$.

5 Asynchronous Verifiable Secret Sharing (AVSS)

We present a $(1-\gamma)$-AVSS scheme, where $\gamma = \frac{n^3(\ell+t-1)}{|\mathbb{F}|-\ell}$ and ℓ is the number of secrets shared in the scheme. For the ease of presentation, we first present an AVSS scheme for sharing a single secret. Later we will brief the modifications needed for the multi-secret version.

5.1 AVSS for Sharing a Single Secret

Our AWC schemes (AWC-Single and AWC-Multiple) readily give "honest dealer" AVSS schemes where Committer takes the role of the dealer D. Saying differently, they offer all the properties an AVSS scheme provides when D is honest. On the contrary, our AWC schemes are not AVSS schemes when a corrupted Committer takes the role of the dealer. There are two reasons for that: the third requirement of the termination condition (that informally says that the termination of the reconstruction protocol is not controlled by the corrupted dealer) and the second requirement of the correctness condition (that informally says that nothing but the committed secret is reconstructed) of AVSS are violated.

In this section, we build our AVSS based on the idea of sharing the secret using a bivariate polynomial of degree t in each variable. The idea of bivariate-polynomial based secret sharing is not new and has been widely used in the literature of VSS in the synchronous setting (for example, see [21, 19, 11, 16, 17] and their references). The same idea was also used in [1] to design a shunning-VSS (SVSS). Our contribution for AVSS is the way we plug in our AWC scheme in the AVSS scheme. Thus far, almost all the existing VSS protocols are in general constructed from AWSS (AWC has been differentiated from AWSS in section 2.3.3 where we argued that AWC has weaker requirements than AWSS and hence can be designed more efficiently). In what follows, we

provide a brief background about symmetric bivariate polynomials, which are used in our protocol.

Bivariate Polynomials: A *symmetric bivariate 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:

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

which implies that $r_{ij} = r_{ji}$, for $i, j = 1, \ldots, t$. Notice that $F(x, y)$ has $(t + 1) + t + \ldots + 1 = \frac{(t+1)(t+2)}{2}$ distinct 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 bivariate 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* $f_i(x)$ polynomials, one can efficiently compute $F(x, y)$, as the knowledge of $t + 1$ such distinct polynomials provides $\frac{(t+1)(t+2)}{2}$ distinct points on $F(x, y)$, which are sufficient to interpolate $F(x, y)$.

Looking ahead, the following important property of the bivariate polynomials will be used to prove the secrecy of our AVSS schemes: let $s \in \mathbb{F}$ be the secret and $F(x, y)$ be a random, symmetric bivariate polynomial of degree t, subject to the condition that $s = F(0, 0)$. Then given only t distinct polynomials $f_i(x)$, where $f_i(x) = F(x, i)$ and $i \in \{1, \ldots, n\}$, no information is revealed about s in the information-theoretic sense; intuitively this is because the knowledge of t such distinct $f_i(x)$ polynomials 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.

Informal Description of Our AVSS: First, consider a simple scheme as follows: A dealer D, on having an input secret s, selects a random, symmetric bivariate 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 and we call $f_i(x)$ as the *share* of s for the party P_i. The distribution of information as above does not violate the secrecy of s for an *honest* D. Next, assume that the parties can agree on a *common* "defining" set ShVCORE of at least $n - t = 2t + 1$ parties, who have received their shares from D. Based on the shares received by the parties in ShVCORE, we define the committed secret $\overline{s}$ as follows: if there exists a unique, symmetric bivariate polynomial of degree t, say $\overline{F}(x, y)$, such that for every *honest* $P_j \in$ ShVCORE, it holds that $f_j(x) = \overline{F}(x, j)$ (we will often say that the share $f_j(x)$ *lies* on $\overline{F}(x, y)$ if this condition is satisfied), where $f_j(x)$ is the share of P_j, then we say that D has committed $\overline{s} = \overline{F}(0, 0)$ during the sharing protocol; otherwise we say that D has committed $\overline{s} = \perp$.[20] Now further assume that the share $f_j(x)$ of each $P_j \in$ ShVCORE is "shared" among the parties in such a way that the following requirements are met:

(R1) If P_j is *honest*, then $f_j(x)$ can be reconstructed back *robustly*;
(R2) If P_j is *corrupted* and $\overline{s} \neq \perp$, then either the correct share $f_j(x)$ or $\perp$ (and nothing else) can be reconstructed back, even *without* any help from P_j[21].

The above described scheme readily gives an AVSS. Namely if $\overline{s} \neq \perp$, then the shares of all *honest* parties in ShVCORE can be reconstructed robustly; moreover corresponding to the corrupted parties in ShVCORE, either the correct share or $\perp$ can be reconstructed. All together, these reconstructed shares will give $\overline{s}$. For $\overline{s} = \perp$, irrespective of what is reconstructed for the corrupted parties $P_j \in$ ShVCORE, the shares of the *honest* parties in ShVCORE can be reconstructed robustly, which along with the other reconstructed shares will output $\perp$. We next discuss how to find an ShVCORE and how to make an agreement on it among the parties.

We employ the following idea to find an ShVCORE: each party P_j on receiving its share $f_j(x)$ from D acts as a Committer and commits $f_j(x)$ by invoking an instance of Com (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_j as Com_j (resp. Decom_j) and the instance of WCORE constructed during Com_j as WCORE_j. A *corrupted* P_j is prevented from committing an incorrect share $f'_j(x)$ different from $f_j(x)$ in the instance Com_j via a trick supported by the "symmetric" property of the bivariate polynomials. Specifically, let $f_j(i)$ be the *AWC-Share* (of the polynomial $f_j(x)$) for the party P_i that it is supposed to receive from the Committer P_j in the instance Com_j; ideally for an *honest* P_j (and D), the condition $f_j(i) \stackrel{?}{=} f_i(j)$ should be true where $f_i(x)$ denotes the share of s (as received by P_i from D). So we enforce that a party P_i participates in the instance Com_j *only* if its AWC-share is "consistent" with the share received from D, namely $f_j(i) \stackrel{?}{=} f_i(j)$ should hold. We refer this checking as *pre-verification* and stress that the purpose of executing Com_j (coupled with this pre-verification) is not

[20] 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, bivariate polynomial of degree t. This is analogous to AWC, where the shares of the parties in WCORE defined the committed univariate polynomial.

[21] If $\overline{s} = \perp$, then we do not bother what is reconstructed back for a corrupted $P_j \in$ ShVCORE.

to provide any "new" information about $f_j(x)$ to the parties, but rather to prevent a corrupted P_j from committing an incorrect share. Once the parties commit their received shares, the parties then try to find a common set of at least $2t+1$ committers ShVCORE, such that each committer $P_j \in$ ShVCORE has committed its share to at least $2t+1$ parties within ShVCORE; i.e. the condition $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t+1$ holds for each committer $P_j \in$ ShVCORE. Looking ahead, we note that the condition $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t+1$ plays a key role to ensure (R1) and (R2) as desired. Now, notice that if D is *honest* then ShVCORE with the above properties is sure to exist. An immediate possibility is the set of all honest parties. The questions that remain to be settled are: **(1).** How to find out such a set of committers and how the parties agree on such a common set, if it exists? **(2).** How (R1) and (R2) can be met? We next discuss how these issues are addressed.

We put on D the job of finding ShVCORE and making the parties agree on the same. Specifically D keeps a "track" of all such P_j whose instance of Com_j is (locally) terminated by D and as soon as D finds $2t+1$ such P_j, it assigns them as a "potential" ShVCORE. Some subtleties arise as the condition $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t+1$ has to be met for each $P_j \in$ ShVCORE. Namely, the "first" $2t+1$ parties whose Com instances have been terminated might not satisfy this condition. The way out is that D and the other parties should not "immediately" terminate an instance Com_j after receiving a "valid" WCORE_j from the Committer P_j; rather they dynamically "update" ShVCORE and WCORE_j by including new "qualified" parties in these sets using the $(\texttt{Sign-Sent}, \star, \star)$ and the $(\texttt{Sign-Received}, \star, \star)$ messages, which are broadcasted in the Com protocol. The process of updating is repeated till D finds a potential set of committers ShVCORE with the desired overlap in the corresponding WCORE_js and broadcasts the same, after which the parties terminate the sharing protocol. The idea is that if D is *honest*, then eventually every honest party will be included in the WCORE of every other honest party (provided D dynamically updates the WCOREs as above beyond its recommended size of $2t+1$) and there are at least $2t+1$ honest parties and so D will eventually find the desired set of committers.

We now briefly sketch the reconstruction protocol where (R1) and (R2) are met. The reconstruction protocol consists of two main steps. First, we weed out a number of corrupted parties from ShVCORE based on several tests and assign the rest of the parties in RecVCORE. Second, the information published by the parties in RecVCORE is used to reconstruct the share of every party in ShVCORE, satisfying (R1) and (R2). More concretely, a three-stage test that every party P_j in ShVCORE must pass is as follows: First, as a committer, $P_j \in$ ShVCORE must decommit a polynomial $\overline{f}_j(x)$ (and not $\perp$) in Decom_j; second, P_j should successfully reveal the secondary signature $\mathsf{ICSig}(P_k, P_j, \mathcal{P}, f_k(j))$ received from committer P_k for *every* P_k in ShVCORE such that $P_j \in \mathsf{WCORE}_k$[22]; third, $f_k(j)$ as revealed above as a part of IC signature should be the same as $\overline{f}_j(k)$ ($\overline{f}_j(x)$ is the decommitted polynomial). If $P_j \in$ ShVCORE fails the three-stage test then it is discarded.

An *honest* $P_j \in$ ShVCORE will pass the three-stage test, due to the correctness property of AWC, the correctness property of AICP for the case of an honest INT and the enforcement of pre-verification mentioned earlier. As soon as $|\mathsf{ShVCORE}| - t$ non-discarded committers have been found, we denote the set by RecVCORE. The information revealed by the committers in RecVCORE is then used to reconstruct the committed shares of all the committers in ShVCORE, satisfying (R1) and (R2). The rest of the details appear in the formal description of protocol AVSS-Single presented in Fig 5.

We now prove the properties of AVSS-Single.

Lemma 8 (Termination) *Protocols* $(\mathsf{Sh}, \mathsf{Rec})$ *satisfy the termination condition of Definition 2 with probability at least* $1-\gamma$*, where* $\gamma = \frac{n^3 t}{|\mathbb{F}|-1}$.

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 (and the parties) do not immediately terminate the instance Com_j after receiving a WCORE_j of size $2t+1$ from Committer 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 ShVCORE $\subseteq \mathcal{T}$ of size at least $2t+1$, such that $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t+1$ holds for every $P_j \in$ ShVCORE. So D will eventually broadcast ShVCORE and WCORE_js corresponding to every $P_j \in$ ShVCORE. By the properties of broadcast, every honest party will eventually receive these sets from the broadcast of D. Moreover, every honest party will eventually receive the desired $(\texttt{Sign-Sent}, \star, \star)$ and $(\texttt{Sign-Received}, \star, \star)$ messages, as D received them while constructing ShVCORE. So every honest party will eventually terminate the protocol Sh. This proves the first requirement.

Let P_{hon} be the first honest party who have terminated Sh. This implies that P_{hon} have received the set ShVCORE of size at least $2t+1$ and the sets WCORE_js corresponding to every $P_j \in$ ShVCORE from the broadcast of D and

[22] Recall that during Com_k, the Committer P_k would have received the *primary signatures* $\{\mathsf{ICSig}(P_j, P_k, \mathcal{P}, f_k(j))\}$ from the parties $P_j \in \mathsf{WCORE}_k$, while every $P_j \in \mathsf{WCORE}_k$ would have received the *secondary signature* $\mathsf{ICSig}(P_k, P_j, \mathcal{P}, f_k(j))$ from the Committer P_k. The secondary signatures were not used in the Decom protocol of the AWC scheme; we will now use them, while executing the Decom instances in the reconstruction protocol of AVSS.

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

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

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

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 invoke an instance of Com to commit the share $f_i(x)$.[a] We call this instance of Com, invoked 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, invoked 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 the Committer P_j, then check whether $f_j(i) \stackrel{?}{=} f_i(j)$. If $f_j(i) = f_i(j)$, then 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. [b]
4. For $j = 1, \ldots, n$, do not terminate and keep participating in the instance Com_j, even after receiving a set WCORE_j of size $2t + 1$ from the broadcast of Committer P_j. [c]

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

1. If a WCORE_j of size $2t + 1$, along with messages $(\texttt{Sign-Received}, k, P_j)$ corresponding to the parties $P_k \in \mathsf{WCORE}_j$ are 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 initialized to $\emptyset$). Do not terminate Com_j and keep participating in the instance Com_j.
2. If the message $(\texttt{Sign-Received}, k, P_j)$ and $(\texttt{Sign-Sent}, k, P_j)$ is received from the broadcast of Committer P_j and party P_k respectively during the instance Com_j, such that $P_j \in \mathcal{T}$ and $P_k \notin \mathsf{WCORE}_j$, then update WCORE_j by including P_k in it. [d]
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 $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t + 1$ holds for every $P_j \in \mathsf{ShVCORE}$. [e]
4. On finding ShVCORE, broadcast the set ShVCORE and the set WCORE_j corresponding to each $P_j \in \mathsf{ShVCORE}$.

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 $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t + 1$ holds for every $P_j \in \mathsf{ShVCORE}$.
2. If the previous checking 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 $P_k \in \mathsf{WCORE}_j$ and the messages $(\texttt{Sign-Received}, k, P_j)$ from the broadcast of P_j, for every $P_k \in \mathsf{WCORE}_j$, during the instance Com_j. On receiving all these messages, terminate the protocol.

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

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 an instance of Decom. Denote this instance of Decom as Decom_j.
2. For every $P_k \in \mathsf{ShVCORE}$ for which $P_j \in \mathsf{WCORE}_k$, 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 DECOMMITTED SHARES, RECONSTRUCTING THE SECRET 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 $P_j \in \mathsf{WCORE}_k$.
3. Construct a dynamic set RecVCORE, which is initialized to $\emptyset$. Include $P_j \in \mathsf{ShVCORE}$ in RecVCORE if all the following hold:
   (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 its 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 $P_j \in (\mathsf{RecVCORE} \cap \mathsf{WCORE}_k)$ 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 bivariate 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 our AWC scheme.

[b] This pre-verification step prevents a corrupted P_j from committing an incorrect share during Com_j.

[c] Recall that as per the protocol-code of Com, a party should (locally) terminate it after receiving and verifying a WCORE of size $2t + 1$ from the corresponding Committer.

[d] This step denotes the update of WCORE_j beyond its initial size of $2t + 1$.

[e] This automatically implies that $|\mathsf{ShVCORE}| \geq 2t + 1$ should hold.

verified that $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t+1$. By the properties of broadcast, every other honest party will also eventually receive these sets which are verified consequently. Party P_{hon} must have received all the desired $(\texttt{Sign-Sent}, \star, \star)$ and $(\texttt{Sign-Received}, \star, \star)$ messages from the broadcast of the corresponding parties before terminating Sh. The properties of broadcast imply that every other honest party will also eventually receive the same messages and will eventually terminate the protocol Sh. This proves the second requirement.

If the honest parties terminate Sh, then they know a set ShVCORE of size at least $2t+1$, with at least $|\mathsf{ShVCORE}|-t$ honest parties in it. During the protocol Rec, each honest party (in ShVCORE) will honestly perform all the steps required during DECOMMITTING THE SHARE; namely except with probability at most δ, it will be able to decommit its share, which is a univariate polynomial of degree at most t, where $\delta = \frac{n^2 t}{|\mathbb{F}|-1}$ (follows from the correctness property of AWC, substituting $\ell = 1$). Moreover each such honest party will be able to correctly reveal any secondary signature (which it is required to), except with probability at most μ, where $\mu = \frac{nt}{|\mathbb{F}|-1}$ (follows from the properties of AICP, substituting $\ell = 1$). So even if the corrupted parties in ShVCORE do not perform their steps correctly during the Rec protocol, the honest parties from ShVCORE will be eventually included in the set RecVCORE, except with probability at most $|\mathsf{ShVCORE}| \cdot \delta \leq \frac{n^3 t}{|\mathbb{F}|-1}$. It is now easy to see that once the set RecVCORE is constructed, every honest party will eventually output either an $\overline{s} \in \mathbb{F}$ 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. During the instance Com_i corresponding to an *honest* party P_i, Adv will obtain at most t AWC-shares of the share $f_i(x)$. These t AWC-shares are already known to Adv, as these can be computed 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 show 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 symmetric bivariate polynomials of degree t, as given in [10]; 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 bivariate 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 bivariate polynomial

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

Note that $Z(x,y)$ is a symmetric bivariate 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 bivariate 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* $(\mathsf{Sh}, \mathsf{Rec})$ *satisfy the correctness condition of Definition 2 with probability at least* $1-\gamma$, *where* $\gamma = \frac{n^3 t}{|\mathbb{F}|-1}$.

PROOF: Let P_{hon} be the first honest party to terminate Sh; this implies that P_{hon} has received the set ShVCORE of size at least $2t+1$ and the corresponding WCORE_js from D and verified that $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t+1$ for every $P_j \in \mathsf{ShVCORE}$. Let $\mathcal{H}$ be the set of honest parties in ShVCORE, so $|\mathcal{H}| \geq |\mathsf{ShVCORE}| - t \geq t+1$. 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 bivariate 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 $\overline{s}$ with probability at least $1-\gamma$; 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 $f_j(x) = F(x,j)$ and $F(x,y)$ is the polynomial selected by D. This is because each *honest* party $P_i \in \mathsf{WCORE}_j$ must have pre-verified that the AWC-share $\overline{f}_j(i)$ received from P_j during Com_j satisfies the condition $\overline{f}_j(i) = f_i(j)$ before participating in Com_j (namely exchanging the primary and secondary signatures); here $f_i(x)$ denotes the share of s received by P_i from D. Moreover, there are at least $t+1$ such honest party P_i in WCORE_j, whose $f_i(j)$ uniquely define the share $f_j(x)$ of P_j (follows from the properties of symmetric bivariate polynomials) and so $\overline{f}_j(x) = f_j(x)$.
   We next claim that during Rec, the share $\overline{f_k}(x)$ computed on the behalf of a party $P_k \in \mathsf{ShVCORE}$ is the same as $f_k(x) = F(x,k)$, except with probability at most δ, where $\delta = \frac{n^2 t}{|\mathbb{F}|-1}$; this will further imply that $s = F(0,0)$ will be output, except with probability at most $|\mathsf{ShVCORE}| \cdot \delta \leq \frac{n^3 t}{|\mathbb{F}|-1}$. 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 AWC ensures that if Committer is honest, then the polynomial (of degree at most t) committed by it 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 δ; this is because the (correctness) property of AWC ensures that if Committer is corrupted and the output of Decom is a polynomial of degree at most t, then except with probability δ, 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) $P_k \notin \mathsf{RecVCORE}$: In this case, $\overline{f}_k(x)$ is computed by interpolating the points $\{(j, f_k(j))\}$, where $P_j \in (\mathsf{RecVCORE} \cap \mathsf{WCORE}_k)$ and P_j (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 (decommitted) share on the behalf of P_j during Rec; i.e. $f_k(j) = \overline{f}_j(k)$ holds. We first notice that there will be at least $t+1$ such interpolating points $\{(j, f_k(j))\}$. This follows from the fact that $P_k \in \mathsf{ShVCORE}$ implies that during Sh, the condition $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_k| \geq 2t+1$ was true 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. This is because $\mathsf{RecVCORE} \subset \mathsf{ShVCORE}$ with $|\mathsf{RecVCORE}| = |\mathsf{ShVCORE}| - t$. Now we have already shown in the previous case that the share $\overline{f}_j(x)$ computed on the behalf of each $P_j \in \mathsf{RecVCORE}$ is the same as the original share $f_j(x)$ except with probability δ; 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 δ, $\overline{f}_k(x) = f_k(x)$.
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 not lie on a unique symmetric bivariate polynomial of degree t. We show that except with probability at most δ, 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 $|\mathcal{H}| \cdot \delta \leq n \cdot \delta$, every honest party will output $\perp$, irrespective of the shares which are computed on behalf of the corrupted parties in ShVCORE.
   If $P_k \in \mathsf{RecVCORE}$, 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 Decom_k and the correctness property of AWC ensures that for an honest Committer, the committed polynomial will be obtained correctly during Decom. If $P_k \notin \mathsf{RecVCORE}$, then also the claim is true, as in this case $\overline{f}_k(x)$ computed on the behalf of P_k is obtained by interpolating the points $\{(j, f_k(j))\}$, where $P_j \in (\mathsf{RecVCORE} \cap \mathsf{WCORE}_k)$ 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 the Committer P_k during the instance Com_k. The **AICP-Correctness3** property ensures that each revealed point $\{(j, f_k(j))\}$ indeed lies on the original polynomial $f_k(x)$, which was committed by P_k during Com_k, except with probability at most μ, where $\mu = \frac{nt}{|\mathbb{F}|-1}$ (by substituting $\ell = 1$); as there can be at most n such interpolating points (on the behalf of P_k), $\overline{f}_k(x) = f_k(x)$ will be true, except with probability at most $n\mu \leq \delta$. □

Theorem 4 *Protocols* $(\mathsf{Sh}, \mathsf{Rec})$ *is a* $(1-\gamma)$*-AVSS scheme for a single secret, where* $\gamma = \frac{n^3 t}{|\mathbb{F}|-1}$. *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 in-

stances 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 1 by substituting $\ell = 1$, Theorem 2 and from Lemmas 8-10. □

5.2 AVSS for Sharing ℓ Secrets

A simple way to share and later reconstruct a vector of secrets $\vec{S} = (s^1, \ldots, s^\ell)$, consisting of ℓ elements from $\mathbb{F}$, where $\ell > 1$, is to execute a dedicated instance of the Sh and Rec protocol for 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. We next 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. For $\ell = \Omega(n)$, the broadcast communication of our protocol will be then $\mathcal{O}(\ell n^2 \log \frac{1}{\epsilon})$ bits, instead of $\mathcal{O}(\ell n^3 \log \frac{1}{\epsilon})$ bits. This is a significant gain in the communication complexity, considering the fact that communication-wise, implementing broadcast through the A-cast protocol over a point-to-point network is expensive.

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 bivariate polynomial $F_l(x, y)$ for sharing each $s^l \in \vec{S}$ and computes ℓ ith shares $f_{1,i}(x), \ldots, f_{\ell,i}(x)$, 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 shares, party P_i executes a *single* instance of Com to commit all the ℓ polynomials (shares) concurrently. The rest of the protocol steps of 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 AVSS-Multiple, as it deals with multiple secrets. The properties of the modified scheme follow using the similar arguments as for the earlier scheme.

We state the following theorem, stating the communication complexity of the protocol AVSS-Multiple, whose proof follows from the properties of the protocol and the communication complexity of our AICP (Theorem 1) and AWC (Theorem 3).

Theorem 5 *Protocols* $(\mathsf{Sh}, \mathsf{Rec})$ *is a* $(1 - \gamma)$*-AVSS scheme for* ℓ *secrets, where* $\gamma = \frac{n^3(\ell+t-1)}{|\mathbb{F}|-\ell}$. *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.*

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

This section starts with the description of the existing common coin protocol from [7] for generating a single common coin based on any AVSS scheme sharing a *single* secret. With the goal of constructing a more efficient common coin protocol, we substitute the AVSS scheme in the existing common coin protocol by AVSS-Multiple. This step requires an additional technique to surpass the issues arising from the fact that individual secrets are not allowed to be reconstructed in AVSS-Multiple and the reconstruction of a *single* shared secret leads to the reconstruction of *all* the secrets shared in the scheme. We further propose a trick that allows to generate $n - 2t = \Theta(n)$ random[23] common coins concurrently with no additional communication. Thus, our protocol is a multi-bit common coin protocol. Looking ahead, our multi-bit common coin protocol leads to an ABA protocol that allows to agree on $\Theta(n)$ bits concurrently.

6.1 Existing Common Coin Protocol

Let $(\mathsf{Sh}, \mathsf{Rec})$ be a given AVSS scheme, for sharing and reconstructing a *single* secret. The existing common coin protocol [7], 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. The ith secret shared by each party is "associated" with the party P_i. Once a party P_i terminates any $t + 1$ instances of Sh, corresponding to $t + 1$ secrets associated with P_i, it broadcasts the identities of the dealers, who have shared those $t + 1$ secrets. We say that these $t+1$ secrets are *attached* to P_i and later these $t+1$ secrets are reconstructed to compute a "value", say V_i, that will be *associated* with P_i.

During the second stage, after terminating the Sh instances corresponding to 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 "sufficient" number of parties, it 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 V_is for every P_i, that it knows is associated with a value. Each party then computes its binary output based on the V_is, in a way described in the protocol, recalled in Fig. 7. If the underlying AVSS has an associated error parameter ϵ' (i.e. $(\mathsf{Sh}, \mathsf{Rec})$ is a $(1 - \epsilon')$-AVSS scheme), then the protocol CC is a $(1 - n^2\epsilon')$-completing, t-resilient, $\frac{1}{4}$-common coin protocol; the complete proof can be found in [7]. Protocol CC requires $\mathcal{O}(n^{11}\kappa^4)$ bits of private communication

[23] Recall that $n = 3t + 1$ and so $n - 2t = t + 1 = \Theta(n)$.

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

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

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

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 $F_l(0, 0) = s^l$. For $i = 1, \ldots, n$, compute $f_{l,i}(x) \stackrel{def}{=} F_l(x, i)$.
2. For $i = 1, \ldots, n$, send the *shares* $f_{1,i}(x), \ldots, f_{\ell,i}(x)$ 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 invoke an instance of Com to commit the shares $f_{1,i}(x), \ldots, f_{\ell,i}(x)$. We call this instance of Com, invoked 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, invoked by the Committer 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 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 (and verifying) a 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 $2t + 1$, along with messages $(\texttt{Sign-Received}, k, P_j)$ corresponding to the parties $P_k \in \mathsf{WCORE}_j$ are 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 initialized to $\emptyset$). Do not terminate Com_j and keep participating in the instance Com_j.
2. If the message $(\texttt{Sign-Received}, k, P_j)$ and $(\texttt{Sign-Sent}, k, P_j)$ is received from the broadcast of Committer P_j and party P_k respectively during the instance Com_j, such that $P_j \in \mathcal{T}$ 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 $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t + 1$ holds for every $P_j \in \mathsf{ShVCORE}$.
4. On finding ShVCORE, broadcast the set ShVCORE and the set WCORE_j corresponding to each $P_j \in \mathsf{ShVCORE}$.

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 $|\mathsf{ShVCORE} \cap \mathsf{WCORE}_j| \geq 2t + 1$ holds for every $P_j \in \mathsf{ShVCORE}$.
2. If the previous checking 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 $P_k \in \mathsf{WCORE}_j$ and the messages $(\texttt{Sign-Received}, k, P_j)$ from the broadcast of P_j, for every $P_k \in \mathsf{WCORE}_j$, during the instance Com_j. On receiving all these messages, terminate the protocol.

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

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. For every $P_k \in \mathsf{ShVCORE}$ for which $P_j \in \mathsf{WCORE}_k$, 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 DECOMMITTED SHARES, RECONSTRUCTING THE SECRETS 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 dynamic set RecVCORE, which is initialized to $\emptyset$. Include $P_j \in \mathsf{ShVCORE}$ in RecVCORE if all the following hold:
   (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 its 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 $P_j \in (\mathsf{RecVCORE} \cap \mathsf{WCORE}_k)$ and P_j revealed the secondary signature $\mathsf{ICSig}(P_k, P_j, \mathcal{P}, (f_{1,k}(j), \ldots, f_{\ell,k}(j)))$.
6. If for $l = 1, \ldots, \ell$, the shares $\{\overline{f}_{l,k}(x) : P_k \in \mathsf{ShVCORE}\}$ lie on a unique, symmetric bivariate 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.

Fig. 7 Existing common coin protocol based on any AVSS scheme (Sh, Rec).

CC

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 invoke 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}.
4. Construct a dynamic set $\mathcal{T}_i$, which is initialized to $\emptyset$. On terminating all the n instances $\mathsf{Sh}_{j1}, \ldots, \mathsf{Sh}_{jn}$ invoked by P_j (as a D), add 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 initialized to $\emptyset$. 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 initialized to $\emptyset$. 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 exists 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 to ensure that the probability computations satisfy certain conditions while proving the properties of the protocol; see [7] for the details.

and $\mathcal{O}(n^{11}\kappa^2 \log n)$ bits of broadcast communication (see Appendix A).

6.2 A More Efficient Multi-bit Common Coin Protocol

Our AVSS scheme AVSS-Single that claims the best known communication complexity for sharing a single secret, readily gives a common coin protocol that improves over the communication complexity of protocol CC by a factor of[24] $\Omega(n^6)$. Our demonstration in Section 5.2 shows that sharing n secrets using a single instance of AVSS-Multiple is "cheaper" by a factor of n, than using n dedicated instances of AVSS-Single. Thus, we can get an even more efficient common coin protocol, by plugging in a *single* instance of AVSS-Multiple for each party, capable of handling n secrets, in the protocol CC. However, this replacement raises a subtle difference in the execution of CC. In CC, the secrets were reconstructed in an "on-demand" basis and so the secrets shared by a party are reconstructed independently of each other at different point of time as and when they are needed to be reconstructed. On contrary, in the modified CC (where AVSS-Multiple is used), concurrent reconstruction of all the secrets shared by a party is unavoidable, because AVSS-Multiple "ties up" all the shared secrets and supports concurrent sharing and reconstruction of *all* the shared secrets. In what follows, we demonstrate that the adversary can take the concurrent reconstruction to its advantage and completely "bias" the output of the common coin protocol.

More concretely, let P_k be an *honest* party, who shares $\vec{S}_k = (x_{k1}, \ldots, x_{kn})$ in CC using AVSS-Multiple. Now the adversary can schedule the messages in the protocol in such a way that a "situation" is created where a particular secret, say x_{ki} *alone*, needs to be reconstructed, leading to the invocation of the reconstruction protocol of AVSS-Multiple. This leads to the adversary learning the *entire* vector $\vec{S}_k$. This allows the adversary to bias the value V_j attached to an honest party P_j by fixing the secrets to be shared by the *corrupted* parties [25] corresponding to P_j. Thus, V_js of some of the honest parties are no longer random values. As a result, one of the crucial lemmas in the correctness proof of the protocol CC is not true anymore for the modified CC, which in turn, leads to a biased and adversarially chosen output in the common coin protocol. Below, we recall the lemma statement verbatim.

[24] If we use AVSS-Single in protocol CC, then the resultant common coin protocol will incur a private communication of $\mathcal{O}(n^5 \log \frac{1}{\epsilon})$ bits and broadcast of $\mathcal{O}(n^5 \log \frac{1}{\epsilon})$ bits, as $\Theta(n^2)$ instances of AVSS-Single are invoked.

[25] Recall that the jth secret of each party's vector can contribute to decide V_j.

Lemma 11 ([7]) *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 to the set $\mathcal{G}_j$, then a unique value, say V_i, is fixed such that the following holds:*

1. *All the honest parties will associate the value V_i with P_i, except with probability at most $n\epsilon'$.*
2. *V_i is distributed uniformly over $[0, \ldots, u]$ and is independent of values associated with the other parties.*

To foil the attack described above, we employ a technique that prevents a corrupted party to share its secret vector via the sharing protocol of AVSS-Multiple *after* it learns the secret vector of any honest party. [26] Lastly, we employ a trick in our common coin protocol that allows to output $\ell = n - 2t = t + 1 = \Theta(n)$ common coins (instead of a single coin), *without* requiring any *additional* communication from the parties. We denote the resultant protocol by MCC to emphasize that it is a multi-bit common coin protocol.

The High Level Idea of MCC**:** First, we note that CC can be converted to a multi-bit common coin protocol by asking each party to "attach" itself with $2t + 1$ secrets, each from a different dealer, in Step 5 of CC. That is, instead of ensuring that $|T_i| = t + 1$, party P_i ensures that $|T_i| = 2t + 1$. This can be enforced without any harm, as there are at least $2t+1$ honest parties who will be eventually included in $\mathcal{T}_i$ of every honest party P_i. This step enables to associate $n - t$ values, instead of a single value V_i, with a party P_i via an application of the randomness extraction algorithm EXT. Next, we introduce/modify some of the steps of protocol CC to foil the attack of biasing the output of common coin protocol:

M1. In step 4, a party P_i includes P_j in $\mathcal{T}_i$ after it is confirmed that $n - t$ parties have terminated the instance Sh_j.

M2. In step 7, party P_i upon confirming that $|\mathcal{S}_i| = 2t + 1$, broadcasts a `Reconstruct Enabled` message, indicating that it is "ready" to participate in the relevant Rec instances. However, only after receiving the `Reconstruct Enabled` message from $n - t$ parties, it actually starts participating in the corresponding Rec instances.

M3. As soon as P_i broadcasts `Reconstruct Enabled`, it stops participating in the Sh instances of all the parties, which are not present in its $\mathcal{T}_i$ 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 its participation in Sh_j only if P_j is included in $\mathcal{T}_i$.

We now argue that protocol MCC which incorporates the above changes is not vulnerable to the attack discussed earlier. The argument is made in three clear steps, each relying on a different step (i.e. either on **M1** or **M2** or **M3**). Let P_j be a *corrupted* party, who wants to select and share its secret vector $\vec{S}_j$ *after* seeing the reconstructed secret vector of some honest party. Such a P_j have to delay its sharing instance (namely Sh_j) until at least $2t + 1$ parties broadcast the `Reconstruct Enabled` message (due to **M2**). This implies that at least $t + 1$ *honest* parties, say H, who broadcast `Reconstruct Enabled` message are yet to terminate Sh_j at the time they initiated the broadcast. Therefore, the parties in H stop participating in Sh_j from then onwards (due to **M3**). So P_j cannot enter into $\mathcal{T}_i$ of any *honest* P_i. This is because for P_j to enter $\mathcal{T}_i$ of any honest P_i, the instance Sh_j must be terminated by at least $2t + 1$ parties (due to **M1**) which must include one party from the set H.[27] However, no party from H will ever terminate Sh_j, as they stopped participating in Sh_j. Protocol MCC is now presented in Fig 8.

We now proceed to prove the properties of the protocol MCC. Most of the properties of MCC follow from the properties of the protocol CC given in [7]. For the sake of completeness, we will present them here. As in [7], 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 $\vec{S}$ of n values is fixed, such that each honest party will eventually complete the corresponding instance of Rec and output $\vec{S}$. Moreover, if the dealer of this instance of Sh is honest, then $\vec{S}$ 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 $1 - n\gamma$ (as there are n instances of AVSS), where $\gamma = \frac{n^3(n+t-1)}{|\mathbb{F}|-n}$ (follows from Theorem 5 by substituting $\ell = n$). This implies that the event E occurs with probability at least $1 - \frac{\epsilon}{4n}$.[28]

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

[26] In the context of distributed key generation for threshold Discrete-Log based cryptosystems, [18] demonstrated a somewhat similar attack of biasing the distribution of a jointly computed key. While the "nature" of their attack is similar to the attack presented here, the attacks themselves are different. Saying differently, the way adversary mounts the attack is completely different in these two scenarios. Thus, their technique to foil the attack is inapplicable in our context. Specifically, the attack in their work exploits the computational security of the underlying VSS scheme. On contrary, our attacker takes advantage of the concurrent reconstruction feature of our underlying VSS scheme AVSS-Multiple. While [18] foils the attack by strengthening the security of their VSS scheme, we prevent the attack by a technique that bars a corrupted party to share its secret vector $\vec{S}$ after seeing the reconstructed secret vector of honest parties.

[27] Note that H contains at least $t + 1$ parties. So any set of $2t + 1$ parties will surely contain at least one party from H, as the total number of parties is $3t + 1$.

[28] Since the field $\mathbb{F}$ is selected such that $\epsilon \geq \frac{4n^5(n+t-1)}{|\mathbb{F}|-n}$, where ϵ will be the error probability of our ABA protocol, we have that $\frac{\epsilon}{4n} \geq \frac{n^4(n+t-1)}{|\mathbb{F}|-n} = n\gamma$.

Fig. 8 Multi-Bit common coin protocol.

Protocol MCC

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 invoke the protocol $\mathsf{Sh}(P_i, \mathcal{P}, \vec{S_i})$, 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 initialized to $\emptyset$. 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 initialized to $\emptyset$. 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 initialized to $\emptyset$. 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 $u = \lceil 0.87n \rceil$. For every party $P_j \in \mathcal{G}_i$, *associate* $n - 2t = t + 1$ values $(V_{j1}, \ldots, V_{j(n-2t)})$ with P_j 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 for every $l \in \{1, \ldots, n - 2t\}$, do the following:
    (a) If there exits a party $P_j \in H_i$ such that $V_{jl} = 0$ then (locally) set $\sigma_l = 0$ as the lth output bit.
    (b) Otherwise (locally) set $\sigma_l = 1$ as the lth output bit.
    Locally output the $\sigma_1, \ldots, \sigma_{n-2t}$ and terminate.

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 a 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 a constant time. Also the protocol for the broadcast terminates in a 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 be eventually 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 the honest P_i above will be eventually satisfied for

every other honest P_j. So P_j will eventually broadcast the `Reconstruct Enabled` message. □

The following important lemma shows that in the protocol MCC, the adversary strategy discussed earlier for biasing the distribution of the value associated with a party, is not applicable.

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

PROOF: Let P_k 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 $\mathcal{T}_k$. To prove the lemma, we first assert that a *corrupted* party P_j will be never included in the set $\mathcal{T}_i$ of *any honest* P_i, if P_j selects and shares its secrets *after* P_k started participating in the Rec instances corresponding to the parties in $\mathcal{T}_k$. We prove this by contradiction.

So let P_k 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_j executes the instance Sh_j only after P_k received the `Reconstruct Enabled` message from the parties in $\mathcal{B}_1$. Furthermore, assume that P_j is included in the set $\mathcal{T}_i$ of some honest P_i. Now $P_j \in \mathcal{T}_i$ implies that P_i must have received the message (P_m `terminated` j) from the broadcast of at least $n-t$ parties P_m, say $\mathcal{B}_2$, which implies that the (honest) parties in $\mathcal{B}_2$ have terminated the instance Sh_j 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_j before broadcasting the `Reconstruct Enabled` message; otherwise $P_\alpha \in \mathcal{B}_2$ would have stopped participating in the instance Sh_j and would never broadcast the message (P_α `terminated` j). This is because during the step 8(b) of the protocol, P_α would have stopped participating in the instance Sh_j while broadcasting the `Reconstruct Enabled` message if P_α has not already terminated the instance Sh_j. This further implies that P_j must have executed the instance Sh_j (which the honest P_α have completed) *before* P_k started participating in the Rec instances. But this is a contradiction to our assumption.

Hence if the corrupted P_j is included in $\mathcal{T}_i$ of *any* honest P_i then it must have invoked the instance Sh_j before any honest party started participating in any Rec instance. Thus while choosing its own secrets for the instance Sh_j, the corrupted P_j will have no knowledge about the secrets shared by the honest parties in their instances of Sh, which follows from the secrecy property of Sh. □

Lemma 14 *Let $u = \lceil 0.87n \rceil$. In the protocol* MCC, *once an honest party P_j receives the message* (`Attach` T_i `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. *All the honest parties will associate $V_{i1}, \ldots, V_{i(n-2t)}$ with P_i, except with probability at most $\frac{\epsilon}{4n}$.*
2. *Each value $V_{i1}, \ldots, V_{i(n-2t)}$ is uniformly distributed 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, all the honest parties will eventually reconstruct $\vec{S_k} = (x_{k1}, \ldots, x_{ki}, \ldots, x_{kn})$ at the completion of Rec_k, except with probability at most γ, where $\gamma = \frac{n^3(n+t-1)}{|\mathbb{F}|-n}$ (follows from Theorem 5 by substituting $\ell = n$). As there are $2t+1$ parties in the set T_i, except with probability at most $n\gamma \leq \frac{\epsilon}{4n}$, all the honest parties 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 the 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* `Reconstruct Enabled` *message, 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 belong 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 ($\texttt{Attach}\ T_j\ \texttt{to}\ P_j$) 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 a 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 any $l \in \{1, \ldots, n-2t\}$, for every possible value $\sigma_l \in \{0,1\}$, with probability at least $\frac{1}{4}$, all the honest parties output σ_l as the lth bit.*

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 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 $(1-\frac{\epsilon}{4n})$, 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 fix an $l \in \{1, \ldots, n-2t\}$ and consider the following two cases:

- We show that the probability of outputting $\sigma_l = 0$ as the lth bit 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 if all the honest parties associate V_{jl} (as the lth value) with P_j, then clearly all the honest parties will output $\sigma_l = 0$. The probability that for at least one party $P_j \in M$, the value $V_{jl} = 0$ is $1-(1-\frac{1}{u})^{|M|}$. Now $u = \lceil 0.87n \rceil$. Also $|M| \geq \frac{n}{3}$. Therefore for all $n > 4$, we have $1-(1-\frac{1}{u})^{|M|} \geq 0.316$. So, the probability that all the honest parties output $\sigma_l = 0$ is at least $0.316 \times (1-\frac{\epsilon}{4n}) \geq 0.25 = \frac{1}{4}$.
- We show that the probability of outputting $\sigma_l = 1$ as the lth bit by all the honest parties is at least $\frac{1}{4}$. It is obvious that if *no* party P_j has $V_{jl} = 0$ and if all honest parties associate V_{jl} with P_j, then all the honest parties will output $\sigma_l = 1$. As $u = \lceil 0.87n \rceil$, the probability of this event is at least $(1-\frac{1}{u})^n \cdot (1-\frac{\epsilon}{4n}) \geq e^{-1.15} \cdot 0.95 \geq 0.25 = \frac{1}{4}$. □

Theorem 6 *For every ϵ, where $0 < \epsilon \leq 0.2$, protocol* MCC *is a $(1-\frac{\epsilon}{4n})$-completing, t-resilient, $\frac{1}{4}$-multi-bit 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 a 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 5 by substituting $\ell = n$. The theorem now follows from Lemmas 12-16. □

7 Existing Voting Protocol

In this section, we recall the existing vote protocol from [7], which will be required for the construction of our ABA protocol (along with the multi-bit common coin protocol).

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 9. In the protocol, party P_i have the input bit x_i.

The properties of the protocol VOTE are stated in the following lemmas and the proofs (taken from [7]) are available in APPENDIX B.

Lemma 17 ([7]) *All the honest parties terminate the protocol* VOTE *in a constant time.*

Lemma 18 ([7]) *If all the honest parties have the same input σ, then all the honest parties will output $(\sigma, 2)$.*

Lemma 19 ([7]) *If some honest party outputs $(\sigma, 2)$, then every other honest party will eventually output either $(\sigma, 2)$ or $(\sigma, 1)$.*

Lemma 20 ([7]) *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 7 *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, 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 [7]. We first informally discuss the underlying idea used in [7] for reaching agreement on a *single* bit by using protocol CC and 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 its 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 it finds a "distinct majority" for the value σ, then the party sets its modified input for the next iteration to σ, irrespective of the value which is going to be output in the instance of CC; otherwise, the party sets its 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 it finds an "overwhelming majority" for the value σ, then it broadcasts σ. Finally, once a party receives σ from the broadcast of $t + 1$ parties, it 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 its 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. A party repeats the same logic explained as above to set each of its modified bit for the next iteration taking into account the outcome of the VOTE and MCC. More specifically, after completing the iteration k, each party sets the lth bit of its modified input for the $(k + 1)$th iteration as follows, for each $l \in \{1, \ldots, n - 2t\}$: if $(\sigma_l, 1)$ is obtained as the output of the lth instance of VOTE during the kth iteration, then

Fig. 9 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 initialized to $\emptyset$. 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 initialized to $\emptyset$. 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, which is initialized to 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 $P_j \in B_i$ have 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.

Fig. 10 Multi-bit ABA protocol to reach agreement on $n - 2t = t + 1$ bits.

MABA(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 termination:
   (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 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 $(\texttt{Terminate with output } v_{(r+1)l}, l)$ and participate in one more instance of VOTE protocol corresponding to the l^{th} bit with $v_{(r+1)l}$ as the input.[b]
      Participate in one more instance of MCC if $(\texttt{Terminate with output } v_{(r+1)l}, l)$ 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 $(\texttt{Terminate with output } \sigma_l, l)$ 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 $(\texttt{Terminate with output } \sigma_l, l)$ broadcasts are completed.

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 10.

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 [7], 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; consequently every honest party will broadcast $(\texttt{Terminate with output } \sigma_l, l)$. □

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 $l \in \{1, \ldots, n - 2t\}$, 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 $(\texttt{Terminate with output } \sigma_l, l)$,

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 $(\sigma_l, 2)$ or $(\sigma_l, 1)$ during this instance of the VOTE protocol. In case P_j outputs $(\sigma_l$, 2), then 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 corrupted 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 for any $l \in \{1, \ldots, n-2t\}$ where $\mathsf{flag}_l = 0$, with probability at least $\frac{1}{4}$, all the honest parties will have the same lth modified input $v_{(k+1)l}$ for the $(k+1)$th iteration.*

PROOF: For any $l \in \{1, \ldots, n-2t\}$ 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 lth output bit obtained in the MCC protocol (during the kth iteration). For such a $v_{(k+1)l}$, we have two possible cases:

- *All the honest parties execute step 2(c)-(iii) in iteration k for setting $v_{(k+1)l}$*: in this case 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 lth output bit after completing MCC (see Lemma 16) and hence with the same probability, all the honest parties will have the same value for $v_{(k+1)l}$.
- *Some honest party(ies) execute either step 2(c)-(i) or step 2(c)-(ii) during iteration k for setting $v_{(k+1)l} = \sigma_l$ for some $\sigma_l \in \{0,1\}$, while the remaining honest party(ies) execute step 2(c)-(iii) for setting $v_{(k+1)l}$*: in this case, first of all 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 all the honest parties will have σ_l as the lth output bit at the end of MCC 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 $v_{(k+1)l} = \sigma_l$. □

Before proceeding further, we 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 will set $\mathsf{flag}_l = 1$ in a constant expected time for any $l \in \{1, \ldots, n-2t\}$.*

PROOF: Let us fix an $l \in \{1, \ldots, n-2t\}$; we first claim that all the honest parties will set $\mathsf{flag}_l = 1$ within a constant time, after the *first* instance (iteration) when some honest party broadcasts (`Terminate with output` σ_l, l), for some $\sigma_l \in \{0,1\}$. So let the first instance when an honest party broadcasts (`Terminate with output` σ_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 (`Terminate with output` σ_l, l) by the end of iteration r_l+1. All these instances of broadcast complete in a 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 (`Terminate with output` σ_l, l), all the honest parties will set $\mathsf{flag}_l = 1$ in a constant time.

We next count the expected number of iterations until (`Terminate with output`σ_l, l) is broadcasted by some honest party, for some $\sigma_l \in \{0,1\}$. More specifically, let the random variable τ_l count r_l; if $\tau_l = \infty$, then the honest parties will not terminate MABA, as the honest parties will wait to set flag_l to 1. Conditioned on the event C, all the honest parties terminate each iteration in a constant time. To complete the proof, it is enough to show that $E(\tau_l|C)$ is a constant. We have

$$\begin{aligned} Prob(\tau_l > k|C_k) \leq\ & Prob(\tau_l \neq 1|C_k) \times \ldots \times \\ & Prob(\tau_l \neq k|C_k \cap \tau_l \neq 1 \\ & \ldots \cap \tau_l \neq k-1). \end{aligned}$$

From Lemma 23, after completing an iteration k, all the honest parties will have the same lth modified input 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 $Prob(\tau_l > k|C_k) \leq (\frac{3}{4})^k$. Now it follows via a simple calculation that $E(\tau_l|C) \leq 16$. □

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

PROOF: For $l \in \{1, \ldots, n-2t\}$, let r_l and τ_l be the quantities as described in the proof of Lemma 24. We compute the probability of the event $\overline{C}$. We have

$$\begin{aligned} Prob(\overline{C}) &\leq \sum_{k\geq 1} Prob(\exists l \in \{1,\ldots,n-2t\} : \tau_l > k \\ &\qquad \cap\ \overline{C_{k+1}}|C_k), \\ &\leq \sum_{k\geq 1}\sum_{l=1}^{n-2t} Prob(\tau_l > k \cap \overline{C_{k+1}}|C_k), \\ &\leq \sum_{k\geq 1}\sum_{l=1}^{n-2t} Prob(\tau_l > k|C_k) \cdot Prob(\overline{C_{k+1}}|C_k \\ &\qquad \cap \tau_l > k). \end{aligned}$$

From the proof of Lemma 24, we have $Prob(\tau_l > k|C_k) \leq (\frac{3}{4})^k$ for any $l \in \{1,\ldots,n-2t\}$. We will now bound the term $Prob(\overline{C_{k+1}}|C_k \cap \tau_l \geq k)$. 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 has an error probability of at most $\frac{\epsilon}{4n}$ for termination (follows from Theorem 6). Thus with probability at least $1-\frac{\epsilon}{4n}$, all the honest parties complete the instance of MCC during the kth iteration. Therefore, for each k, $Prob(\overline{C_{k+1}}|C_k \cap \tau_l \geq k) \leq \frac{\epsilon}{4n}$. So we get

$$Prob(\overline{C}) \leq \sum_{k\geq 1}(\frac{3}{4})^k \cdot \frac{\epsilon}{4n} \cdot (n-2t) \leq \epsilon. \qquad \square$$

Theorem 8 (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 $n-2t$ bits 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 a 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 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 an (expected) constant number of such iterations. The theorem now follows from Lemmas 21-25. □

9 Conclusion and Open Problems

We 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 [8]. Our protocol also has better communication complexity than the almost-surely terminating ABA protocol of [1] (however 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 → AWC → 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 [8] 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 with 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 our ABA protocol. Improving the communication complexity of the almost-surely terminating ABA protocol of [1] is another interesting open problem. In that regard, it may be worth defining a "shunning" variant of AWC and investigating if the new primitive can be used to design SVSS more efficiently than the known construction. Furthermore, one can try to find the applicability of our tricks for dealing with multiple secrets concurrently in the context of the ABA of [1]. Perhaps the most challenging open problem is to get an almost-surely terminating, optimally-resilient, ABA protocol with a *constant* expected running time and with low communication complexity.

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. We would also like to acknowledge them for pointing out certain flaws in the probability calculations. Finally we would like to thank Prof. Nigel Smart for several comments on various parts of the paper.

The work of the first author has been supported in part by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO and by EPSRC via grant EP/I03126X. The work of the second author has been supported in part by EPSRC via grant EP/I03126X.

References

1. I. Abraham, D. Dolev, and J. Y. Halpern. An almost-surely Terminating Polynomial Protocol for Asynchronous Byzantine Agree-

ment 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. G. Bracha. An Asynchronous $\lfloor (n-1)/3 \rfloor$-resilient Consensus Protocol. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 154 – 162. ACM Press, 1984.
7. R. Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute, Israel, 1995.
8. 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.
9. A. Choudhury and A. Patra. Brief Announcement: Efficient Optimally Resilient Statistical AVSS and its Applications. In D. Kowalski and A. Panconesi, editors, *Proceedings of the 31st Annual ACM Symposium on Principles of Distributed Computing, PODC 2012, Funchal, Portugal, July 16-18, 2012*, pages 103–104. ACM Press, 2012.
10. R. Cramer and I. Damgård. *Multiparty Computation, an Introduction*. Contemporary Cryptography. Birkhuser Basel, 2005.
11. 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.
12. P. Feldman and S. Micali. An Optimal Algorithm for Synchronous Byzantine Agreement. 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.
13. P. Feldman and S. Micali. An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement. *SIAM Journal of Computing*, 26(4):873–933, 1997.
14. M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *JACM*, 32(2):374–382, 1985.
15. M. Fitzi. *Generalized Communication and Security Models in Byzantine Agreement*. PhD thesis, ETH Zurich, 2002.
16. 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.
17. 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.
18. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *J. Cryptology*, 20(1):51–83, 2007.
19. 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.
20. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
21. 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.
22. 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.
23. M. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *JACM*, 27(2):228–234, 1980.
24. T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Verlag, 1991.
25. 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.
26. 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.
27. 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 [8]

The communication complexity analysis of the AVSS and ABA protocol of [8] 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 [8] given in Canetti's thesis [7]. To bound the error probability by ϵ, all communication and computation in the protocol(s) of [8] are 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 of [8], Signer gives $\mathcal{O}(\kappa)$ field elements to INT and $\mathcal{O}(\kappa)$ field elements to Verifier. Though

the ICP of [7] 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 [7] 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 with n verifiers in Shamir secret-sharing [27], the authors in [8] designed an asynchronous primitive A-RS, consisting of two sub-protocols A-RS-Share and A-RS-Rec. In the A-RS-Share protocol, D generates n (Shamir) shares of a secret s and for each of the n shares, D executes an instance of ICP with n verifiers. So the A-RS-Share protocol of [8] 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. During the A-RS-Rec protocol, the IC signatures given by D during 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 [8] designed an AWSS scheme using their A-RS protocol. The scheme consists of two sub-protocols, namely AWSS-Share and AWSS-Rec. In the AWSS-Share protocol, D generates n (Shamir) shares of the secret and instantiates n instances of ICP 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 ICP. 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 then further extended their AWSS-Share protocol to Two&SumAWSS-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&SumAWSS-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.

Using Two&SumAWSS-Share and AWSS-Rec, the authors in [8] then finally deign their AVSS scheme, consisting of sub-protocols 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 involves a private communication of $\mathcal{O}(n^9\kappa^4)$ 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 [8] 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 [8], common coin protocol is called for $\mathsf{R} = \mathcal{O}(1)$ expected time. Hence the ABA protocol of [8] 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 [8] 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

Proof of Lemma 17: Every honest party P_i will broadcast its 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 a constant time. □

Proof of Lemma 18: 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 its vote $b_k = \sigma$. Hence the honest P_i will output $(\sigma, 2)$. □

Proof of Lemma 19: 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 $(\sigma, 1)$. □

Proof of Lemma 20: 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 $(\Lambda, 0)$. □