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

Efficient Asynchronous Byzantine Agreement with Optimal Resilience

Arpita Patra · Ashish Choudhary · C. Pandu Rangan

Received: date / Accepted: date

Abstract Byzantine agreement (BA) is considered as one of the most fundamental primitives for fault-tolerant distributed computing and cryptographic protocols. BA among a set of n parties each having an input value, allows them to reach agreement on a common value even if some of the parties are faulty and try to prevent agreement among the non-faulty parties.

An important variant of BA is Asynchronous Byzantine Agreement (ABA). An ABA protocol is carried out among n parties in a completely asynchronous network, where every two parties are directly connected by a private channel and t out of the n parties are under the control of a *computationally unbounded Byzantine (active)* adversary $\mathcal{A}_t$. The communication complexity of ABA is one of its most important complexity measures. In this paper, we present a simple and efficient ABA protocol whose communication complexity is significantly better than the communication complexity of the existing ABA protocols in the literature. Our protocol is *optimally resilient* and thus requires $n = 3t + 1$ parties for its execution.

Specifically, the *amortized* communication complexity of our ABA protocol is $\mathcal{O}(\mathcal{C}n^4 \log \frac{1}{\epsilon})$ bits for attaining agreement on a *single* bit, where ϵ denotes the probability of non-termination and $\mathcal{C}$ denotes the *expected running time* of our protocol. Conditioned on the event that our ABA protocol terminates, it does so in constant expected time; i.e., $\mathcal{C} = \mathcal{O}(1)$. We compare our result with most recent optimally resilient, ABA protocols proposed in [17] and [1] and show that our protocol gains by a factor of $\mathcal{O}(n^7 (\log \frac{1}{\epsilon})^3)$ over the ABA of [17] and by a factor of $\mathcal{O}(n^4 \frac{\log n}{\log \frac{1}{\epsilon}})$ over the ABA of [1].

Towards the designing of our efficient ABA protocol, we first present a novel and simple *asynchronous verifiable secret sharing* (AVSS) protocol with $n = 3t + 1$, which significantly improves the communication complexity of the only known AVSS protocol of [17] with $n = 3t + 1$. Our AVSS shares multiple secrets *concurrently* and is far better than multiple parallel executions of AVSS sharing single secret. Thus our AVSS brings forth several advantages of *concurrently* sharing multiple secrets. We believe that our AVSS can be used in

A preliminary version of this paper appeared in PODC 2009. The work of Arpita Patra is supported by Microsoft Research India PhD fellowship. Arpita Patra would also like to thank ACM and IARCS for financial support to attend PODC 2009 and present a preliminary version of this paper. The work of Ashish Choudhary is supported by Infosys Technology India PhD fellowship. Ashish Choudhary would also like to acknowledge financial support from Project No CSE/O5-06/076/DITX/CPAN on Protocols for Secure Communication and Computation, sponsored by Department of Information Technology, Govt. of India to attend PODC 2009, where a preliminary version of this paper is presented. The work of C. Pandu Rangan is supported by project grants of Project No CSE/O5-06/076/DITX/CPAN on Protocols for Secure Communication and Computation, sponsored by Department of Information Technology, Govt. of India.

Arpita Patra
Dept. of Computer Science and Engineering
IIT Madras, Chennai India 600036
Tel.: +91-44-22575370
E-mail: arpitapatra_10@yahoo.co.in, arpitapatra10@gmail.com

Ashish Choudhary
Dept. of Computer Science and Engineering
IIT Madras, Chennai India 600036
Tel.: +91-44-22575370
E-mail: partho_31@yahoo.co.in, partho31@gmail.com

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

many other applications for improving communication complexity and hence is of independent interest.

The common coin primitive is one of the most important building blocks for the construction of ABA protocol. The only known efficient (i.e polynomial communication complexity) common coin protocol [31,16] uses AVSS sharing a single secret as a black-box. Unfortunately, this common coin protocol does not achieve its goal when multiple invocations of AVSS sharing single secret are replaced by single invocation of AVSS sharing multiple secrets. Hence in this paper, we extend the existing common coin protocol to make it compatible with our new AVSS. As a byproduct, our new common coin protocol is much more communication efficient than the existing common coin protocol.

1 Introduction

The problem of Byzantine Agreement (BA) was introduced in [56] and since then it has emerged as the most fundamental problem in distributed computing. It has been used as building block for several important secure distributed computing tasks such as Secure Multiparty Computation (MPC) [66,43,7,60,21,44,3,47,22,5,53,54], Verifiable Secret Sharing (VSS) [18,60,42,36,45,51] etc. Roughly speaking, the BA problem is as follows: there are n parties, each having an input binary value; the goal is for all honest parties to agree on a consensus value. The challenge lies in reaching agreement despite the presence of faulty parties, who may deviate from the protocol arbitrarily. The BA problem has been investigated extensively in various models, characterized by the synchrony of the network, privacy of the channels, computational power of the faulty parties and many other parameters [32,6,12,17,16,50,35,46,2,9–11,13–15,19,24,23,25–28,37,33,34,29,31,39–41,48,49,57,58,65,63,64]. An interesting and practically motivated variant of BA is *Asynchronous* BA (ABA) tolerating a *computationally unbounded* malicious adversary. This problem has got relatively less attention in comparison to the BA problem in synchronous network (see [50,35] and their references). Since asynchronous networks model the real life networks like Internet more appropriately than synchronous networks, the fundamental problem like Byzantine Agreement (BA) is worthy of deep investigation over asynchronous networks.

1.1 The Model and Definition

In this paper, we follow the network model of [17,16]. Specifically, our Asynchronous Byzantine Agreement (ABA) protocol is carried out among a set of n parties, say $\mathcal{P} = \{P_1, \ldots, P_n\}$, where every two parties are directly connected by a secure channel and t out of the n parties can be under the influence of a *computationally unbounded Byzantine (active) adversary*, denoted as $\mathcal{A}_t$. The adversary $\mathcal{A}_t$, completely dictates the parties under its control and can force them to deviate from the protocol in any arbitrary manner. Moreover, we assume $\mathcal{A}_t$ to be *rushing* [51,42,21], who first listens all the messages sent to the corrupted parties by the honest parties, before allowing the corrupted parties to send their messages. The parties not under the influence of $\mathcal{A}_t$ are called *honest or uncorrupted*.

The underlying network is asynchronous, where the communication channels between the parties have arbitrary, yet finite delay (i.e the messages are guaranteed to reach eventually). To model this, $\mathcal{A}_t$ is given the power to schedule the delivery of *all* messages in the network. However, $\mathcal{A}_t$ can *only schedule* the messages communicated between honest parties, without having any access to them. In asynchronous network, the inherent difficulty in designing a protocol comes from the fact that when a party does not receive an expected message then he cannot decide whether the sender is corrupted (and did not send the message at all) or the message is just delayed. So a party can not wait to consider the values sent by all parties, as waiting for all of them could turn out to be endless. Hence the values of up to t (potentially honest) parties may have to be ignored. Due to this the protocols in asynchronous network are generally involved in nature and require new set of primitives. For comprehensive introduction to asynchronous protocols, see [16].

We now formally define ABA.

Definition 1 (ABA [17]) : Let Π be an asynchronous protocol executed among the set of parties $\mathcal{P}$, with each party having a private binary input. We say that Π is an ABA protocol tolerating $\mathcal{A}_t$ if the following hold, for every possible behavior of $\mathcal{A}_t$ and every possible input:

1. **Termination**: All honest parties eventually terminate the protocol.
2. **Correctness**: All honest parties who have terminated the protocol hold identical outputs. Furthermore, if all honest parties had the same input, say ρ, then all honest parties output ρ.

We now define (ϵ, δ)-ABA protocol, where both δ and ϵ are negligibly small values.

Definition 2 ((ϵ, δ)-ABA) : An ABA protocol Π is called (ϵ, δ)-ABA if

1. Π satisfies **Termination**, except with an error probability of ϵ;

2. Π satisfies **Correctness** property, except with an error probability of δ.

The important parameters of any ABA protocol are:

1. **Resilience**: It is the maximum number of corrupted parties (t) that the protocol can tolerate and still satisfy its properties;
2. **Communication Complexity**: It is the total number of bits communicated by the *honest* parties in the protocol; and
3. **Computational Complexity:** It is the computational resources required by the honest parties during a protocol execution. An ABA protocol is called computationally efficient if the computational resources required by each honest party are polynomial in n and $\log \frac{1}{\epsilon}$.
4. **Running Time**: We present an informal, but standard definition of the running time of an asynchronous protocol. For more detailed definition of asynchronous time, see [50]. The current definition is taken from [17,16]. Consider a virtual 'global clock' measuring time in the network. Note that the parties cannot read this clock. Let the *delay* of a message be the time elapsed from its sending to its receipt. Let the *period* of a finite execution of a protocol be the longest delay of a message in the execution. The *duration* of a finite execution is the total time measured by the global clock divided by the period of the execution.
   The *expected running time* of a protocol, *conditioned on an event*, is the maximum over all inputs and applicable adversaries, of the average over the random inputs of the parties, of the duration of executions of the protocol in which this event occurs.

1.2 The History of ABA

From [56,50], any ABA protocol tolerating $\mathcal{A}_t$ is possible iff $n \geq 3t+1$. Thus any ABA protocol designed with $n = 3t+1$ is therefore called as *optimally resilient.* By the seminal result of [34], any ABA protocol, irrespective of the value of n, must have some *non-terminating* runs/executions, where some honest party(ies) may not output any value and thus may not terminate at all. So in any (ϵ, δ)-ABA protocol with non-zero ϵ, the probability of the occurrence of a non-terminating execution is at most ϵ (these type of protocols are called $(1-\epsilon)$-terminating [17,16]). On the other hand in any $(0, \delta)$-ABA protocol, the *probability* of occurrence of a non-terminating execution is *asymptotically zero* (these type of protocols are called *almost-surely terminating*, a term coined by Abraham et. al in [1]).

We now describe the chain of results that has appeared in the literature of ABA. Rabin [58] and Ben-Or [6] presented ABA protocols with $n \geq 8t+1$ and $n \geq 5t+1$ respectively. Since, both these protocols were not *optimally resilient*, researchers have tried to design ABA protocol with *optimal resilience* or close to optimal resilience. In this direction the first attempt is by Bracha [12] who reported an optimally resilient $(0, 0)$-ABA protocol. However, the protocol of Bracha [12] requires exponential ($\Theta(2^n)$) expected time and exponential ($\Theta(2^n)$) communication complexity. Subsequently, Feldman and Micali [30] presented a $(0, 0)$-ABA protocol which runs in constant expected time and requires polynomial communication complexity (they actually extend their BA protocol in synchronous settings [30] to asynchronous settings). However, the ABA protocol of Feldman and Micali [30] is not optimally resilient and requires $4t+1$ parties. So it remained an open question whether there exists an optimally resilient ABA with polynomial running time and communication complexity. Canetti and Rabin [17] answered this question in affirmative and provided an $(\epsilon, 0)$-ABA protocol that offers optimal resilience, *constant* expected running time and polynomial communication complexity. But it is to be noted that so far in the literature there was no optimally resilient $(0, 0)$-ABA protocol with polynomial communication complexity. This long standing open question was resolved by Abraham et. al. [1]. However, the protocol of [1] requires polynomial running time (as opposed to constant expected running time achieved by the ABA protocols of [30,17]). Hence indeed there is an interesting open problem to come up with an optimally resilient $(0, 0)$-ABA with constant expected running time and polynomial communication complexity. In Table 1, we summarize the best known existing ABA protocols.

Over a period of time, the techniques and the design approaches of ABA has evolved spectacularly. In his seminal paper [58], Rabin reduced the problem of ABA to that of a 'common coin'. Specifically, Rabin designed an ABA assuming that the parties have access to a 'common coin' (namely, a common source of randomness). However Rabin did not provide any implementation of common coin. In brief, common coin protocol allows the honest parties to output a common random bit with some probability which we may call as success probability of that common coin protocol. The first ever implementation of common coin was done by Bracha [12]. However, the common coin protocol of [12] is very straight forward. Essentially in Bracha's common coin protocol every party tosses a coin locally and then they hope that they all got the same value; clearly this happens with probability which is exponentially small in

Table 1 Summary of Best Known Existing ABA Protocols

Ref.	Type	Resilience	Communication Complexity (CC)	Expected Running Time (ERT)
[12]	$(0,0)$-ABA	$t < n/3$	$\mathcal{O}(2^n)$	$\mathcal{C} = \mathcal{O}(2^n)$
[30,31]	$(0,0)$-ABA	$t < n/4$	$poly(n)$	$\mathcal{C} = \mathcal{O}(1)$
[17,16]	$(\epsilon,0)$-ABA	$t < n/3$	$poly(n, \frac{1}{\epsilon})$	$\mathcal{C} = \mathcal{O}(1)$
[1]	$(0,0)$-ABA	$t < n/3$	$poly(n)$	$\mathcal{C} = \mathcal{O}(n^2)$

the number of parties, namely $\Theta(2^{-n})$ (so the success probability of Bracha's common coin is $\Theta(2^{-n})$). Consequently the common coin of Bracha [12] while incorporated to design ABA causes the ABA of [12] to run for exponential expected time and also calls for exponential communication complexity. Bracha's design approach of ABA using common coin protocol provided an insightful implication which actually paved the future path for designing efficient ABA protocol with constant expected running time: *The expected running time of ABA is inversely proportional to the success probability of common coin protocol.* The above finding shows a natural direction towards designing efficient common coin protocol with constant success probability in order to construct an efficient ABA with constant expected running time (following the design approach of Bracha).

That is what is exactly achieved by Feldman and Micali [30,31], who are the first to come up with a common coin protocol that has constant success probability in contrast to the exponential success probability of Bracha [12]. Using the new common coin, the ABA of [30,31] follows the same design approach of Bracha and achieves constant expected running time and polynomial communication complexity. At the heart of the common coin protocol of [30] is an efficient Asynchronous Verifiable Secret Sharing (AVSS) protocol. In fact, the essence of [30] is the reduction of the common coin to that of implementing an AVSS protocol. Given an AVSS with n parties, the common coin of [30] requires $\min(n, 3t+1)$ parties for execution. In [31], Feldman et. al. have designed an AVSS with $n = 4t+1$ parties and using the AVSS, they designed an ABA with $4t+1$ parties.

After that the researchers almost followed the same approach of reducing the design of ABA to that of designing AVSS. To design an ABA with optimal resilience i.e $n = 3t+1$, Canetti et. al. [17] have designed an AVSS with $n = 3t+1$ for the first time in literature. As the AVSS had negligible error probability in termination, the resultant ABA of [17] is of type $(\epsilon, 0)$ (in contrast to the ABA of [12,30] which are of type $(0,0)$). Recently, Abraham et. al [1] have reported an weaker variant of AVSS, named as shunning AVSS which was used to designing shunning common coin which is further used to design ABA protocol. The shunning AVSS has no error probability in termination and thus the resultant ABA of [1] is of type $(0,0)$. But the shunning AVSS satisfies all the properties of AVSS only when all the parties including the corrupted ones behave according to the protocol steps. On the other hand, when at least a single corrupted party misbehaves the shunning AVSS ensures at least one *honest* party will shun a *corrupted* party from then onwards. Due to this property of shunning AVSS, the ABA [1] protocol requires $\mathcal{O}(n^2)$ expected running time (in contrast to constant running time of the ABA protocol of [30,17]). A more detailed discussion on the ABA protocols of [17,1] is presented later.

1.3 The Motivation of Our Work

The communication complexity of BA protocol is one of its important parameters. In the literature, a lot of attention has peen paid to improve the communication complexity of BA protocols in synchronous settings (see for example [11,19,25,57,38]). Unfortunately, not too much attention has been paid to design communication efficient ABA protocols with optimal resilience. Though the communication complexity of the known optimally resilient ABA protocols [17,1] is polynomial in n and $\log \frac{1}{\epsilon}$, they involve fairly very high communication complexity. Especially, though the AVSS (and hence ABA) protocol of [17] is a seminal result, it is very much involved and complex in nature. In a real-life distributed network, fast and communication efficient protocols find lot of application. Naturally, designing optimally resilient, communication efficient, fast ABA protocol which runs in constant expected time is an important and interesting problem. Our result in this paper marks a significant progress in this direction. Above all our ABA protocol is reasonably simple.

1.4 Our Contribution

In this article, we present an optimally resilient, $(\epsilon, 0)$-ABA protocol whose *amortized* communication complexity for agreeing on a single bit is $\mathcal{O}(\mathcal{C}n^4 \log \frac{1}{\epsilon})$ bits of private communication[1] as well as A-cast[2], where $\mathcal{C}$ is the *expected running time* of the protocol. Specifically, our ABA protocol requires private communication, as well as A-cast of $\mathcal{O}(\mathcal{C}n^5 \log \frac{1}{\epsilon})$ bits for reaching agreement on $t+1 = \Theta(n)$ bits *concurrently*. Conditioned on the event that our ABA protocol terminates, it does so in constant expected time; i.e., $\mathcal{C} = \mathcal{O}(1)$.

We compare our ABA with the optimally resilient $(\epsilon, 0)$-ABA protocol of [17] which also has constant expected running time; i.e., $\mathcal{C} = \mathcal{O}(1)$. The ABA of [17] *privately* communicates $\mathcal{O}(\mathcal{C}n^{11}(\log \frac{1}{\epsilon})^4)$ bits and A-casts $\mathcal{O}(\mathcal{C}n^{11}(\log \frac{1}{\epsilon})^2 \log n)$ bits. So our ABA achieves a huge gain in communication complexity over the ABA of [17], while keeping all other properties in place.

In another landmark work, Abraham et. al [1] proposed an optimally resilient $(0,0)$-ABA protocol which requires $\mathcal{O}(\mathcal{C}n^6 \log n)$ bits of private communication as well as A-cast. But ABA protocol of Abraham et. al. takes polynomial ($\mathcal{C} = \mathcal{O}(n^2)$) expected time to terminate. Our ABA enjoys the following merits over the ABA of Abraham et. al. [1]:

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

In Table 2, we compare and contrast our ABA protocol with the ABA protocols of [17,1].

Our construction of ABA protocol employs a novel *asynchronous verifiable secret sharing* (AVSS) scheme with $n = 3t + 1$. AVSS is a two phase protocol (Sharing and Reconstruction) carried out among the parties in $\mathcal{P}$ in the presence of $\mathcal{A}_t$. Informally, the goal of the AVSS protocol is to allows a special party in $\mathcal{P}$ called *dealer* to share a secret s among the parties in $\mathcal{P}$ during the sharing phase in a way that would later allow for a unique reconstruction of this secret in the reconstruction phase, while preserving the secrecy of s until the reconstruction phase. Our AVSS protocol is significantly better in terms of communication complexity than the AVSS protocol of [17], while having the same properties. Our AVSS shares multiple secrets *concurrently* and brings forth several advantages of concurrently sharing multiple secrets. We believe that our AVSS can be used in many other applications for improving communication complexity and hence is of independent interest.

As discussed earlier in subsection 1.2, the *common-coin* protocol is a very important building block of ABA protocol. Previously, AVSS sharing single secret was used to design the only known *common-coin* protocol with polynomial communication complexity [31,16]. Informally, in the common coin protocol of [31], each party P_i in $\mathcal{P}$ is asked to act as a dealer and share n random secrets using AVSS. For this P_i invokes n parallel instances of AVSS as a dealer to parallely share n secrets. It is obvious that we can do better if P_i invokes *single* instance of AVSS, which shares n secrets *concurrently*. However, our detailed analysis of the existing common coin protocol shows that the above modification does not lead to a correct solution for common coin protocol. Hence we bring several new modifications to the existing *common-coin* protocol so that it can use our new AVSS (that can share multiple secrets concurrently). As a result, our new common coin protocol is more communication efficient than the existing common coin protocol of [16,17].

1.5 Primitives To be Used

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

Definition 3 (Statistical Asynchronous Weak Secret Sharing (AWSS) [17])
Let (Sh, Rec) be a pair of protocols in which a dealer $D \in \mathcal{P}$ shares a secret s. We say that (Sh, Rec) is a t-resilient statistical AWSS scheme for n parties if all the following hold for every possible behavior of $\mathcal{A}_t$:

– **Termination**: With probability at least $1 - \epsilon$, the following requirements hold:
  1. If D is *honest* then each party will eventually terminate protocol Sh.
  2. If some honest party has terminated protocol Sh, then irrespective of the behavior of D, each honest party will eventually terminate Sh.
  3. If all honest parties have terminated Sh and invoked Rec, then each honest party will eventually terminate Rec.
– **Correctness**: With probability at least $1 - \epsilon$, the following requirements hold:
  1. If D is *honest* then each honest party upon terminating Rec, outputs the shared secret s.

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

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

Ref.	Type	Resilience	Communication Complexity (CC)	Expected Running Time (ERT)
[17]	$(\epsilon, 0)$	$t < \frac{n}{3}$	Private– $\mathcal{O}(\mathcal{C}n^{11}(\log \frac{1}{\epsilon})^4)$ A-cast– $\mathcal{O}(\mathcal{C}n^{11}(\log \frac{1}{\epsilon})^2 \log n)$	$\mathcal{C} = \mathcal{O}(1)$
[1]	$(0, 0)$	$t < \frac{n}{3}$	Private– $\mathcal{O}(\mathcal{C}n^6 \log n)$ A-cast– $\mathcal{O}(\mathcal{C}n^6 \log n)$	$\mathcal{C} = \mathcal{O}(n^2)$
This Article	$(\epsilon, 0)$	$t < \frac{n}{3}$	Private– $\mathcal{O}(\mathcal{C}n^4(\log \frac{1}{\epsilon}))$ A-cast– $\mathcal{O}(\mathcal{C}n^4(\log \frac{1}{\epsilon}))$	$\mathcal{C} = \mathcal{O}(1)$

2. If D is *faulty* and some honest party has terminated Sh, then there exists a *unique* $s' \in \mathbb{F} \cup \{NULL\}$, such that each honest party upon terminating Rec will output *either* s' *or* $NULL$. This property is also called as weak-commitment.

– **Secrecy**: If D is *honest* and no honest party has begun executing protocol Rec, then $\mathcal{A}_t$ has no information about s.

Remark 1 We stress that in the second **Correctness** property, when $s' \neq NULL$, it is possible that some of the honest parties output s', while other honest parties output $NULL$. The adversary $\mathcal{A}_t$ can schedule the messages and hence accordingly decide during the execution of Rec, which parties will output $NULL$.

Definition 4 (Statistical Asynchronous Verifiable Secret Sharing (AVSS) [17])
The **Termination** and **Secrecy** conditions for AVSS are same as in AWSS. The only difference is in the second **Correctness** property, which is *strengthened* as follows:

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

The above definition of AWSS and AVSS can be extended for secret S containing multiple elements (say ℓ with $\ell > 1$) from $\mathbb{F}$.

Definition 5 (A-cast [17]) It is an asynchronous broadcast primitive, which was introduced and elegantly implemented by Bracha [12] with $n = 3t + 1$. Let Π be an asynchronous protocol initiated by a special party (called the sender), having input m (the message to be broadcast). We say that Π is a t-resilient A-cast protocol if the following hold, for every possible behavior of $\mathcal{A}_t$:

– **Termination**:
1. If the sender is honest and all the honest parties participate in the protocol, then each honest party will eventually terminate the protocol.
2. Irrespective of the behavior of the sender, if any honest party terminates the protocol then each honest party will eventually terminate the protocol.

– **Correctness**: If the honest parties terminate the protocol then they do so with a common output m^*. Furthermore, if the sender is honest then $m^* = m$.

Remark 2 Notice that the **Termination** property of A-cast is much *weaker* than the **Termination** property of ABA because for A-cast, it is not required that the honest parties terminate the protocol when the sender is faulty.

For the sake of completeness, we recall Bracha's A-cast protocol from [16] and present it in Fig. 1.

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

Bracha-A-cast($S, \mathcal{P}, M$)

Code for the sender S (with input M): only S executes this code

1. Send message (MSG, M) privately to all the parties.

Code for party P_i: every party in $\mathcal{P}$ executes this code

1. Upon receiving a message (MSG, M), send $(ECHO, M)$ privately to all parties.
2. Upon receiving $n - t$ messages $(ECHO, M')$ that agree on the value of M', send $(READY, M')$ privately to all the parties.
3. Upon receiving $t + 1$ messages $(READY, M')$ that agree on the value of M', send $(READY, M')$ privately to all the parties.
4. Upon receiving $n - t$ messages $(READY, M')$ that agree on the value of M', send (OK, M') privately to all the parties, accept M' as the output message and terminate the protocol.

Theorem 1 *Protocol A-cast privately communicates $\mathcal{O}(\ell n^2)$ bits for an ℓ bit message.*

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

2 A Brief Discussion on the Approaches Used in the ABA Protocols of [17,1] and Current Article

We now briefly discuss the approach used in the optimally resilient ABA protocols of [17], [1] and the current article.

1. The ABA protocol of Canetti et.al [17,16] uses the reduction from AVSS to ABA. Hence they have first designed an AVSS with $n = 3t + 1$. There are well known inherent difficulties in designing AVSS with $n = 3t + 1$ (see [17,16]). To overcome these difficulties, the authors in [17] used the following approach to design their AVSS scheme. They first designed a tool called *Information Checking Protocol* (ICP). Then a protocol called *Asynchronous Recoverable Sharing* (A-RS) was designed using ICP as a black box. Subsequently, using A-RS as a primitive, the authors have designed an AWSS protocol. Then the authors presented a variation of AWSS scheme called *Two & Sum AWSS*. Finally using their *Two & Sum AWSS*, an AVSS scheme was presented. Thus pictorially, the route taken by [17] to design their AVSS scheme is as follows: *ICP* $\rightarrow$ *A-RS* $\rightarrow$ *AWSS* $\rightarrow$ *Two & Sum AWSS* $\rightarrow$ *AVSS*. Since the final AVSS scheme is designed on the top of so many sub-protocols, it is highly communication intensive as well as very much involved. The protocol privately communicates $\mathcal{O}(n^9(\log\frac{1}{\epsilon})^4)$ bits, A-cast $\mathcal{O}(n^9(\log\frac{1}{\epsilon})^2\log(n))$ bits during *sharing phase* and privately communicates $\mathcal{O}(n^6(\log\frac{1}{\epsilon})^3)$ bits, A-cast $\mathcal{O}(n^6(\log\frac{1}{\epsilon})\log(n))$ bits during *reconstruction phase* [3] for sharing a single secret s, where all the honest parties terminate the protocol with probability at least $1 - \epsilon$.
2. The ABA protocol of [1] used the same reduction from AVSS to ABA as in [17], except that the use of AVSS is replaced by a variant of AVSS that the authors called *shunning* (asynchronous) VSS (SVSS), where each party is guaranteed to terminate *almost-surely*. SVSS is a slightly weaker notion of AVSS in the sense that if all the parties behave correctly, then SVSS satisfies all the properties of AVSS without any error. Otherwise it does not satisfy the properties of AVSS but enables some honest party to identify at least one corrupted party, whom the honest party shuns from then onwards. The use of SVSS instead of AVSS in generating common coin causes the ABA of [1] to run for $\mathcal{O}(n^2)$ expected time. The SVSS protocol requires private communication of $\mathcal{O}(n^4\log(n))$ bits and A-cast of $\mathcal{O}(n^4\log(n))$ bits.
3. Our ABA protocol also follows the same reduction from AVSS to ABA as in [17]. In the course of designing our ABA protocol, our first step is to design a communication efficient AVSS protocol. Instead of following a fairly complex route taken by [17] to design an AVSS scheme, we follow a shorter route: *ICP* $\rightarrow$ *AWSS* $\rightarrow$ *AVSS*. Beside this, we significantly improve each of these building blocks by employing new design approaches. Also each of the building blocks deals with multiple secrets concurrently and thus lead to significant gain in communication complexity.
   As mentioned earlier, the existing common coin protocol [31,16] calls AVSS dealing with single secret as a black box. Our detailed analysis of the existing common coin protocol shows that the common coin protocol does not achieve its properties when the invocations of AVSS sharing single secret are replaced by invocations of our AVSS sharing multiple secrets concurrently. Hence, we have modified the existing common coin protocol so that it can use our AVSS sharing multiple secrets as a building block. Together, this lead to our efficient ABA protocol which we believe to be much simpler than the ABA of [17].

3 Organization of the Paper

For the ease of presentation, we divide the paper into two parts. In the first part, our focus is to describe the main ideas used in our AWSS and AVSS protocols. Hence for ease of understanding, we present our AWSS and AVSS scheme sharing *single* secret. By incorporating this AVSS into the existing common coin protocol [31,16], we devise an ABA scheme which allows the parties to agree on a *single* bit and requires private communication as well as A-cast of $\mathcal{O}(n^6(\log\frac{1}{\epsilon}))$ bits. In fact, this ABA scheme was reported in [55].

In the second part of the paper, we extend our AWSS and AVSS scheme to share *multiple* secrets concurrently. As the existing common coin protocol [31,16] can work only with an AVSS scheme which shares a single secret, we show how to modify the common coin protocol of [31,16] and present a new common coin protocol that use our AVSS sharing multiple secrets concurrently. Finally, using this common coin protocol, we

[3] The exact communication complexity analysis of the AVSS (and ABA) scheme of [17] was not done earlier. For the sake of completeness, we carry out the same in **APPENDIX A**.

present our new ABA scheme whose *amortized* communication cost of reaching agreement on a *single* bit is $\mathcal{O}(n^4(\log \frac{1}{\epsilon}))$ bits of private as well as A-cast communication. We then conclude our article with conclusion and open problems.

4 AVSS Scheme for Sharing a Single Secret

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

4.1 Information Checking Protocol (ICP)

The Information Checking Protocol (ICP) is a tool for authenticating messages in the presence of computationally unbounded corrupted parties. The notion of ICP was first introduced by Rabin et.al [60] who have designed an ICP in *synchronous* settings. The ICP of Rabin et. al. was also used as a tool by Canetti et. al. [17] for designing their ABA scheme.

As described in [60,17,21], an ICP is executed among three parties: a *dealer* D, an *intermediary* INT and a *verifier* R. The dealer D hands over a secret value s to INT. At a later stage, INT is required to hand over s to R and convince R that s is indeed the value which INT received from D. The basic definition of ICP involves only a *single* verifier R [60,21,17]. We extend this notion to *multiple* verifiers, where all the n parties in $\mathcal{P}$ act as verifiers. Thus our ICP is executed among three entities: a dealer $D \in \mathcal{P}$, an intermediary $INT \in \mathcal{P}$ and the entire set $\mathcal{P}$ acting as verifiers. This will be later helpful in using ICP as a tool in our AWSS protocol. Moreover, in contrast to the existing ICP protocols that deal with single secret, our ICP can deal with *multiple* secrets *concurrently* and thus achieves better communication complexity than multiple execution of ICP dealing with single secret. Note that, as opposed to the case of a single verifier, when multiple verifiers *simultaneously* participate in ICP, we need to distinguish between synchronity and asynchronity of the network. Our ICP is executed in asynchronous settings and thus we refer it as AICP. As in [60,17], our AICP is also structured into sequence of following three phases:

1. **Generation Phase**: This phase is initiated by D. Here D hands over the secret S containing ℓ elements from $\mathbb{F}$ to *intermediary* INT. In addition, D sends some *authentication information* to INT and some *verification information* to individual verifiers in $\mathcal{P}$.
2. **Verification Phase**: This phase is initiated by INT to acquire an IC Signature on S that will be later accepted by every honest verifiers in $\mathcal{P}$. Depending on the nature of D and/or INT, secret S OR S along with the *authentication information* (which is/are held by INT at the end of **Verification Phase**) will be called as D's IC *signature* on S which is denoted by $ICSig(D, INT, \mathcal{P}, S)$.
3. **Revelation Phase**: This phase is carried out by INT and the verifiers in $\mathcal{P}$. Here INT reveals $ICSig(D, INT, \mathcal{P}, S)$. The verifiers publish their responses after verifying $ICSig(D, INT, \mathcal{P}, S)$ with respect to their verification information. Depending upon the responses by the verifiers, every verifier $P_i \in \mathcal{P}$ either accepts $ICSig(D, INT, \mathcal{P}, S)$ or rejects it. Upon acceptance (resp., rejection), verifier P_i sets $\mathsf{Reveal}_i = S$ (resp., $\mathsf{Reveal}_i = NULL$).

Any AICP should satisfy the following properties (which are almost same as the properties of ICP defined in [17]):

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

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

The Intuition: In Multi-Verifier-AICP, D selects a random polynomial $F(x)$ of degree $\ell + t$, whose first ℓ coefficients are the elements of S and delivers $F(x)$ to INT. In addition, to each individual verifier, D privately gives the value of $F(x)$ at a random *evaluation point* α_i. This distribution by D helps to achieve **AICP-Correctness2** property. Specifically, if D is *honest*, then a *corrupted* INT cannot produce an *incorrect* $F'(x) \neq F(x)$ during **Revelation Phase** without being detected by an *honest* verifier. This is because a corrupted INT will have no information about the evaluation point of an honest verifier and hence with very high probability, $F'(x)$ will not satisfy the evaluation point held by an honest verifier.

The above distribution by D also maintains **AICP-Secrecy** property. The degree of $F(x)$ is $\ell+t$. But only up to t points on $F(x)$ will be disclosed to $\mathcal{A}_t$ through t corrupted verifiers. Therefore $\mathcal{A}_t$ will fall short by ℓ points to *uniquely* interpolate $F(x)$.

But the above distribution alone is not enough to achieve **AICP-Correctness3**. A *corrupted* D might distribute $F(x)$ to INT and value of some other polynomial (different from $F(x)$) to each honest verifier. To avoid this situation, INT and the verifiers interact in *zero knowledge* fashion to check the consistency of $F(x)$ and the value of $F(x)$. The specific details of the zero knowledge, along with other formal steps of protocol Multi-Verifier-AICP are given in Fig. 2.

We now prove the properties of protocol Multi-Verifier-AICP.

Claim If D and INT are honest then D will never A-cast S during Ver.

Proof: Since D is honest, he will send the verification information (α_i, v_i, r_i) to verifier P_i in $\mathcal{P}$. The honest verifiers (at least $2t+1$) will eventually receive the verification information from D and will inform INT by sending `Received-From-D` signal. Hence, the honest INT will eventually construct *ReceivedSet* and will correctly A-cast $(d, B(x), \mathit{ReceivedSet})$ during Ver. So during Ver, D will find $B(\alpha_i) = dv_i + r_i$ for all $P_i \in \mathit{ReceivedSet}$. Thus D will never A-cast S during Ver. □

Lemma 1 (AICP-Correctness1) *If D and INT are honest, then $ICSig(D, INT, \mathcal{P}, S)$ produced by INT during* **Revelation Phase** *will be accepted by each honest verifier.*

Proof: For an honest D, $(F(x), R(x))$ held by honest INT and (α_i, v_i, r_i) held by honest verifier P_i in *ReceivedSet* will satisfy $v_i = F(\alpha_i)$ and $r_i = R(\alpha_i)$. Moreover by previous claim, D will never A-cast S during Ver. Hence $ICSig(D, INT, \mathcal{P}, S) = F(x)$. Now every honest verifier P_i in *ReceivedSet* will A-cast `Accept` during Reveal-Public as **C1** i.e $v_i = F(\alpha_i)$ will hold in protocol Reveal-Public. Since there are at least $t+1$ honest verifiers in *ReceivedSet*, $ICSig(D, INT, \mathcal{P}, S)$ will be accepted by every honest verifier. □

Claim If $(F(x), R(x))$ held by an honest INT and (α_i, v_i, r_i) held by an honest verifier $P_i \in \mathit{ReceivedSet}$ satisfies $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$, then except with probability ϵ, $B(\alpha_i) \neq dv_i + r_i$.

Proof: We first prove that for $(F(x), R(x))$ held by an honest INT and (α_i, v_i, r_i) held by honest verifier $P_i \in \mathit{ReceivedSet}$, there is *only one* non-zero d for which $B(\alpha_i) = dv_i + r_i$, even though $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$. For otherwise, assume there exists another non-zero element $e \neq d$, 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

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

Protocol Multi-Verifier-AICP$(D, INT, \mathcal{P}, S, \epsilon)$

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

1. D picks and sends the following to INT:
   (a) A random degree-$(\ell+t)$ polynomial $F(x)$ over $\mathbb{F}$, such that the lower order ℓ coefficients of $F(x)$ are elements of S.
   (b) A random degree-$(\ell+t)$ polynomial $R(x)$ over $\mathbb{F}$.
2. D privately sends the following to every verifier P_i:
   (a) (α_i, v_i, r_i), where $\alpha_i \in \mathbb{F} - \{0\}$ is random (all α_i's are distinct).
   (b) $v_i = F(\alpha_i)$ and $r_i = R(\alpha_i)$.
   The polynomials $F(x), R(x)$ is called **authentication information**, while for $i = 1, \ldots, n$, the values (α_i, v_i, r_i) are called **verification information**.

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

1. For $i = 1, \ldots, n$, verifier P_i sends a `Received-From-D` signal to INT after receiving (α_i, v_i, r_i) from D.
2. Upon receiving `Received-From-D` from $2t+1$ verifiers, INT creates a set $\mathit{ReceivedSet} = \{P_i \mid INT$ received `Received-From-D` signal from $P_i\}$ (clearly $|\mathit{ReceivedSet}| = 2t+1$). INT then chooses a random $d \in \mathbb{F} \setminus \{0\}$ and A-casts $(d, B(x), \mathit{ReceivedSet})$, where $B(x) = dF(x) + R(x)$.
3. D checks $dv_i + r_i \overset{?}{=} B(\alpha_i)$ for every $P_i \in \mathit{ReceivedSet}$. If D finds any inconsistency, he A-casts S. Otherwise D A-casts `OK`.
4. Upon receiving the A-cast of D, INT sets
   (a) $ICSig(D, INT, \mathcal{P}, S) = F(x)$ if `OK` is received from the A-cast of D.
   (b) $ICSig(D, INT, \mathcal{P}, S) = S$ if S is received from the A-cast of D.

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

1. INT A-casts $ICSig(D, INT, \mathcal{P}, S)$.
2. On receiving $ICSig(D, INT, \mathcal{P}, S)$ from the A-cast of INT, verifier $P_i \in \mathit{ReceivedSet}$ who indeed sent Received-From-D to INT during Ver, A-cast `Accept` in the following conditions.
   (a) If $ICSig(D, INT, \mathcal{P}, S) = S$ and the S A-casted by D during Ver is same as $ICSig(D, INT, \mathcal{P}, S)$.
   (b) If $ICSig(D, INT, \mathcal{P}, S) = F(x)$ and one of the following holds.
      i. **C1:** $v_i = F(\alpha_i)$; OR
      ii. **C2:** $B(\alpha_i) \neq dv_i + r_i$, where $B(x)$ was A-casted by INT during Ver.
   Otherwise, P_i A-cast `Reject`.

Local Computation (By Every Verifier in $\mathcal{P}$): If at least $(t+1)$ verifiers from *ReceivedSet* have A-casted `Accept` during Reveal-Public then accept $ICSig(D, INT, \mathcal{P}, S)$ and set Reveal$_i = S$. Else reject $ICSig(D, INT, \mathcal{P}, S)$ and set Reveal$_i = NULL$.

that $(d-e)F(\alpha_i) = (d-e)v_i$ or $F(\alpha_i) = v_i$, which is a contradiction. Now since d is randomly chosen by honest INT *only after* D handed over $(F(x), R(x))$ to INT and (α_i, v_i, r_i) to every *honest* $P_i \in ReceivedSet$, a corrupted D has to guess d in advance during Gen to make sure that $B(\alpha_i) = dv_i + r_i$ holds. However, D can guess d with probability at most $\frac{1}{|\mathbb{F}|-1} \approx \epsilon$. Hence only with probability at most ϵ, corrupted D can make $B(\alpha_i) = dv_i + r_i$, even though $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$. □

Lemma 2 (AICP-Correctness2) *If an honest INT holds an $ICSig(D, INT, \mathcal{P}, S)$ at the end of* **Verification Phase***, then $ICSig(D, INT, \mathcal{P}, S)$ will be accepted in* **Revelation Phase** *by each honest verifier, except with probability ϵ.*

Proof: We prove the lemma considering D to be corrupted because when D is honest, the lemma follows from Lemma 1. Now the proof can be divided into following two cases:

1. $ICSig(D, INT, \mathcal{P}, S) = S$: In this case, the lemma holds trivially, without any error.
2. $ICSig(D, INT, \mathcal{P}, S) = F(x)$: Here, we show that except with probability ϵ, each honest verifier in *ReceivedSet* will A-cast `Accept` during Reveal-Public. So let P_i be an honest verifier in *receivedSet*. We now have the following cases depending on the relation that holds between the information held by INT (i.e $(F(x), R(x))$) and information held by the honest P_i (i.e (α_i, v_i, r_i)):
   (a) $F(\alpha_i) = v_i$: Here P_i will A-cast `Accept` without any error probability as **C1** (i.e $F(\alpha_i) = v_i$) will hold.
   (b) $F(\alpha_i) \neq v_i$ *and* $R(\alpha_i) = r_i$: Here P_i will A-cast `Accept` without any error probability, as **C2** (i.e $B(\alpha_i) \neq dv_i + r_i$) will hold.
   (c) $F(\alpha_i) \neq v_i$ *and* $R(\alpha_i) \neq r_i$: Here P_i will A-cast `Accept` except with probability ϵ, as **C2** will hold from the previous claim.

   As there are at least $t+1$ honest verifiers in *Received Set* who will A-cast `Accept` during Reveal-Public, each honest verifier will accept $ICSig(D, INT, \mathcal{P}, S)$ during Reveal-Public, except with probability ϵ.

   This completes the proof of the lemma. □

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

Proof: Here again we have two cases. If $ICSig(D, INT, \mathcal{P}, S) = S$, then the lemma holds trivially from the protocol steps. So we now prove the lemma when $ICSig(D, INT, \mathcal{P}, S) = F(x)$. Here a corrupted INT can produce $S' \neq S$ by A-casting $F'(x) \neq F(x)$ during Reveal-Public such that the lower order ℓ coefficients of $F'(x)$ is S'. We now claim that if INT does so, then except with probability ϵ, every honest verifier P_i in *ReceivedSet* will A-cast `Reject` during Reveal-Public. In the following, we show that the conditions for which an honest verifier P_i in *ReceivedSet* would A-cast `Accept` are either impossible or may happen with probability ϵ:

1. $F'(\alpha_i) = v_i$: Since P_i and D are honest, corrupted INT has no information about α_i, v_i. Hence the probability that INT can ensure $F'(\alpha_i) = v_i = F(\alpha_i)$ is same as INT correctly guesses α_i, which is at most $\frac{\ell+t}{|\mathbb{F}-1|} \approx 2^{-\Omega(\kappa)} \approx \epsilon$ (since $F(x)$ and $F'(x)$ can have same value at most at $\ell + t$ values of x.).
2. $B(\alpha_i) \neq dv_i + r_i$: This case is never possible since D is honest. If $B(\alpha_i) \neq dv_i + r_i$ corresponding to $P_i \in ReceivedSet$, then honest D would have A-casted S during Ver.

Thus if a corrupted INT produces $F'(x) \neq F(x)$ during Ver, then except with probability ϵ, every (honest) verifier P_i (at least $t+1$) in *ReceivedSet* will A-cast `Reject` during Reveal-Public. Hence $ICSig(D, INT, \mathcal{P}, S')$ will be rejected. □

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

Proof: During Distr, $\mathcal{A}_t$ will know t distinct points on $F(x)$ and $R(x)$. Since both $F(x)$ and $R(x)$ are of degree-$(\ell+t)$, the lower order ℓ coefficients of both $F(x)$ and $R(x)$ are information theoretically secure. During Ver, $\mathcal{A}_t$ will know d and $dF(x) + R(x)$. Since both $F(x)$ and $R(x)$ are random and independent of each other, it still holds that the lower order ℓ coefficients of $F(x)$ is information theoretically secure. Also, if D and INT are honest, then D will never broadcast S during Ver. Hence the lemma. □

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

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

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

1. *"P_i sends $ICSig(P_i, P_j, \mathcal{P}, S)$ to P_j" to mean that P_i as a dealer D, executes Gen$(P_i, P_j, \mathcal{P}, S, \epsilon)$, considering P_j as INT.*
2. *"P_i receives $ICSig(P_j, P_i, \mathcal{P}, S)$ from P_j" to mean that P_i as INT has completed Ver$(P_j, P_i, \mathcal{P}, S, \epsilon)$ with the help of the verifiers in $\mathcal{P}$ and obtained $ICSig(P_j, P_i, \mathcal{P}, S)$ from P_j, where P_j is the dealer.*
3. *"P_i reveals $ICSig(P_j, P_i, \mathcal{P}, S)$" to means P_i as INT executes Reveal-Public$(P_j, P_i, \mathcal{P}, S, \epsilon)$ along with the participation of the verifiers in $\mathcal{P}$.*
4. *"P_k completes revelation $ICSig(P_j, P_i, \mathcal{P}, S)$ with Reveal$_k = \overline{S}$" to mean that P_k as a verifier has successfully completed Reveal-Public$(P_j, P_i, \mathcal{P}, S, \epsilon)$ with Reveal$_k = \overline{S}$.*

4.2 AWSS Scheme for Sharing a Single Secret

We now present a novel AWSS scheme with $n = 3t + 1$, consisting of sub-protocols AWSS-Share and AWSS-Rec. While AWSS-Share allows D to share a secret s, AWSS-Rec enables public reconstruction of either D's shared secret or $NULL$. Moreover, if D is *corrupted*, then s can be either from $\mathbb{F}$ or it can be $NULL$ (in a sense explained in the sequel). We follow the general idea of [7, 21, 42, 36, 45] in synchronous settings for sharing the secret s with a symmetric bivariate polynomial $F(x, y)$ of degree-t in x and y, where each party P_i gets the univariate polynomial $f_i(x) = F(x, i)$. In particular, our AWSS scheme is somewhat inspired by the WSS scheme of [21] in synchronous settings, with several new ideas added to it, to deal with the asynchrony of the network.

High Level Description of AWSS-Share: First D hands over $ICSig(D, INT, \mathcal{P}, f_i(j))$ for every $j = 1, \ldots, n$ to P_i. This step implicitly implies that P_i will receive $f_i(x)$ from D. After receiving these IC signatures from D, every pair of parties (P_i, P_j) exchange their IC signature on their common value, namely $f_i(j) = f_j(i) = F(i, j)$. Then D, in conjunction with all other parties, perform a sequences of communications and computations. As a result of this, at the end of AWSS-Share, every party agrees on a set of $2t + 1$ parties, called $WCORE$, such that every party $P_j \in WCORE$ is *IC-committed* to $f_j(0)$ using $f_j(x)$ to a set of $2t + 1$ parties, called as OKP_j. P_j is *IC-committed* to $f_j(0)$ using $f_j(x)$ among the parties in OKP_j only when every $P_k \in OKP_j$ received (a) $ICSig(D, P_k, \mathcal{P}, f_k(j))$ and (b) $ICSig(P_j, P_k, \mathcal{P}, f_j(k))$ and ensures $f_k(j) = f_j(k)$ (this should ideally hold due to the selection and distribution of symmetric bivariate polynomial). In some sense, we may view this as every $P_j \in WCORE$ is attempting to commit his received (from D) polynomial $f_j(x)$ among the parties in OKP_j (by giving his *IC Signature* on one point of $f_j(x)$ to each party) and the parties in OKP_j allowing him to do so after verifying that they have got D's IC signature on the same value of $f_j(x)$. We will show that later in reconstruction phase, every honest P_j's (in $WCORE$) *IC-commitment* will be reconstructed correctly irrespective of whether D is honest or corrupted. Moreover, a corrupted P_j's *IC-commitment* will be reconstructed correctly when D is honest. But on the other hand, a corrupted P_j's *IC-commitment* can be reconstructed to any value when D is corrupted. These properties are at the heart of our AWSS protocol.

Achieving the agreement (among the parties) on $WCORE$ and corresponding OKP_js is a bit tricky in asynchronous network. Even though these sets are constructed on the basis of information that are A-casted, parties may end up with different versions of $WCORE$ and OKP_j's while attempting to generate them locally, due to the asynchronous nature of the network. We solve this problem by asking D to construct $WCORE$ and OKP_js based on A-casted information and then ask D to A-cast the same. After receiving $WCORE$ and OKP_js from the A-cast of D, individual parties ensure the validity of these sets by receiving the same A-cast using which D would have formed these sets. A similar approach was used in the protocols of [1]. Protocol AWSS-Share is formally presented in Fig. 3.

Before moving into the discussion and description of AWSS-Rec, we now define what we call as D's AWSS-commitment.

Remark 3 (D's **AWSS-commitment***)* We say that D is *AWSS-committed* to a secret $s \in \mathbb{F}$ in AWSS-Share if there is a unique degree-t univariate polynomial $f(x)$ such that $f(0) = s$ and every *honest* P_i in $WCORE$ receives $f(i)$ from D and *IC-commits* to $f(i)$ among the parties in OKP_i. Otherwise, we say that D has

Fig. 3 Sharing Phase of AWSS Scheme for Sharing a Single Secret s with $n = 3t + 1$

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

DISTRIBUTION: CODE FOR $D$ – Only $D$ executes this code.
    1. Select a random, symmetric bivariate polynomial $F(x, y)$ of degree-$t$ in $x$ and $y$, such that $F(0, 0) = s$.
       For $i = 1, \ldots, n$, let $f_i(x) = F(x, i)$.
    2. For $i = 1, \ldots, n$, send $ICSig(D, P_i, \mathcal{P}, f_i(j))$ to $P_i$ for each $j = 1, \ldots, n$. For this, $D$ initiates $n^2$ instances of Gen, each with an error parameter of $\epsilon' = \frac{\epsilon}{n^2}$.
VERIFICATION: CODE FOR $P_i$ – Every party including $D$ executes this code.
    1. Wait to receive $ICSig(D, P_i, \mathcal{P}, f_i(j))$ for each $j = 1, \ldots, n$ from $D$.
    2. Check if $(f_i(1), \ldots, f_i(n))$ defines degree-$t$ polynomial. If yes then send $ICSig(P_i, P_j, \mathcal{P}, f_i(j))$ to $P_j$ for all $j = 1, \ldots, n$.
    3. If $ICSig(P_j, P_i, \mathcal{P}, f_j(i))$ is received from $P_j$ and if $f_i(j) = f_j(i)$, then A-cast OK$(P_i, P_j)$.
WCORE CONSTRUCTION : CODE FOR $D$ – Only $D$ executes this code.
    1. For each $P_j$, build a set $OKP_j = \{P_k | D$ receives OK$(P_k, P_j)$ from the A-cast of $P_k\}$.
       When $|OKP_j| = 2t + 1$, then $P_j$'s IC-commitment on $f_j(0)$ is over (or we may say that $P_j$ is IC-committed to $f_j(0)$) and add $P_j$ in $WCORE$ (which was initially empty).
    2. Wait until $|WCORE| = 2t + 1$. Then A-cast $WCORE$ and $OKP_j$ for all $P_j \in WCORE$.
WCORE VERIFICATION & AGREEMENT ON WCORE : CODE FOR $P_i$ — Every party executes this code
    1. Wait to obtain $WCORE$ and $OKP_j$ for all $P_j \in WCORE$ from $D$'s A-cast, such that $|WCORE| = 2t + 1$ and $|OKP_j| = 2t + 1$ for each $P_j \in WCORE$.
    2. Wait to receive OK$(P_k, P_j)$ for all $P_k \in OKP_j$ and $P_j \in WCORE$. After receiving all these OKs, accept the $WCORE$ and $OKP_j$'s received from $D$ and terminate AWSS-Share.
```

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

Our discussion in the sequel will show that for a corrupted D, irrespective of the behavior of the corrupted parties, either D's *AWSS-committed* secret s (which belongs to $\mathbb{F} \cup \{NULL\}$) or $NULL$ will be reconstructed by each honest party in AVSS-Rec.

High Level Idea of AWSS-Rec: In AWSS-Rec, the parties in $WCORE$ and corresponding OKP_j's are used in order to reconstruct D's AWSS-committed secret. Precisely, for every $P_j \in WCORE$, P_j's *IC-commitment* $(f_j(0))$ is reconstructed by asking every party $P_k \in OKP_j$ to reveal $ICSig(D, P_k, \mathcal{P}, f_k(j))$ and $ICSig(P_j, P_k, \mathcal{P}, f_j(k))$ such that $f_k(j) = f_j(k)$ holds. Since there are at least $t + 1$ honest parties in OKP_j, eventually at least $t + 1$ $f_j(k)$'s and $f_k(j)$'s will be revealed with which $f_j(x)$ and thus $f_j(0)$ will be reconstructed. Then $f_j(0)$'s are used to construct the univariate polynomial $f_0(x)$ that is committed by D during AWSS-Share.

Asking $P_k \in OKP_j$ to reveal D's IC signature ensures that when D is *honest*, then even for a *corrupted* $P_j \in WCORE$, the reconstructed polynomial $f_j(x)$ will be same as the one handed over by D to P_j in sharing phase (that is a corrupted P_j's IC-commitment $f_j(0)$ will be reconstructed correctly). This helps our AWSS protocol to satisfy **Correctness 1** property of AWSS. Now asking P_k in OKP_j to reveal P_j's signature ensures that even if D is *corrupted*, for an *honest* $P_j \in WCORE$, the reconstructed polynomial $f_j(x)$ will be same as the one received by P_j from D in AWSS-Share (that is an honest P_j's IC-commitment $f_j(0)$ will be reconstructed correctly even though D is corrupted). This helps to ensure **Correctness 2** property. Summing up, when at least one of D and P_j is honest, P_j's *IC-commitment* (i.e $f_j(0)$) will be revealed properly. But when both D and P_j are corrupted, P_j's *IC-Commitment* can be revealed as any $\overline{f_j(0)}$ which may or not be equal to $f_j(0)$. It is the later property that makes our protocol to qualify as a AWSS protocol rather than a AVSS protocol. Protocol AWSS-Rec is formally given in Fig. 4.

We now prove the properties of our AWSS scheme.

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

PROOF:

- **Termination 1:** When D is honest then eventually all honest parties will receive desired IC signatures from D and will also eventually exchange IC signatures on their common values and will A-cast OK for each other. Hence every honest P_j will eventually complete his *IC-commitment* on $f_j(0)$ with at least $2t+1$ honest parties in OKP_j. So D will eventually include $2t+1$ parties in $WCORE$ (of which at least $t + 1$ are honest) and A-cast the same. Now by the property of A-cast, each honest party will eventually listen $WCORE$ from the A-cast of D. Finally,

Fig. 4 Reconstruction Phase of AWSS Scheme for Sharing a Single Secret s with $n = 3t + 1$

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

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

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

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

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

since honest D had included P_j in $WCORE$ after receiving the OK signals from the parties in OKP_j's, each honest party will also receive the same and will eventually terminate AWSS-Share.

– **Termination 2:** If an honest P_i has terminated AWSS-Share, then he must have received $WCORE$ and OKP_j's from the A-cast of D and verified their validity. By properties of A-cast, each honest party will also receive the same and will eventually terminate AWSS-Share.
– **Termination 3:** Since each instance of AICP is executed with an error parameter $\epsilon' = \frac{\epsilon}{n^2}$, if P_i (acting as INT) is honest and has received an IC signature, then IC signature produced by P_i during Reveal-Public will be accepted by every honest verifier without any error probability when D is honest (by **AICP-Correctness1** i.e Lemma 1) and except with probability ϵ' when D is corrupted (by **AICP-Correctness2** i.e Lemma 2). Since for every $P_j \in WCORE$, $|OKP_j| = 2t + 1$, there are at least $t + 1$ honest parties in OKP_j and each of them may be present in $ValidP_j$ except with probability ϵ'. Thus except with probability $n^2\epsilon' = \epsilon$, P_j's *IC-commitment* will be reconstructed for all $P_j \in WCORE$. Thus except with probability ϵ, each *honest* verifier will terminate AWSS-Rec after executing remaining steps of [LOCAL COMPUTATION]. □

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

PROOF: We have to consider the case when D is honest. The proof follows from the secrecy of our AICP protocol and properties of symmetric bivariate polynomial of degree-t in x and y [20]. Specifically, without loss of generality, let $P_1, \ldots, P_t$ be under the control of $\mathcal{A}_t$. So during the execution of AWSS-Share, $\mathcal{A}_t$ will know $f_1(x), \ldots, f_t(x)$ and t points on $f_{t+1}(x), \ldots, f_n(x)$. However, $\mathcal{A}_t$ still lacks one more point to uniquely interpolate $F(x, y)$. Hence, $s = F(0, 0)$ will be information theoretically secure. □

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

PROOF:

– **Correctness 1:** Here we have to consider the case when D is *honest*. We show that D's *AWSS-commit ment* will be reconstructed correctly except with probability ϵ. We prove the lemma by showing that when D is *honest*, P_j's *IC-commitment* $f_j(0)$ will be correctly reconstructed with probability at least $(1 - \frac{\epsilon}{n})$ for every $P_j \in WCORE$, irrespective of whether P_j is honest or corrupted. Consequently, as $|WCORE| = 2t + 1$, all the honest parties will reconstruct $f_0(x) = F(x, 0)$ and hence the secret $s = f_0(0)$ with probability at least $(1 - (2t+1)\frac{\epsilon'}{n}) \approx (1 - \epsilon)$. So we consider the following two cases:
  1. Consider an honest P_j in $WCORE$. From **AICP-Correctness3** (Lemma 3), a corrupted $P_k \in OKP_j$ can successfully produce $ICSig(P_j, P_k, \mathcal{P}, \overline{f_j(k)})$ such that $\overline{f_j(k)} \neq f_j(k)$, with probability at most ϵ'. As there can be at most t corrupted parties in $ValidP_j$, except with probability $t\epsilon' = \frac{\epsilon}{n}$, the value $\overline{f_j(k)}$ is same as $f_j(k)$ for all $P_k \in ValidP_j$. Hence honest P_j's *IC-commitment* $f_j(0)$ will be correctly reconstructed with probability at least $(1 - \frac{\epsilon}{n})$.
  2. Consider a corrupted P_j in $WCORE$. Now a corrupted $P_k \in OKP_j$ will be able to produce $ICSig(D, P_k, \mathcal{P}, \overline{f_k(j)})$ such that $\overline{f_k(j)} \neq f_k(j)$, with probability ϵ' due to **AICP-Correctness3**. Thus except with probability $t\epsilon' = \frac{\epsilon}{n}$, corresponding to each corrupted $P_j \in WCORE$, the parties in $ValidP_j$ have produced correct points on $f_j(x)$.

- **Correctness 2:** Here we consider the case, when D is *corrupted*. Now there are two cases: (a) D's *AWSS-committed* secret s belongs to $\mathbb{F}$; (b) D's *AWSS-committed* secret s is $NULL$. Whatever may be case, we show that except with probability ϵ, each honest party will either reconstruct s or $NULL$.
  1. We first consider the case when $s \in \mathbb{F}$. This implies that the $f_j(0)$ values received by the honest parties in $WCORE$ lies on a degree-t polynomial $f_0(x)$. Moreover every honest P_j in $WCORE$ is *IC-committed* to $f_j(0)$. We now show that in AWSS-Rec, *IC-commitment* of all honest parties in $WCORE$ will be reconstructed correctly with probability at least $(1-\epsilon)$. So let P_j be an honest party in $WCORE$. Now from **AICP-Correctness3**, a corrupted $P_k \in OKP_j$ can not produce $ICSig(P_j, P_k, \mathcal{P}, \overline{f_j(k)})$ such that $\overline{f_j(k)} \neq f_j(k)$ with probability at least $(1-\epsilon')$. Hence for honest P_j, $f_j(x)$ and thus $f_j(0)$ will be reconstructed correctly with probability at least $(1-(t+1)\epsilon') \approx (1-\frac{\epsilon}{n})$. As there are at least $t+1$ honest parties in $WCORE$, the probability that the above event happens for all honest parties is $(1-\epsilon)$.
     But for a *corrupted* P_j in $WCORE$, P_j's *IC-commitment* can be revealed to any value $\overline{f_j}(0)$. This is because a corrupted $P_k \in OKP_j$ can produce a valid signature of P_j on any $\overline{f_j(k)}$ as well as a valid signature of D (who is corrupted as well) on $\overline{f_k(j)} = \overline{f_j(k)}$. Also the adversary can delay the messages such that the values of corrupted $P_k \in OKP_j$ are revealed (to parties) before the values of honest parties in OKP_j. Now if reconstructed $\overline{f_j(0)} = f_j(0)$ for all corrupted $P_j \in WCORE$, then s will be reconstructed. Otherwise, $NULL$ will be reconstructed. However, since for all the honest parties of $WCORE$, *IC-commitment* will be reconstructed correctly with probability at least $(1-\epsilon)$ (who in turn define $f_0(x)$), no other secret (other than s) can be reconstructed.
  2. We next consider the second case when D's AWSS-committed secret is $NULL$. This implies that the points $(j, f_j(0))$ corresponding to honest P_j's in $WCORE$ do not define a unique degree-t polynomial. It is easy to see that in this case, irrespective of the behavior of the corrupted parties $NULL$ will be reconstructed. This is because the points $f_j(0)$ corresponding to each honest $P_j \in WCORE$ will be reconstructed correctly except with probability ϵ (following the argument given in previous case).

□

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

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

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

Proof: The proof follows from Lemma 5, Lemma 6 and Lemma 7.

Notation 3 *In our AVSS scheme, we will invoke AWSS-Share as AWSS-Share($D, \mathcal{P}, f(x), \epsilon$) to mean that D commits to $f(x)$ in AWSS-Share. Essentially here D is asked to choose a symmetric bivariate polynomial $F(x,y)$ of degree-t in x and y, where $F(x,0) = f(x)$ holds. D then tries to give $F(x,i)$ and hence $F(0,i) = f(i)$ to party P_i. Similarly, AWSS-Rec will be invoked as AWSS-Rec($D, \mathcal{P}, f(x), \epsilon$), which allows the parties to reconstruct either $f(x)$ or $NULL$, except with an error probability of ϵ.* □

4.3 Our AVSS Scheme for Sharing a Single Secret

In this section, we present our novel AVSS scheme consisting of sub-protocols AVSS-Share and AVSS-Rec. While AVSS-Share allows D to share a secret s, AVSS-Rec enables public reconstruction of D's shared secret. Moreover, if D is *corrupted*, then s can be either from $\mathbb{F}$ or it can be $NULL$ (in a sense explained in the sequel).

High Level Idea of AVSS-Share: To design AVSS-Share, we use the general approach of [60,59,51,36,45] used in synchronous settings for designing VSS using WSS as a black box. The high level idea of AVSS-Share is as follows: D selects a symmetric bivariate polynomial $F(x, y)$ of degree-t in x and y, such that $F(0, 0) = s$ and sends $f_i(x) = F(x, i)$ to party P_i. Now the parties communicate with each other to perform what we say *commitment upon verification*. Here each party P_i is asked to *commit* the polynomial $f_i(x)$, that he has received from D. However, P_i is allowed to commit $f_i(x)$, only after the parties have *verified* that they have received same points on $f_i(x)$ from D as well as P_i. More formally, to achieve *commitment upon verification*, party P_i, acting as a dealer, shares his polynomial $f_i(x)$ by initiating an instance of AWSS-Share with a degree-t symmetric bivariate polynomial $Q^{P_i}(x, y)$, such that $Q^{P_i}(x, 0) = f_i(x)$. Since party P_j receives $q_j^{P_i}(x) = Q^{P_i}(x, j)$ from P_i as part of AWSS-Share, he can check whether $q_j^{P_i}(0) \overset{?}{=} f_j(i)$, as ideally $q_j^{P_i}(0) = f_i(j) = f_j(i)$ should hold in case of honest D, P_i and P_j. A party P_j participates in the remaining steps of the instance of AWSS-Share where P_i is the dealer, only if $q_j^{P_i}(0) = f_j(i)$ holds. Once *commitment upon verification* is over, the parties want to agree on a set of at least $2t + 1$ parties, denoted as $VCORE$, such that for every P_j in $VCORE$, P_j's instance of AWSS-Share terminates with a $WCORE$ set, denoted as $WCORE^{P_j}$ and $|VCORE \cap WCORE^{P_j}| \geq 2t + 1$. Informally, this means that each party $P_j \in VCORE$ has 'successfully' committed his polynomial $f_j(x)$ to at least $2t + 1$ parties in $VCORE$, who have verified that they have received correct points on $f_j(x)$. We will refer this commitment as P_j's *AWSS-commitment* on $f_j(x)$. It should be noted that *AWSS-Commitment* is strictly stronger commitment than *IC-commitment* that was enforced in AWSS-Share. These two commitments can be distinguished by the facts that when both D and P_j are corrupted (a) *AWSS-commitment* ensures that reconstruction of *AWSS-commitment* can not be changed to some other value other than $NULL$ and (b) *IC-commitment* can not ensure the same. The agreement on $VCORE$ and corresponding $WCORE^{P_j}$'s is achieved using a mechanism, similar to the one used in AWSS-Share for achieving agreement on $WCORE$ and corresponding OK sets. Protocol AVSS-Share is formally presented in Fig. 5.

Remark 4 (D's **AVSS-commitment***)* We say that D has AVSS-committed $s \in \mathbb{F}$ in AVSS-Share if there is a unique symmetric bivariate polynomial $F(x, y)$ of degree-t in x and y, such that $F(0, 0) = s$ and every *honest* P_i in $VCORE$ receives $f_i(x) = F(x, i)$ from D and *AWSS-commits* $f_i(0)$ using $f_i(x)$ among the parties in $WCORE^{P_i}$. Otherwise, we say that D has committed $NULL$. Notice that the above condition implies that there exist a unique degree-t univariate polynomial $f(x)(= f_0(x) = F(x, 0))$ such that $f(0) = s$ and every honest $P_i \in VCORE$ receives $f(i)(= f_0(i) = f_i(0))$ from D. **The value $f(i)$ is referred as i^{th} share of** s. An honest D always AVSS-commits s from $\mathbb{F}$ as he always chooses a proper symmetric bivariate polynomial $F(x, y)$ and properly distributes $f_i(x) = F(x, i)$ to party P_i. But AVSS-Share can *not* ensure that corrupted D also commits $s \in \mathbb{F}$. When a corrupted D commits

Fig. 5 Sharing Phase of AVSS Scheme for Sharing a Single Secret s with $n = 3t + 1$

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

DISTRIBUTION: CODE FOR D — Only D executes this code

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

COMMITMENT UPON VERIFICATION: CODE FOR P_i — Every party, including D executes this code

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

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

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

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

1. Wait to receive $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$ from D's A-cast.
2. Wait to terminate AWSS-Share^{P_j} corresponding to every P_j in $VCORE$.
3. Wait to receive OK(P_m, P_k) for every $P_k \in WCORE^{P_j}$ and every $P_m \in OKP_k^{P_j}$, corresponding to every $P_j \in VCORE$.
4. Accept $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$ and terminate AVSS-Share.

$NULL$, the $f_i(x)$ polynomials of the honest parties in $VCORE$ do not define a symmetric bivariate polynomial of degree-t in x and y. This further implies that there will be an honest pair (P_γ, P_δ) in $VCORE$ such that $f_\gamma(\delta) \neq f_\delta(\gamma)$. When $s = NULL$, we say that D's AVSS-committed secret s is not *meaningful*.

In our following discussion, we show that irrespective of whether s is chosen from $\mathbb{F}$ or it is $NULL$, s will be reconstructed in AVSS-Rec by *every* honest party, except with probability ϵ.

High Level Idea of AVSS-Rec: In AVSS-Rec, D's AVSS-committed secret is recovered with the help of the parties in $VCORE$ and $WCORE^{P_j}$'s. Specifically, in the reconstruction phase, for every $P_j \in VCORE$, *AWSS-commitment* on $f_j(x)$ is revealed by reconstructing it with the help of the parties in $WCORE^{P_j}$. This is done by executing an instance of AWSS-Rec with the parties in $WCORE^{P_j}$. This results in the reconstruction of either $f_j(x)$ or NULL depending on whether P_j is honest or corrupted. Since $|VCORE| \geq 2t+1$, for (at least $t+1$) honest parties, $f_j(x)$'s will be recovered correctly. Now with the $f_j(x)$'s, $F(x,y)$ will be reconstructed. The formal details of AVSS-Rec is given in Fig. 6.

Fig. 6 Reconstruction Phase of AVSS Scheme for Sharing a Single Secret s with $n = 3t+1$

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

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

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

We now prove the properties of protocol AVSS-Share and AVSS-Rec.

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

PROOF:

– **Termination 1:** Notice that in AVSS-Share, D keeps on adding new parties to $WCORE^{P_j}$ in instance AWSS-Share^{P_j}, even after $WCORE^{P_j}$ contains $2t+1$ parties. So if D is honest, then corresponding to every honest P_j, $2t+1$ honest parties will be eventually included in $WCORE^{P_j}$. Now eventually at least $2t+1$ honest parties will be included in $VCORE$, such that $|VCORE \cap WCORE^{P_j}| \geq 2t+1$ for each $P_j \in VCORE$. Now from similar argument given in **Termination 1** of Lemma 5, all honest parties will eventually agree on $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ and will terminate AVSS-Share.
– **Termination 2:** If some honest party has terminated AVSS-Share then it implies that he has received $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$ from the A-cast of D and checked their validity. So by the property of A-cast, every other honest party will also eventually do the same and terminate AVSS-Share.
– **Termination 3:** Follows from the fact that corresponding to each $P_j \in VCORE$, every honest P_i will eventually terminate AWSS-Rec^{P_j} (from **Termination 3** of Lemma 5), except with an error probability of ϵ'. As there are at least $t+1$ honest parties in $VCORE$, AWSS-Rec corresponding to all the honest parties will terminate with probability at least $(1-(t+1)\epsilon') \approx (1-\epsilon)$. □

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

PROOF:

– **Correctness 1:** We have to consider the case when D is honest. If D is *honest* then we prove that except with probability ϵ', for every $P_i \in FINAL$, P_i's *AWSS-Commitment* will be reconstructed correctly. In other words, AWSS-Rec^{P_i} will reconstruct $\overline{f_i(x)}$ which is same as $f_i(x)$ selected by honest D. For every *honest* $P_i \in FINAL$ this is trivially true and follows from the **Correctness1** of our AWSS scheme. We have to prove the above statement for a corrupted $P_i \in FINAL$. If a corrupted P_i belongs to $FINAL$, it implies that AWSS-Rec^{P_i} is successful (i.e., the output is a symmetric bivariate polynomial of degree-t in x and y) and AWSS-Share^{P_i} had terminated during AVSS-Share, such that $|VCORE \cap WCORE^{P_i}| \geq 2t+1$. The above statements have the following implications:

1. P_i must have given consistent polynomials to the honest parties in $WCORE^{P_i}$ (during AWSS-Share^{P_i}) such that they induce valid symmetric bivariate polynomial of degree-t in x and y (see **Correctness 2** of Lemma 7).
2. P_i must have agreed with the honest parties of $WCORE^{P_i}$ with respect to the common values given by D. This means that as a part of AWSS-Share^{P_i}, P_i handed over $q_j^{P_i}(x)$ to an honest P_j (in $WCORE^{P_i}$) satisfying $f_j(i) = q_j^{P_i}(0)$.

The above two implications together further imply that P_i must have committed (to the honest parties in $WCORE^{P_i}$ whose count is at least $t+1$) some bivariate polynomial $Q^{P_i}(x,y)$ satisfying $Q^{P_i}(x,0) = f_i(x)$. Thus if AWSS-Rec^{P_i} is successful, then except with probability ϵ', $\overline{Q^{P_i}(x,y)} = Q^{P_i}(x,y)$ and hence $\overline{f_i(x)} = f_i(x)$. Since D is honest, $\overline{f_i(x)}$'s corresponding to $P_i \in FINAL$ will define $\overline{F(x,y)} = F(x,y)$. In the worst case, there can be at most t corrupted parties in $FINAL$ and hence except with probability $\epsilon' t \approx \epsilon$, $\overline{f_i(x)}$'s corresponding to each $P_i \in FINAL$ will define $\overline{F(x,y)} = F(x,y)$ and thus $s = \overline{F(0,0)} = F(0,0)$ will be recovered.

– **Correctness 2:** Here we have to consider the case when D is corrupted. Now there are two cases: (a) D's *AVSS-committed* secret s belongs to $\mathbb{F}$; (b) D's *AVSS-committed* secret s is $NULL$. Whatever may be case, we show that except with probability ϵ, each honest party will reconstruct s.
  1. Let $s = NULL$. Now for every honest $P_i \in VCORE$, AWSS-Rec^{P_i} will reconstruct $\overline{f_i(x)}$ correctly and thus P_i will be added to $FINAL$, except with error probability ϵ'. Consequently since there are at least $t+1$ honest parties in $VCORE$, all the honest parties from $VCORE$ will be added to $FINAL$ except with error probability of $n\epsilon' = \epsilon$. Now irrespective of the remaining (corrupted) parties included in $FINAL$, the consistency checking (i.e., $\overline{f_\gamma(\delta)} \overset{?}{=} \overline{f_\delta(\gamma)}$) will fail for some pair (P_γ, P_δ) of honest parties and thus $NULL$ will be reconstructed.
  2. On the other hand, let $s \in \mathbb{F}$ (i.e *meaningful*) and $s = F(0,0)$. This means that $F(x,y)$ is defined by the $f_i(x)$'s of the honest parties in $VCORE$. This case now completely resembles with the case when D is honest and hence the proof follows from the proof of **Correctness 1**. □

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

PROOF: We have to consider the case when D is honest. Without loss of generality, let $P_1, \ldots, P_t$ be under the control of $\mathcal{A}_t$. It is easy to see that through out AVSS-Share, $\mathcal{A}_t$ will know $f_1(x), \ldots, f_t(x)$ and t points on $f_{t+1}(x), \ldots, f_n(x)$. However, from the property of symmetric polynomial of degree-t in x and y [20], the adversary $\mathcal{A}_t$ will lack one more point on $F(x,y)$ to uniquely interpolate $F(x,y)$. Hence $s = F(0,0)$ will be information theoretically secure. □

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

PROOF: The proof follows from Lemma 8 and the fact that in AVSS-Share, there can be $\Theta(n)$ instances of AWSS-Share, each executed with an error parameter of $\epsilon' = \frac{\epsilon}{n}$. Moreover, in AVSS-Rec there can be $\Theta(n)$ instances of AWSS-Rec. □

Remark 5 Protocol AVSS-Share does not force *corrupted* D to AVSS-commit some *meaningful* secret (i.e., an element from $\mathbb{F}$). Hence, the secret s, AVSS-committed by a *corrupted* D can be either from $\mathbb{F}$ or $NULL$. We may assume that if D's AVSS-committed secret is $NULL$, then D has AVSS-committed some predefined value $s^* \in \mathbb{F}$, which is known publicly. Hence in AVSS-Rec, whenever $NULL$ is reconstructed, every honest party replaces $NULL$ by the predefined secret s^*. Interpreting this way, we say that our AVSS scheme allows D to *AVSS-commit* secret from $\mathbb{F}$.

5 Existing Common Coin Protocol

Here we first recall the definition of common coin and then recall the construction of common coin protocol following the description of [16]. The common coin protocol invokes many instances of AVSS scheme. In the following description of our common coin protocol, we replace the AVSS scheme of [16] by our AVSS scheme presented in Section 4.3. We start with the definition of common coin protocol.

Definition 6 (Common Coin [16]) Let π be an asynchronous protocol, where each party has local random input and binary output. We say that π is a $(1-\epsilon)$-terminating, t-resilient common coin protocol if the following requirements hold for every adversary $\mathcal{A}_t$:

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

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

Now in the second stage, after terminating the AVSS-Share instances of all the secrets attached to some P_i, party P_j is sure that a fixed (yet unknown) value is attached to P_i. Once P_j is assured that values have been attached to enough number of parties, he participates in AVSS-Rec instances of the relevant secrets. This process of ensuring that there are enough parties that are attached with values is the core idea of the protocol. Once all the relevant secrets are reconstructed, each party locally computes his binary output based on the reconstructed secrets, in a way described in the protocol presented in the sequel. Protocol Common-Coin is presented in Fig. 7.

Let E be an event, defined as follows: All invocations of AVSS scheme have been terminated properly. That is, if an honest party has terminated AVSS-Share, then a value, say s' is fixed. All honest parties will terminate the corresponding invocation of AVSS-Rec with output s'. Moreover if dealer D is honest then s' is D's shared secret. It is easy to see that event E occurs with probability at least $1 - n^2\epsilon' = 1 - \epsilon$.

We now state the following lemmas which are more or less identical to the Lemmas 5.28-5.31 presented in [16]. For the sake of completeness, we recall all of them with proofs.

Fig. 7 Existing Common Coin Protocol

Protocol Common-Coin(ϵ)

CODE FOR P_i: — Every party executes this code

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

Lemma 13 ([16]) *All honest parties terminate Protocol Common-Coin in constant time.*

PROOF: First we show that every *honest* party P_i will A-cast "`Reconstruct Enabled`". By the termination property of our AVSS protocol, every honest party will eventually terminate all the instances of AVSS-Share of every other honest party. As there are at least $n-t$ honest parties, for every honest party P_i, $\mathcal{T}_i$ will eventually contain at least $t+1$ parties (in fact $n-t$ parties) and thus P_i will eventually A-cast "`Attach` T_i `to` P_i". So eventually, P_i will receive "`Attach` T_j `to` P_j" from every *honest* P_j. Now since every party P_k that is included in $\mathcal{T}_j$ will be eventually included in $\mathcal{T}_i$ (follows from the termination property of AVSS protocol), $T_j \subseteq \mathcal{T}_i$ will hold good. Therefore, every honest P_j will be eventually included in $\mathcal{A}_i$. Thus for an honest P_i, $\mathcal{A}_i$ will eventually be of size $n-t$ and hence P_i will A-cast "P_i `Accepts` A_i". Now following the similar argument as above, we can show that for an honest P_i, $\mathcal{S}_i$ will eventually be of size $n-t$ and hence P_i will A-cast "`Reconstruct Enabled`".

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

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

Lemma 14 ([16]) *In protocol Common-Coin, once some honest party P_j receives "`Attach` T_i `to` P_i" from the A-cast of P_i and includes P_i in $\mathcal{A}_j$, a unique value V_i is fixed such that*

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

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

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

Lemma 15 ([16]) *Once an honest party A-cast "`Reconstruct Enabled`", there exists a set M such that:*

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

Proof: Let P_i be the *first honest party* to A-cast "`Reconstruct Enabled`". Then let $M = \{P_k \mid P_k$ belongs to $A_l's$ of at least $t+1$ $P_l's$ who belongs to $\mathcal{S}_i$ when P_i A-casted `Reconstruct Enabled` $\}$. We now show the parties in M satisfies the properties mentioned in the lemma.

It is clear that $M \subseteq H_i$. Thus party P_i has received "`Attach` T_j `to` P_j" from the A-cast of every $P_j \in M$. As P_i is assumed to be honest here, the first part of the lemma is asserted.

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

Now we prove the third part of the lemma i.e $|M| \geq \frac{n}{3}$. A counting argument is used for this purpose. Let $m = |\mathcal{S}_i|$ at the time P_i A-casted "`Reconstruct Enabled`". So we have $m \geq n - t$. Now consider an $n \times n$ table Λ_i (relative to party P_i), whose l^{th} row and k^{th} column contains 1 for $k, l \in \{1, \ldots, n\}$ iff the following hold:

1. P_i has received "P_l `Accepts` A_l" from A-cast of P_l and included P_l in $\mathcal{S}_i$ before A-casting "`Reconstruct Enabled`" AND
2. $P_k \in A_l$

The remaining entries (if any) of Λ_i are left blank. Then M is the set of parties P_k such that k^{th} column in Λ_i contains 1 at least at $t+1$ positions. Notice that each row of Λ_i contains 1 at $n - t$ positions. Thus Λ_i contains 1 at $m(n-t)$ positions.

Let q denote the minimum number of columns in Λ_i that contain 1 at least at $t+1$ positions. We will show that $q \geq \frac{n}{3}$. The worst distribution of 1 entries in Λ_i is letting q columns to contain all 1 entries and letting each of the remaining $n - q$ columns to contain 1 at t locations. This distribution requires Λ_i to contain 1 at no more than $qm + (n-q)t$ positions. But we have already shown that Λ_i contains 1 at $m(n-t)$ positions. So we have

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

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

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

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

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

PROOF: By Lemma 14, for every party P_i that is included in $\mathcal{A}_j$ of some honest party P_j, there exists some fixed (yet unknown) value V_i that is distributed uniformly and independently over $[0,\ldots,u]$ and with probability $1-\frac{\epsilon}{n}$ all honest parties will associate V_i with P_i. Consequently, as there are n^2 instances of AVSS-Rec, each with an error probability of $\epsilon' = \frac{\epsilon}{n^2}$, with probability at least $1-n^2\epsilon' = (1-\epsilon)$, all honest parties will agree on the value associated with each one of the parties. Now we consider two cases:

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

Hence the lemma. □

Theorem 4 ([16]) *Protocol Common-Coin is a $(1-\epsilon)$-terminating, t-resilient common coin protocol for $n = 3t+1$ parties for every $0 < \epsilon \leq 0.2$.*

PROOF: The **Termination** property follows from Lemma 13. The **Correctness** property follows from Lemma 14, Lemma 15 and Lemma 16. □

Due to the use of efficient AVSS scheme (as presented in our article) in the place of relatively inefficient AVSS protocol of [16], protocol Common-Coin provides better communication complexity than the common coin protocol presented in [16].

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

PROOF: Easy, as Common-Coin executes at most n^2 instances of AVSS-Share and AVSS-Rec, each with an error parameter of $\frac{\epsilon}{n^2}$. □

6 Existing Voting Protocol

The Voting protocol is another requirement for the construction of ABA protocol. In a Voting protocol, every party has a single bit as input. Roughly, Voting protocol tries to find out whether there is a detectable majority for some value among the inputs of the parties. Here we recall the Voting protocol called Vote from [16].

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

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

We will show that:

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

The Vote protocol consists of three stages, having similar structure. In the first stage, each party A-casts his input value, waits to receive $n-t$ A-casts of other parties, and sets his `vote` to the majority value among these inputs. In the second phase, each party A-casts his `vote` (along with the identities of the $n-t$ parties whose A-casted inputs were used to compute `vote`), waits to receive $n-t$ A-casts of other `vote`s that are consistent with the A-casted inputs of the first phase, and sets his `re-vote` to the majority value among these `vote`s. In the third phase, each party A-casts his `re-vote` along with the identities of the $n-t$ parties whose A-casted `vote`s were used to compute the `re-vote`, and waits to complete $n-t$ A-casts of other re-votes that are consistent with the consistent votes of second phase. Now if all the consistent votes received by a party agree on a value, σ, then the party outputs $(\sigma, 2)$. Otherwise, if all the consistent re-votes received by the party agree on a value, σ, then the party outputs $(\sigma, 1)$. Otherwise, the party outputs $(\Lambda, 0)$. Protocol Vote is presented formally in Fig. 8. In the protocol, we assume party P_i has input bit x_i.

Fig. 8 Existing Vote Protocol

Protocol Vote()

CODE FOR P_i: — Every party executes this code

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

We now recall the proofs for the properties of protocol Vote from [16].

Lemma 17 ([16]) *All the honest parties terminate protocol Vote in constant time.*

PROOF: Every honest party P_i will eventually receive $(\texttt{input}, P_j, x_j)$ from the A-cast of every honest P_j. Thus every honest P_i will eventually have $|\mathcal{A}_i| = n-t$ and will A-cast $(\texttt{vote}, P_i, A_i, a_i)$. Now every honest party P_i will eventually receive $(\texttt{vote}, P_j, A_j, a_j)$ from the A-cast of every honest P_j. Thus every honest P_i will eventually have $|\mathcal{B}_i| = n-t$ and will A-cast $(\texttt{re-vote}, P_i, B_i, b_i)$. Now every honest party P_i will eventually receive $(\texttt{re-vote}, P_j, B_j, b_j)$ from the A-cast of every honest P_j. Thus every honest P_i will eventually have $|C_i| = n-t$. Consequently, every honest P_i will terminate the protocol in constant time. □

Lemma 18 ([16]) *If all the honest parties have the same input σ, then all the honest parties will eventually output $(\sigma, 2)$ in protocol Vote.*

PROOF: Consider an honest party P_i. If all the honest parties have same input σ, then at most t (corrupted) parties may A-cast $\overline{\sigma}$ as their input. Therefore, it is easy to see every P_k (irrespective of whether honest or corrupted), who is included in $\mathcal{B}_i$ must have A-casted his vote $b_k = \sigma$. Hence honest P_i will output $(\sigma, 2)$. □

Lemma 19 ([16]) *If some honest party outputs $(\sigma, 2)$, then every other honest party will eventually output either $(\sigma, 2)$ or $(\sigma, 1)$ in protocol Vote.*

PROOF: Let an *honest* party P_i outputs $(\sigma, 2)$. This implies that all the parties $P_j \in B_i$ had A-casted the same vote $a_j = \sigma$. As the size of B_i is $n-t = 2t+1$, it implies that for every other *honest* P_j, it holds that $|B_i \cap B_j| \geq t+1$. This means that every other *honest* P_j is bound to A-cast re-vote b_i as σ. Hence every other honest party will eventually output either $(\sigma, 2)$ or $(\sigma, 1)$. □

Lemma 20 ([16]) *If some honest party outputs $(\sigma, 1)$ and no honest party outputs $(\sigma, 2)$ then every other honest party will eventually output either $(\sigma, 1)$ or $(\Lambda, 0)$ in protocol Vote.*

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

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

PROOF: In protocol Vote, each party P_i A-casts A_i and B_i, each containing the identity of $n-t = 2t+1$ parties. Since the identity of each party can be represented by $\log n$ bits, protocol Vote involves A-cast of $\mathcal{O}(n^2 \log n)$ bits. □

7 Efficient ABA Protocol for Single Bit

Once we have an efficient Common Coin protocol and Vote protocol, we can design an efficient ABA protocol using the approach of [16]. The ABA protocol proceeds in iterations where in each iteration every party computes a 'modified input' value. In the first iteration the 'modified input' of party P_i is nothing but the private input bit x_i. In each iteration, every party executes two protocols *sequentially*: Vote and Common-Coin. That is protocol Common-Coin is executed only after the termination of Vote. If a party outputs $\{(\sigma, 2), (\sigma, 1)\}$ in Vote protocol, then he sets his 'modified input' for next iteration to σ, irrespective of the value which is going to be output in Common-Coin. Otherwise, he sets his 'modified input' for next iteration to be the output of Common-Coin protocol which is invoked by all the honest parties in each iteration irrespective of whether the output of Common-Coin is used or not. Once a party outputs $(\sigma, 2)$, he A-casts σ and once he receives $t+1$ A-cast for σ, he terminates the ABA protocol with σ as final output. The protocol is formally presented in Fig. 9.

Fig. 9 Efficient ABA Protocol for Single Bit.

Protocol ABA(ϵ)

Code for P_i: Every party executes this code

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

We now state the following lemmas which are more or less identical to the Lemmas 5.36-5.39 presented in [16]. For the sake of completeness, we recall all of them with proofs.

Lemma 21 ([16]) *In protocol ABA if all honest parties have input σ, then all honest parties terminate and output σ.*

Proof: If all honest parties have input σ, then by Lemma 18 every honest party will output $(y_1, m_1) = (\sigma, 2)$ upon termination of Vote and consequently A-cast (Terminate with σ) in the first iteration. Therefore every honest party will eventually receive $n - t$ A-cast of (Terminate with σ). Hence every honest party will terminate ABA with σ as output. □

Lemma 22 ([16]) *In protocol ABA, if an honest party terminates with output σ, then all honest parties will eventually terminate with output σ.*

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

As all the honest parties will eventually A-cast (Terminate with σ), every honest party will receive $n-t$ A-casts of (Terminate with σ) and at most t A-casts of (Terminate with $\overline{\sigma}$). Therefore every honest party will eventually output σ. □

Lemma 23 ([16]) *If all honest parties have initiated and completed some iteration k, then with probability at least $\frac{1}{4}$ all honest parties have same value for 'modified input' v_{k+1}.*

Proof: We have two cases here:

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

prevent agreement. Hence with probability at least $\frac{1}{4}$, all the honest parties will set $v_{k+1} = \sigma$. □

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

Lemma 24 ([16]) *Conditioned on the event C, all honest parties terminate protocol ABA in constant expected time.*

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

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

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

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

Lemma 25 ([16]) $Prob(C) \geq (1 - \epsilon)$.

Proof: We have

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

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

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

The above equation implies that $Prob(C) \geq (1 - \epsilon)$. □
Summing up, we have the following theorem.

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

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

8 Efficient ABA Protocol for Multiple Bits

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

In real-life applications typically ABA protocols are invoked on *long messages* (whose size can be in gigabytes) rather than on single bit. Even in asynchronous multiparty computation (AMPC) [8,16,4,52], where typically lot of ABA invocations are required, many of the invocations can be parallelized and optimized to a single

[4] Here MB stands for multiple bits.

invocation with a long message. Hence ABA protocols with long message are very relevant to many real life situations. All existing protocols for ABA [58,6,12,30, 31,17,16,1,55] are designed for single bit message. A naive approach to design ABA for $\ell > 1$ bit message is to parallelize ℓ invocations of existing ABA protocols dealing with single bit. This approach requires a communication complexity that is ℓ times the communication complexity of the existing protocols for single bit and hence is inefficient. In this article, we provide a far better way to design an ABA with multiple bits. For ℓ bits message with $\ell \geq t+1$, we may break the message in to blocks of $t+1$ bits and invoke one instance of our ABA-MB for each one of the $t+1$ blocks.

To design our protocol ABA-MB, we first extend our AWSS and AVSS scheme to share $\ell \geq 1$ secrets *simultaneously.* Our AWSS and AVSS scheme sharing ℓ secrets simultaneously involves less communication complexity than ℓ parallel invocations of our AWSS and AVSS scheme sharing *single* secret.

8.1 AWSS Scheme for Sharing Multiple Secrets

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

Notice that we could have executed protocol AWSS-Share ℓ times parallely, each sharing individual elements of S. However, from Lemma 8 this would incur a private communication of $\mathcal{O}(\ell n^3 \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(\ell n^3 \log \frac{1}{\epsilon})$ bits. On the other hand, AWSS-MS-Share shares all elements of S concurrently, requiring a private communication of $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits. Thus for sufficiently large ℓ, the communication complexity of AWSS-MS-Share is less than what would have been required by ℓ parallel executions of AWSS-Share. Similarly, protocol AWSS-MS-Rec reconstructs all the ℓ secrets simultaneously, incurring A-cast communication of $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits.

The Intuition: The high level idea of protocol AWSS-MS-Share is similar to AWSS-Share. For each $s^l, l = 1, \ldots, \ell$, the dealer D selects a random symmetric bivariate polynomial $F^l(x, y)$ of degree-t in x and y, where $F^l(0, 0) = s^l$ and gives his IC-signature on $f_i^l(1), \ldots, f_i^l(n)$ to party P_i, for $i = 1, \ldots, n$. For this, D can execute n instances of AICP, one for each $f_i^l(j)$, for $j = 1, \ldots, n$ (this approach was used in AWSS-Share). However, this would require a total of ℓn instances of AICP (each dealing with a *single* secret) to be executed by D for every party P_i. Clearly, this would require a private communication and A-cast of $\mathcal{O}(\ell n^2 \log \frac{1}{\epsilon})$ bits. Instead of this, a better solution would be to ask D to execute n instances of AICP, where in the j^{th} instance, D gives his IC-signature *collectively* on $(f_i^1(j), f_i^2(j), \ldots, f_i^l(j))$ to party P_i. This requires private communication and A-cast of $\mathcal{O}((\ell n + n^2) \log \frac{1}{\epsilon})$ bits.

Next each party P_i tries to *IC-commit* $(f_i^1(0), \ldots, f_i^\ell(0))$ simultaneously. For this, every pair of parties P_i and P_j privately exchange $(f_i^1(j), \ldots, f_i^\ell(j))$ and $(f_j^1(i), \ldots, f_j^\ell(i))$, along with their respective IC signature on these values. Again notice that P_i and P_j pass on their IC-signature *collectively* on $(f_i^1(j), \ldots, f_i^\ell(j))$ and $(f_j^1(i), \ldots, f_j^\ell(i))$ respectively. Next the parties pair-wise check whether $f_i^l(j) = f_j^l(i)$ *for all* $l = 1, \ldots, \ell$ and if so they A-cast OK signal. After this, the remaining steps (like $WCORE$ construction, agreement on $WCORE$, etc) are same as in AWSS-Share. So essentially, the difference between AWSS-Share and AWSS-MS-Share is the way the parties give their IC-signature and the conditions required for A-casting OK signal. Protocol AWSS-MS-Share is formally given in Fig. 10.

Remark 6 (D's **AWSS-commitment***)* In AWSS-MS-Share, we say that D is AWSS-committed to $S = (s^1, \ldots, s^\ell) \in \mathbb{F}^\ell$ if for every $l = 1, \ldots, \ell$ there is a unique degree-t polynomial $f^l(x)$ such that $f^l(0) = s^l$ and every *honest* P_i in $WCORE$ receives $f^l(i)$ from D and *IC-commits* $f^l(i)$ among the parties in OKP_i. Otherwise, we say that D is AWSS-committed to $NULL$. An honest D always AWSS-commits $S \in \mathbb{F}^\ell$ as in this case $f^l(x) = f_0^l(x) = F^l(x, 0)$, where $F^l(x, y)$ is the symmetric bivariate polynomial of degree-t chosen by D. But AWSS-MS-Share can *not* ensure that corrupted D also AWSS-commits $S \in \mathbb{F}^\ell$.

Protocol AWSS-MS-Rec is a straightforward extension of protocol AWSS-Rec and is given in Fig. 11.

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

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

PROOF: The proof follows from the fact that in AWSS-MS-Share and AWSS-MS-Rec, there are $\mathcal{O}(n^2)$ instances

[5] Here MS stands for multiple secrets

Fig. 10 Sharing Phase of AWSS Scheme for Sharing S Containing $\ell \geq 1$ Secrets

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

DISTRIBUTION: CODE FOR $D$ – Only $D$ executes this code.
    1. For $l = 1, \ldots, \ell$, select a random, symmetric bivariate polynomial $F^l(x, y)$ of degree-$t$ in $x$ and $y$ such that $F^l(0, 0) = s^l$. Let $f_i^l(x) = F^l(x, i)$, for $l = 1, \ldots, \ell$.
    2. For $i = 1, \ldots, n$, send $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \ldots, f_i^\ell(j)))$ for each $j = 1, \ldots, n$ to $P_i$. For this, $D$ initiates $n^2$ instances of Gen, each with an error parameter of $\epsilon' = \frac{\epsilon}{n^2}$.
VERIFICATION: CODE FOR $P_i$ – Every party including $D$ executes this code.
    1. Wait to receive $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \ldots, f_i^\ell(j)))$ for $j = 1, \ldots, n$ from $D$.
    2. Check if $(f_i^l(1), \ldots, f_i^l(n))$ defines degree-$t$ polynomial for every $l = 1, \ldots, \ell$. If yes then send $ICSig(P_i, P_j, \mathcal{P}, (f_i^1(j), \ldots, f_i^\ell(j)))$ to $P_j$ for all $j = 1, \ldots, n$.
    3. If $ICSig(P_j, P_i, \mathcal{P}, (f_j^1(i), \ldots, f_j^\ell(i)))$ is received from $P_j$ and if $f_j^l(i) = f_i^l(j)$ for all $l = 1, \ldots, \ell$, then A-cast OK$(P_i, P_j)$.
WCORE CONSTRUCTION : CODE FOR $D$ – Only $D$ executes this code.
    1. For each $P_j$, build a set $OKP_j = \{P_i | D$ receives OK$(P_i, P_j)$ from the A-cast of $P_i\}$. When $|OKP_j| = 2t + 1$, then $P_j$'s IC-commitment on $(f_j^1(0), \ldots, f_j^\ell(0))$ is over (or we may say that $P_j$ is IC-committed to $(f_j^1(0), \ldots, f_j^\ell(0))$) and add $P_j$ in $WCORE$ (which is initially empty).
    2. Wait until $|WCORE| = 2t + 1$. Then A-cast $WCORE$ and $OKP_j$ for all $P_j \in WCORE$.
WCORE VERIFICATION & AGREEMENT ON WCORE : CODE FOR $P_i$ — Every party executes this code
    1. Wait to obtain $WCORE$ and $OKP_j$ for all $P_j \in WCORE$ from $D$'s A-cast, such that $|WCORE| = 2t+1$ and $|OKP_j| = 2t + 1$ for each $P_j \in WCORE$.
    2. Wait to receive OK$(P_k, P_j)$ for all $P_k \in OKP_j$ and $P_j \in WCORE$. After receiving all these OKs, accept the $WCORE$ and $OKP_j$'s received from $D$ and terminate AWSS-MS-Share.
```

Fig. 11 Reconstruction Phase of AWSS Scheme for Sharing S Containing ℓ Secrets

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

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

of our AICP, each dealing with ℓ values and having an error parameter of $\epsilon' = \frac{\epsilon}{n^2}$. □

Notation 4 *We will invoke AWSS-MS-Share as AWSS-MS-Share($D, \mathcal{P}, (f^1(x), \ldots, f^\ell(x)), \epsilon$) where D is asked to choose symmetric bivariate polynomials $F^1(x, y), \ldots, F^\ell(x, y)$, each of degree-$t$ in x and y, such that $F^l(x, 0) = f^l(x)$ holds for $l = 1, \ldots, \ell$. D then tries to give $F^l(x, i)$ and hence $F^l(0, i) = f^l(i)$ to party P_i, for $l = 1, \ldots, \ell$. Similarly, AWSS-MS-Rec will be invoked as AWSS-MS-Rec($D, \mathcal{P}, (f^1(x), \ldots, f^\ell(x))$, ϵ). This will lead to the public reconstruction of either $(f^1(x), \ldots, f^\ell(x))$ or $NULL$.*

8.2 AVSS Scheme for Sharing Multiple Secrets

We now extend protocol AVSS-Share and AVSS-Rec to AVSS-MS-Share and AVSS-MS-Rec respectively [6]. Protocol AVSS-MS-Share allows $D \in \mathcal{P}$ to concurrently share a secret $S = (s^1 \ldots s^\ell)$, containing ℓ elements. Moreover, if D is corrupted then either $S \in \mathbb{F}^\ell$, where each element of S belongs to $\mathbb{F}$ or $S = NULL$ (in a sense explained in the sequel). Protocol AVSS-MS-Rec allows the parties in $\mathcal{P}$ to reconstruct S.

A simple approach for sharing ℓ secrets would be to execute AVSS-Share ℓ times parallely, each sharing a single secret. From Lemma 12, this naive approach would require a private communication and A-cast of $\mathcal{O}(\ell n^4 \log \frac{1}{\epsilon})$ bits. On the other hand, protocol AVSS-MS-Share shares all elements of S concurrently, requiring a private communication and A-cast of $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$ bits. Thus for sufficiently large ℓ, the com-

[6] Here MS stands for multiple secrets

munication complexity of AVSS-MS-Share is less than what would have been required by ℓ parallel executions of AVSS-Share. Similarly, protocol AVSS-MS-Rec reconstructs all the ℓ secrets simultaneously, incurring A-cast communication of $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$ bits.

The Intuition: The high level idea of AVSS-MS-Share is similar to AVSS-Share. Specifically, for each $s^l \in S$, the dealer D selects a symmetric bivariate polynomial $F^l(x,y)$ of degree-t in x and y, such that $F^l(0,0) = s^l$ and sends $f_i^l(x) = F^l(x,i)$ to party P_i. Then each party P_i is asked to *AWSS-commit* his received polynomials $f_i^1(x), \ldots, f_i^\ell(x)$. However, instead of executing ℓ instances of AWSS-Share, one for committing each $f_i^l(x)$, party P_i executes a *single* instance of AWSS-MS-Share to commit $f_i^1(x), \ldots, f_i^\ell(x)$ simultaneously. It is this step, which leads to the reduction in the communication complexity of AVSS-MS-Share. The remaining steps like $VCORE$ construction, agreement on $VCORE$, etc are similar to protocol AVSS-Share. Protocol AVSS-MS-Share is formally presented in Fig. 12.

Remark 7 (D's **AVSS-commitment)** We say that D has AVSS-committed $S = (s^1, \ldots, s^\ell) \in \mathbb{F}^\ell$ in AVSS-Share if for every $l = 1, \ldots, \ell$ there is a unique degree-t symmetric bivariate polynomial $F^l(x,y)$ such that $F^l(0,0) = s^l$ and every *honest* P_i in $VCORE$ receives $f_i^l(x) = F^l(x,i)$ from D and *AWSS-commits* $f_i^l(0)$ using $f_i^l(x)$ among the parties in $WCORE^{P_i}$. Otherwise, we say that D has committed $NULL$. Notice that the above condition implies that for $l = 1, \ldots, \ell$ there exist a unique degree-t univariate polynomial $f^l(x) (= f_0^l(x) = F^l(x,0))$ such that $f^l(0) = s^l$ and every honest $P_i \in VCORE$ receives $f^l(i) (= f_0^l(i) = f_i^l(0))$ from D. **The value $f^l(i)$ is referred as i^{th} share of s^l.** An honest D always commits s^l from $\mathbb{F}$ as he always chooses a proper symmetric bivariate polynomial $F^l(x,y)$ and properly distributes $f_i^l(x) = F^l(x,i)$ to party P_i. But AVSS-Share can *not* ensure that corrupted D also commits $s^l \in \mathbb{F}$ for all l. When a corrupted D commits $NULL$, the $f_i^l(x)$ polynomials of the honest parties in $VCORE$ do not define a symmetric bivariate polynomial of degree-t in x and y for at least one l. This further implies that there will be an honest pair (P_γ, P_δ) in $VCORE$ such that $f_\gamma^l(\delta) \neq f_\delta^l(\gamma)$. If S belongs to $\mathbb{F}^\ell$, then it is considered as *meaningful.*

Protocol AVSS-MS-Rec is a straightforward extension of protocol AVSS-Rec and is given in Fig. 13.

We do not provide the proof of the properties of protocols AVSS-MS-Share and AVSS-MS-Rec, as it may cause repetition of the proofs provided for protocols AVSS-MS-Share and AVSS-MS-Rec. For the sake of completeness, we state the following theorem on the communication complexity of AVSS-MS-Share and AVSS-MS-Rec.

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

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

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

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

COMMITMENT UPON VERIFICATION: CODE FOR P_i — Every party in $\mathcal{P}$, including D, executes this code.

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

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

1. If AWSS-Share^{P_j} is terminated, then denote corresponding $WCORE$ and OKP_k sets by $WCORE^{P_j}$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$. Add P_j in a set $VCORE$ (initially empty).
2. Keep updating $VCORE$, $WCORE^{P_j}$ and corresponding $OKP_k^{P_j}$'s for every $P_j \in VCORE$ upon receiving new A-casts of the form OK(.,.) (during AWSS-Share^{P_j}s), until for at least $2t+1$ $P_j \in VCORE$, the condition $|VCORE \cap WCORE^{P_j}| \geq 2t+1$ is satisfied. Exclude every other $P_j \in VCORE$ not satisfying the above condition.
3. A-cast $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$. Conclude that each $P_j \in VCORE$ is *AWSS-committed* to $(f_j^1(x), \ldots, f_j^\ell(x))$.

VCORE VERIFICATION & AGREEMENT ON VCORE : CODE FOR P_i — Every party in $\mathcal{P}$, including D, executes this code.

1. Wait to receive $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$ from D's A-cast, such that each of the sets are of size at least $2t+1$ and $|VCORE \cap WCORE^{P_j}| \geq 2t+1$.
2. Wait to terminate AWSS-MS-Share^{P_j} corresponding to every P_j in $VCORE$.
3. For every $P_j \in VCORE$, wait to receive OK(P_m, P_k) for every $P_m \in OKP_k^{P_j}$ and $P_k \in WCORE^{P_j}$. Then accept $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$ and terminate AVSS-MS-Share.

Theorem 9 (AVSS-MS-Communication Complexity) *Protocol AVSS-MS-Share incurs a private commu-*

Fig. 13 Reconstruction Phase of AVSS Scheme for Sharing Secret S Containing ℓ Elements

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

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

1. For every $P_j \in VCORE$, reconstruct $P_j$'s AWSS-commitment on $(f_j^1(x), \ldots, f_j^\ell(x))$ as follows:
   (a) Participate in AWSS-MS-Rec($P_j, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x)), \epsilon'$) with $WCORE^{P_j}$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$, where $\epsilon' = \frac{\epsilon}{n}$. We call this instance of AWSS-MS-Rec as AWSS-MS-Rec$^{P_j}$.
   (b) Wait for the termination of AWSS-MS-Rec$^{P_j}$ with output either $(\overline{Q^{(P_j,1)}(x,y)}, \ldots, \overline{Q^{(P_j,\ell)}(x,y)})$ or $NULL$.
2. Wait for the reconstruction of $P_j$'s AWSS-commitment for every $P_j \in VCORE$.
3. Add $P_j \in VCORE$ to $FINAL$ (which is initially empty) if AWSS-Rec$^{P_j}$ gives a non-$NULL$ output. Now for $P_j \in FINAL$, assign $\overline{f_j^l(x)} = \overline{Q^{(P_j,l)}(x,0)}$, for $l = 1, \ldots, \ell$.
4. For $l = 1, \ldots, \ell$, do the following:
   (a) For every pair $(P_\gamma, P_\delta) \in FINAL$ check $\overline{f_\gamma^l(\delta)} \overset{?}{=} \overline{f_\delta^l(\gamma)}$. If the test passes for every pair of parties then recover $\overline{F^l(x,y)}$ using $\overline{f_j^l(x)}$'s corresponding to each $P_j \in FINAL$ and reconstruct $\overline{s^l} = \overline{F^l(0,0)}$. Else reconstruct $\overline{s^l} = NULL$.
5. For $l = 1, \ldots, \ell$, if any $\overline{s^l} = NULL$ then output $\overline{S} = NULL$ and terminate AVSS-MS-Rec. Else output $\overline{S} = (\overline{s^1}, \ldots, \overline{s^\ell})$ and terminate AVSS-MS-Rec.
```

nication and A-cast of $\mathcal{O}((\ell n^3 + n^4)\log\frac{1}{\epsilon})$ bits. Protocol AVSS-MS-Rec involves A-cast of $\mathcal{O}((\ell n^3 + n^4)\log\frac{1}{\epsilon})$ bits.

PROOF: The proof follows from the fact that in AVSS-MS-Share and AVSS-MS-Rec, there can be $\mathcal{O}(n)$ instances of AWSS-MS-Share and AWSS-MS-Rec respectively, each dealing with ℓ values and having an error parameter of $\epsilon' = \frac{\epsilon}{n}$. □

Remark 8 As mentioned earlier, Protocol AVSS-MS-Share does not force *corrupted* D to AVSS-commit some *meaningful* secret (i.e., S, containing ℓ elements from $\mathbb{F}$). We may assume that if D's AVSS-committed secret is $NULL$, then D has AVSS-committed some predefined $S^* \in \mathbb{F}^\ell$, which is known publicly. Hence in AVSS-MS-Rec, whenever $NULL$ is reconstructed, every honest party replaces $NULL$ by the predefined S^*. Interpreting this way, we say that our AVSS scheme allows D to *AVSS-commit* secret from $\mathbb{F}^\ell$.

8.3 An Incorrect Common Coin Protocol

In the previous section, we have presented an AVSS scheme (protocol AVSS-MS-Share, AVSS-Ms-Rec) that can share and reconstruct *multiple* secrets simultaneously and therefore it is much more communication efficient than multiple executions of AVSS scheme sharing and reconstructing *single* secret (i.e protocol AVSS-Share, AVSS-Rec). In section 5, we had recalled Common-Coin protocol from [16] that uses our protocols AVSS-Share and AVSS-Rec as black box. Specifically, each party in Common-Coin invokes n instances of protocol AVSS-Share each sharing a single secret. Simple thinking would suggest that those n instances of protocol AVSS-Share, each sharing a single secret could be replaced by more efficient single instance of AVSS-MS-Share, sharing n secrets simultaneously. This would naturally lead to more efficient common coin protocol, which would further imply more efficient ABA protocol. In the following, we do the same in protocol Common-Coin-Wrong. But as the name suggests, we then show that this direct replacement of AVSS-Share by AVSS-MS-Share without further modification will lead to an incorrect common coin protocol (i.e Common-Coin-Wrong is not a correct common coin protocol). In what follows, we first describe Common-Coin-Wrong and then point out the exact property where Common-Coin-Wrong deviates from Common-Coin. This will imply that Common-Coin-Wrong is not a correct solution for a common coin protocol. Protocol Common-Coin-Wrong is given in Fig. 14.

We now show that protocol Common-Coin-Wrong does not satisfy Lemma 14 which will further imply that Common-Coin-Wrong is not a correct common coin protocol. Specifically though it is true that: once some honest party P_j receives "`Attach` T_i `to` P_i" from the A-cast of P_i and includes P_i in $\mathcal{A}_j$, a unique value V_i is fixed such that any honest party will `associate` V_i with P_i; but now it is no longer ensured that V_i is distributed uniformly over $[0, \ldots, u]$. That is the adversary $\mathcal{A}_t$ can decide V_i for up to $t-1$ honest parties and thus those V_i are no longer random and uniformly distributed over $[0, \ldots, u]$. Consequently, $\mathcal{A}_t$ can enforce *some honest parties* to *always output* 0, while other honest parties *may output* $\sigma \in \{0,1\}$ with probability at least $\frac{1}{4}$. This will strictly violate the property of of common coin that *every honest party* should output $\sigma \in \{0,1\}$ with probability at least $\frac{1}{4}$.

Let P_i be an honest party. We now describe a specific behavior of $\mathcal{A}_t$ in Common-Coin-Wrong which would allow $\mathcal{A}_t$ to decide V_i to be 0 and thus make honest P_i to output 0 (this can be extended for $t-1$ honest P_is) whereas the remaining honest parties output $\sigma \in \{0,1\}$ with probability at least $\frac{1}{4}$. It is known that there are t parties under the control of $\mathcal{A}_t$. The specific behavior is given in Fig. 15.

Fig. 14 An Incorrect Common Coin Protocol Obtained by Replacing AVSS-Share and AVSS-Rec by AVSS-MS-Share and AVSS-MS-Rec Respectively in Protocol Common-Coin

Protocol Common-Coin-Wrong(ϵ)

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

1. For $j = 1, \ldots, n$, choose a random value x_{ij} and execute AVSS-MS-Share($P_i, \mathcal{P}, (x_{i1}, \ldots, x_{in}), \epsilon'$) where $\epsilon' = \frac{\epsilon}{n}$.
2. Participate in AVSS-MS-Share($P_j, \mathcal{P}, (x_{j1}, \ldots, x_{jn}), \epsilon'$) for every $j \in \{1, \ldots, n\}$. We denote AVSS-MS-Share($P_j, \mathcal{P}, (x_{j1}, \ldots, x_{jn}), \epsilon'$) by AVSS-MS-Share$_j$.
3. Create a dynamic set $\mathcal{T}_i$. Add party P_j to $\mathcal{T}_i$ if AVSS-MS-Share($P_j, \mathcal{P}, (x_{j1}, \ldots, x_{jn}), \epsilon'$) has been completed. Wait until $|\mathcal{T}_i| = t+1$. Then assign $T_i = \mathcal{T}_i$ and A-cast "`Attach` T_i `to` P_i". We say that the secrets $\{x_{ji} | P_j \in T_i\}$ are the secrets `attached` to party P_i.
4. Create a dynamic set $\mathcal{A}_i$. Add party P_j to $\mathcal{A}_i$ if the following holds:
   (a) "`Attach` T_j `to` P_j" is received from the A-cast of P_j and
   (b) $T_j \subseteq \mathcal{T}_i$.

   Wait until $|\mathcal{A}_i| = n - t$. Then assign $A_i = \mathcal{A}_i$ and A-cast "P_i `Accepts` A_i".
5. Create a dynamic set $\mathcal{S}_i$. Add party P_j to $\mathcal{S}_i$ if the following holds:
   (a) "P_j `Accepts` A_j" is received from the A-cast of P_j and
   (b) $A_j \subseteq \mathcal{A}_i$.

   Wait until $|\mathcal{S}_i| = n - t$. Then A-cast "`Reconstruct Enabled`". Let H_i be the current content of $\mathcal{A}_i$.
6. Participate in AVSS-MS-Rec($P_k, \mathcal{P}, (x_{k1}, \ldots, x_{kn}), \epsilon'$) for every $P_k \in T_j$ of every $P_j \in \mathcal{A}_i$ (Note that some parties may be included in $\mathcal{A}_i$ after the A-cast of "`Reconstruct Enabled`". The corresponding AVSS-MS-Rec are invoked immediately). We denote AVSS-MS-Rec($P_k, \mathcal{P}, (x_{k1}, \ldots, x_{kn}), \epsilon'$) by AVSS-MS-Rec$_k$.
7. Let $u = \lceil 0.87n \rceil$. Every party $P_j \in \mathcal{A}_i$ is `associated` with a value, say V_j which is computed as follows: $V_j = (\sum_{P_k \in T_j} x_{kj}) \mod u$ where x_{kj} is reconstructed back after executing AVSS-MS-Rec($P_k, \mathcal{P}, (x_{k1}, \ldots, x_{kn}), \epsilon'$).
8. Wait until the values `associated` with all the parties in H_i are computed. Now if there exits a party $P_j \in H_i$ such that $V_j = 0$, then output 0. Otherwise output 1.

Fig. 15 Specific Adversary Behavior in Protocol Common-Coin-Wrong

Possible Behavior of $\mathcal{A}_t$ in Protocol Common-Coin-Wrong()

1. Except a single corrupted party P_j, $\mathcal{A}_t$ asks all the remaining $t-1$ corrupted parties to participate in Common-Coin-Wrong honestly. P_j is asked to honestly participate in the instances of AVSS-MS-Share and AVSS-MS-Rec initiated by every other party acting as a dealer. But P_j is directed to *hold back* his invocation of AVSS-MS-Share as a dealer.
2. $\mathcal{A}_t$ being the scheduler in the network, stops all the messages sent to P_i and sent by P_i, except the messages related to P_i's instance of AVSS-MS-Share and AVSS-MS-Rec (this will not stop P_i to be part of anybody else's $\mathcal{T}_j$), until the following happens:
   (a) $n-t-1$ honest parties (except P_i) and $t-1$ corrupted parties (except P_j) carry out steps of Common-Coin-Wrong honestly, construct respective sets, A-cast "`Reconstruct Enabled`" and start invoking corresponding AVSS-MS-Rec$_k$ protocols.
   (b) This way the n secrets of each of $n-t-1$ honest parties (except P_i) and $t-1$ corrupted parties will be revealed. /* It is to be noted that the corrupted parties can successfully reconstruct secrets in AVSS-MS-Rec by behaving honestly even if the honest P_i does not participate in AVSS-MS-Rec.*/
   (c) Now $\mathcal{A}_t$ constructs a set T_i of size $t+1$ containing any t honest parties whose shared values $(x_{k1}, \ldots, x_{kn})$ are already disclosed to him plus corrupted party P_j.
   (d) Now $\mathcal{A}_t$ selects x_{ji} such that $V_i = (\sum_{P_k \in T_i} x_{kj}) \mod u = 0$ and asks P_j to invoke AVSS-MS-Share$_j$ with x_{ji} as the secret assigned to P_i.
3. $\mathcal{A}_t$ now schedules the messages to P_i such that P_i A-casts "`Attach` T_i `to` P_i" and eventually includes P_i in $\mathcal{A}_i$. So clearly H_i will contain P_i and hence P_i will output 0 since V_i is 0.

__**The Reason for the Problem**__: The adversary behavior specified in Fig. 15 become possible due to the fact that a corrupted party is able to commit his secrets for party P_i even after knowing what other parties has committed for P_i. This was strictly controlled in Common-Coin, where a corrupted party did not have any information about the secrets committed by other parties for P_i, while committing his own secret for P_i. In Common-Coin, secrets associated with P_i (that is the secrets corresponding to T_i) were disclosed only after T_i was fixed. This was possible as every party $P_k \in T_i$ committed their secrets independently using different instance of AVSS-Share. Thus as per requirement, corresponding AVSS-Rec was invoked to reconstruct the desired secret.

The above is not possible in Common-Coin-Wrong, because of simultaneous commitment and disclosure of n secrets in our AVSS-MS-Share and AVSS-MS-Rec. So a party P_l containing P_k in T_l may A-cast "`Reconstruct Enabled`" early and starts executing P_k's instance of AVSS-MS-Rec. This process will disclose the desired secret x_{kl}; but at the same time it will disclose other undesired secrets assigned to other parties. Now later the adversary may always schedule messages such that P_i includes such P_k's in T_i and some other corrupted parties who have seen the secrets committed by P_k for P_i and then has committed his own secrets. This clearly shows that the adversary can completely control the final output of P_i by deciding the value to be associated with P_i.

The above problem can be eliminated if we can ensure that no corrupted party can ever commit any secret after a single honest party starts reconstructing secrets. This is what we have achieved in our new common coin protocol presented in the next section.

8.4 A New and Efficient Common Coin Protocol for Multiple Bits

In this section, we show how to enhance protocol Common-Coin, so that it can handle the problem described in the previous section and can still use protocols AVSS-MS-Share and AVSS-MS-Rec as black-boxes. Before that we first extend the basic definition of common coin for multiple bit binary output.

Definition 7 (Multi-Bit Common Coin) Let π be an asynchronous protocol, where each party has local random input and ℓ bit output, where $\ell \geq 1$. We say that π is a $(1-\epsilon)$-terminating, t-resilient, multi-bit common coin protocol if the following requirements hold for every adversary $\mathcal{A}_t$:

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

The Intuition: We now present a multi-bit common coin protocol, called Common-Coin-MB. Protocol Common-Coin-MB goes almost in the same line as Common-Coin-Wrong except that we add some more steps and modify some of the steps due to which the corrupted parties are forced to commit/share their secrets much before they can reconstruct and access anybody elses' secrets. Thus contrary to protocol Common-Coin-Wrong, the values associated with every party P_i are now indeed random and are uniformly distributed over $[0, \ldots, u]$.

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

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

Another important feature of protocol Common-Coin-MB is that is a multi-bit common coin protocol. This is attained by using the ability of Vandermonde matrix [62,22] for extracting randomness. As a result, we could associate $n-2t$ values with each P_i, namely $V_{i1}, \ldots, V_{i(n-2t)}$ in Common-Coin-MB, while a single value V_i was associated with P_i in Common-Coin. This leads every party to output $\ell = n-2t$ bits in protocol Common-Coin-MB. We will show that the amortized communication cost of generating a single bit output in protocol Common-Coin-MB is far better than the communication cost of Common-Coin. As described in the subsequent sections, this is a definite move towards the improvement of the communication complexity of ABA protocol. We now briefly recall Vandermonde matrix and then present protocol Common-Coin-MB.

Vandermonde Matrix and Randomness Extraction [62,22]: Let $\beta_1, \ldots, \beta_c$ be distinct and publicly known elements. We denote an $(r \times c)$ Vandermonde matrix by $V^{(r,c)}$, where for $i = 1, \ldots, c$, the i^{th} column of $V^{(r,c)}$ is $(\beta_i^0, \ldots, \beta_i^{r-1})^T$. The idea behind extracting randomness using $V^{(r,c)}$ is as follows: without loss of generality, assume that $r > c$. Moreover, let $(x_1, \ldots, x_r)$ be such that:

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

Now if we compute $(y_1, \ldots, y_c) = (x_1, \ldots, x_r)V$, then $(y_1, \ldots, y_c)$ is a random vector of length c unknown to $\mathcal{A}_t$, extracted from $(x_1, \ldots, x_r)$ [62,22]. This principle is used in protocol Common-Coin-MB, which is given in Fig. 16.

Let E be an event, defined as follows: All invocations of AVSS scheme have been terminated properly. That is, if an honest party has terminated AVSS-MS-Share, then n values, say $(s'_1, \ldots, s'_n)$ are fixed. All honest parties will terminate the corresponding invocation of AVSS-MS-Rec with output $(s'_1, \ldots, s'_n)$. Moreover if dealer D is honest then $(s'_1, \ldots, s'_n)$ is D's shared secrets. It is easy to see that event E occurs with probability at least $1 - n\epsilon' = 1 - \epsilon$.

We now prove the properties of protocol Common-Coin-MB.

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

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

So let us first prove the first statement. Assuming every honest party has A-casted "`Reconstruct Enabled`", it will hold that eventually every honest party P_i will receive $n - t$ A-casts of "`Reconstruct Enabled`" from $n-t$ honest parties and will invoke AVSS-MS-Rec corresponding to every party in $\mathcal{T}_i$. Now it remains to show that AVSS-MS-Rec protocols invoked by every honest party P_i will be terminated eventually. It clear that a party P_k that is included in $\mathcal{T}_i$ of some honest party P_i will be eventually included in $\mathcal{T}_j$ of every other honest party P_j. Hence if P_i participates in AVSS-MS-Rec$_k$, then eventually every other honest party will do the same and thus AVSS-MS-Rec$_k$ will be completed by every body. Now every honest party will terminate protocol Common-Coin-MB after executing the remaining steps of Common-Coin-MB such as computing $V_{i1}, \ldots, V_{i(n-2t)}$ etc. Given event E, all invocations of AVSS-MS-Rec terminate in constant time. The black box pro-

Fig. 16 Multi-Bit Common Coin Protocol using Protocol AVSS-MS-Share and AVSS-MS-Rec as Black-Boxes

Protocol Common-Coin-MB(ϵ)

CODE FOR P_i: — All parties execute this code

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

tocol for A-cast terminates in constant time. This proves the first statement.

We next show that there is at least one honest party who will A-cast "`Reconstruct Enabled`". So assume that P_i is the *first honest party* to A-cast "`Reconstruct Enabled`". We will first show that this event will always take place. First notice that till P_i A-cast "`Reconstruct Enabled`", no honest party would halt any instance of AVSS-MS-Share. By the termination property of AVSS-MS-Share, every honest party will eventually terminate the instance of AVSS-MS-Share of every other honest party. Hence for every honest party P_j, every honest P_i will eventually receive A-cast of "P_k `terminated` P_j" from $n-t$ honest P_k's. Thus as there are at least $n-t$ honest parties, for every honest party P_i, $\mathcal{T}_i$ will eventually contain at least $n-t$ parties and hence P_i will eventually A-cast "`Attach` T_i `to` P_i". Furthermore eventually P_i will receive "`Attach` T_j `to` P_j" from every *honest* P_j. Now it is obvious that every party P_k included in $\mathcal{T}_j$ will be eventually included in $\mathcal{T}_i$ and thus $T_j \subseteq \mathcal{T}_i$ will hold good. Therefore, every honest P_j will be eventually included in $\mathcal{A}_i$. Thus for an honest P_i, $\mathcal{A}_i$ will eventually be of size $n-t$ and hence P_i will A-cast "P_i `Accepts` A_i". Now following the similar argument as above, we can show that for an honest P_i, $\mathcal{S}_i$ will eventually be of size $n-t$ and hence P_i will A-cast "`Reconstruct Enabled`". After this, P_i may stop executing at most t instances of AVSS-MS-Share corresponding to t parties.

Now we show that every other honest party P_j will also A-cast "`Reconstruct Enabled`" eventually. It is easy to see that every party that is included in $\mathcal{T}_i$ will also be included in $\mathcal{T}_j$ eventually. Now as P_i has already ensured that $\mathcal{T}_i$ contains at least $n-t$ parties, the same will hold good for P_j and P_j will eventually A-cast "`Attach` T_j `to` P_j". Furthermore, as P_i has already received "`Attach` T_j `to` P_j" from at least $n-t$ parties and checked that $T_j \subseteq \mathcal{T}_i$, eventually the same will hold for P_j and he will A-cast "P_j `Accepts` A_j". Following similar argument as above, P_j will A-cast "`Reconstruct Enabled`".

Given event E, all invocations of AVSS-MS-Share terminate in constant time. The black box protocol for A-cast terminates in constant time. Thus every honest party will A-cast "`Reconstruct Enabled`" in constant time. Hence protocol Common-Coin-MB terminates in constant time. □

We now prove the following important lemma, which is at the heart of protocol Common-Coin-MB. The lemma shows that the specific adversary behavior as specified in Fig. 15 is not applicable in protocol Common-Coin-MB.

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

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

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

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

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

PROOF: Once some honest party P_j receives "`Attach` T_i `to` P_i" from the A-cast of P_i and includes P_i in $\mathcal{A}_j$, $n-2t$ unique values $V_{i1}, \ldots, V_{i(n-2t)}$ are fixed. Here $V_{i1}, \ldots, V_{i(n-2t)}$ are defined following the step 8 of the protocol. We now prove the first part of the lemma. According to the lemma condition, $P_i \in \mathcal{A}_j$. This implies that $T_i \subseteq \mathcal{T}_j$. So honest P_j will participate in AVSS-MS-Rec$_k$ corresponding to each $P_k \in T_i$. Moreover, eventually $T_i \subseteq \mathcal{T}_k$ and $P_i \in \mathcal{A}_k$ will be true for *every other honest* P_k. So, every other honest party will participate in AVSS-MS-Rec$_k$ corresponding to each $P_k \in T_i$. Now by the property of AVSS-MS-Rec, each honest party will reconstruct x_{ki} at the completion of AVSS-MS-Rec$_k$, except with probability ϵ' (recall that each instance of AVSS-MS-Share, AVSS-MS-Rec has an associated error probability of ϵ' in termination). Thus, with probability $1-(n-t)\epsilon' \approx 1-\epsilon$, every honest party will associate $V_{i1}, \ldots, V_{i(n-2t)}$ with P_i.

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

Lemma 29 *In protocol Common-Coin-MB, once an honest party A-casts "`Reconstruct Enabled`", there exists a set M such that:*

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

Proof: The proof directly follows from the proof of Lemma 15 □

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

Proof: By Lemma 28, for every party P_i that is included in $\mathcal{A}_j$ of some honest party P_j, there exists some fixed (yet unknown) values $V_{i1}, \ldots, V_{i(n-2t)}$ that are distributed uniformly over $[0, \ldots, u]$ and with probability $(1-\epsilon)$ all honest parties will associate those $n-2t$ with P_i. Consequently, with the same probability, all the honest parties will agree on the value associated with each one of the parties (as there are n instances of AVSS-Rec, each with an error probability of $\epsilon' = \frac{\epsilon}{n}$). Now for every l^{th} bit, we may run the same argument as given in the proof of Lemma 16. □

Theorem 10 *Protocol Common-Coin-MB is a $(1-\epsilon)$-terminating, t-resilient multi-bit common coin protocol with $n-2t = t+1$ bits output for $n = 3t+1$ parties for every $0 < \epsilon \leq 0.2$.*

Proof: The **Termination** property follows from Lemma 26. The **Correctness** property follows from Lemma 27, Lemma 28, Lemma 29 and Lemma 30. □

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

Proof: Easy, as Common-Coin-MB executes n instances of AVSS-MS-Share and AVSS-MS-Rec with $\ell = n$ secrets and having an error parameter of $\frac{\epsilon}{n}$. □

Above theorem clearly leads to the following corollary.

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

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

8.5 Final ABA Protocol for Achieving Agreement on $t+1$ Bits Concurrently

Using our multi-bit common coin protocol, we now construct an ABA protocol, which allows the parties to reach agreement on multiple bits. Specifically, we design protocol ABA-MB, which attains agreement on $n-2t = t+1$ bits concurrently. So initially every party has a private input of $n-2t$ bits. Let the $n-2t$ bit input of P_i be denoted by $x_{i1}, \ldots, x_{i(n-2t)}$.

The Intuition: The high level idea of the protocol ABA-MB is similar to protocol ABA (given in Section 7). The ABA protocol proceeds in iterations where in each iteration every party computes his 'modified input', consisting of $n-2t$ bits. In the first iteration the 'modified input' of party P_i is nothing but the private input bits of P_i. In *each* iteration, every party executes the following protocols *sequentially*:

1. $n-2t$ parallel instances of Vote protocol, one corresponding to each bit of the 'modified input';

2. A *single* instance of Common-Coin-MB.

Notice that the parties participate in the instance of Common-Coin-MB, only after terminating all the $n-2t$ instances of Vote protocol. Now corresponding to l^{th} bit of his 'modified input', every party does the following computation: If the party outputs $(\sigma_l, 2)$ or $(\sigma_l, 1)$ in the l^{th} instance of Vote protocol, then he sets the l^{th} bit of his 'modified input' for next iteration to σ_l, irrespective of the l^{th} bit which is going to be output in Common-Coin-MB. Otherwise, he sets the l^{th} bit of his 'modified input' for next iteration to be the l^{th} bit, which is the output of Common-Coin-MB protocol. In case the party outputs $(\sigma_l, 2)$, he A-casts (σ_l, l) and once he receives $t+1$ A-casts for (σ_l, l), he concludes that agreement is reached for the l^{th} bit and therefore sets σ_l as the l^{th} output bit and performs no further computation related to l^{th} bit except for participating in the Common-Coin-MB instance of subsequent iterations. Finally, if a party concludes that agreement is reached on all the $t+1$ bits, he terminates the protocol ABA-MB. So essentially, in protocol ABA-MB, the parties parallely perform *almost* similar computation as in ABA, corresponding to each of the $t+1$ bits. However, instead of executing $n-2t$ instances of Common-Coin protocol, the parties execute *only a single instance* of Common-Coin-MB, which leads to the reduction in the communication complexity of protocol ABA-MB. The protocol is formally given in Fig. 17.

We now prove the properties of protocol ABA-MB.

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

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

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

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

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

PROOF: Follows from the proof of Lemma 23. □

Fig. 17 ABA Protocol to Reach Agreement on $n-2t = t+1$ Bits

```
Protocol ABA-MB($\epsilon$)

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

We now recall event C_k and C from section 7. Let C_k be the event that each honest party completes all the iterations he initiated up to (and including) the k^{th} iteration (that is, for each iteration $1 \le r \le k$ and for each party P, if P initiated iteration r then he computes $v_{(r+1)l}$ for every l^{th} bit). Let C denote the event that C_k occurs for all k.

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

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

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

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

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

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

PROOF: Follows from the proof of Lemma 25. □

Summing up, we have the following theorem.

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

9 Conclusion and Open Problems

We have presented a novel, constant expected time, optimally resilient, $(\epsilon, 0)$-ABA protocol whose communication complexity is significantly better than best known existing ABA protocols of [17, 1] (though the ABA protocol of [1] has a strong property of being *almost surely terminating*) with optimal resilience. Here we summarize the key factors that have contributed to the gain in the communication complexity of our ABA protocol: (a) A shorter route: $ICP \rightarrow AWSS \rightarrow AVSS \rightarrow ABA$, (b) Improving each of the building blocks by introducing new techniques and (c) By exploiting the advantages of dealing with multiple secrets concurrently in each of these blocks. It is to be mentioned that our new AVSS scheme significantly outperforms the existing AVSS schemes in the same settings in terms of communication complexity. An interesting open problem is to further improve the communication complexity of ABA protocols. Also one can try to provide an *almost surely terminating*, optimally resilient, constant expected time ABA protocol whose communication complexity is less than the ABA protocol of [1].

References

1. I. Abraham, D. Dolev, and J. Y. Halpern. An almost-surely terminating polynomial protocol for asynchronous Byzantine Agreement with optimal resilience. In R. A. Bazzi and B. Patt-Shamir, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC 2008, Toronto, Canada, August 18-21, 2008*, pages 405–414. ACM Press, 2008.
2. B. Altmann, M. Fitzi, and U. M. Maurer. Byzantine Agreement secure against general adversaries in the dual failure model. In P. Jayanti, editor, *Distributed Computing, 13th International Symposium, Bratislava, Slavak Republic, September 27-29, 1999, Proceedings*, volume 1693 of *Lecture Notes in Computer Science*. Springer Verlag, 1999.
3. Z. Beerliová-Trubíniová and M. Hirt. Efficient multi-party computation with dispute control. 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 305–328. Springer Verlag, 2006.
4. Z. Beerliová-Trubíniová and M. Hirt. Simple and efficient perfectly-secure asynchronous MPC. In K. Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 376–392. Springer Verlag, 2007.
5. Z. Beerliová-Trubíniová and M. Hirt. Perfectly-secure MPC with linear communication complexity. In R. Canetti, editor, *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230. Springer Verlag, 2008.
6. M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the Second Annual ACM SIGACT-SIGOPS Symposium on Princiles of Distributed Computing, August 17-19, 1983, Montreal, Quebec, Canada*, pages 27–30. ACM Press, 1983.
7. 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.
8. M. BenOr, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience. In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, California, USA, August 14-17*, pages 183–192. ACM Press, 1994.
9. P. Berman and J. A. Garay. Asymptotically optimal distributed consensus (extended abstract). In G. Ausiello,

M. Dezani-Ciancaglini, and S. R. D. Rocca, editors, *Automata, Languages and Programming, 16th International Colloquium, ICALP89, Stresa, Italy, July 11-15, 1989, Proceedings*, volume 372 of *Lecture Notes in Computer Science*, pages 80–94. Springer, 1989.

10. P. Berman and J. A. Garay. Cloture votes: n/4-resilient distributed consensus in t+1 rounds. *Mathematical Systems Theory*, 26(1):3–19, 1993.
11. P. Berman, J. A. Garay, and K. J. Perry. Towards optimal distributed consensus (extended abstract). In *Proceedings of 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, 30 October - 1 November 1989*, pages 410–415. IEEE Computer Society, 1989.
12. G. Bracha. An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. In *Proceedings of the Third Annual ACM Symposium on Princiles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 154 – 162. ACM Press, 1984.
13. G. Bracha. Asynchronous Byzantine Agreement protocols. *Inf and Computation*, 75(2):130–143, 1987.
14. G. Bracha. An $O(\log n)$ expected rounds randomized Byzantine generals protocol. *J. ACM*, 34(4):910–920, 1987.
15. G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985.
16. R. Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute, Israel, 1995.
17. 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.
18. B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 383–395. ACM Press, 1985.
19. B. A. Coan and J. L. Welch. Modular construction of a Byzantine Agreement protocol with optimal message bit complexity. *Information and Computation*, 97(1):61–85, 1992.
20. R. Cramer and I. Damgård. *Multiparty Computation, an Introduction*. Contemporary Cryptography. Birkhuser Basel, 2005.
21. 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.
22. I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In A. Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer Verlag, 2007.
23. D. Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982.
24. D. Dolev, M. J. Fischer, R. J. Fowler, N. A. Lynch, and H. R. Strong. An efficient algorithm for Byzantine Agreement without authentication. *Information and Control*, 52(3):257–274, 1982.
25. D. Dolev and R. Reischuk. Bounds on information exchange for Byzantine Agreement. *Journal of ACM*, 32(1):191–204, 1985.
26. D. Dolev, R. Reischuk, and H. R. Strong. Early stopping in Byzantine Agreement. *Journal of ACM*, 37(4):720–741, 1990.
27. D. Dolev and H. R. Strong. Polynomial algorithms for multiple processor agreement. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 401–407. ACM Press, 1982.
28. D. Dolev and H. R. Strong. Authenticated algorithms for Byzantine Agreement. *SIAM Journal of Computing*, 12(4):656–666, 1983.
29. P. Feldman and S. Micali. Byzantine Agreement in constant expected time (and trusting no one). In *Proceedings of 26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, 21-23 October 1985*, pages 267–276. IEEE Computer Society, 1985.
30. P. Feldman and S. Micali. An optimal algorithm for synchronous Byzantine Agreemet. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 639–648. ACM Press, 1988.
31. P. Feldman and S. Micali. An optimal probabilistic protocol for synchronous Byzantine Agreement. *SIAM Journal of Computing*, 26(4):873–933, 1997.
32. M. J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *FCT*, pages 127–140, 1983.
33. M. J. Fischer, N. A. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. In *Fault-Tolerant Distributed Computing*, pages 147–170, 1986.
34. M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *JACM*, 32(2):374–382, 1985.
35. M. Fitzi. *Generalized Communication and Security Models in Byzantine Agreement*. PhD thesis, ETH Zurich, 2002.
36. 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.
37. M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein, and A. Smith. Detectable Byzantine Agreement secure against faulty majorities. In *PODC 2002, Proceedings of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing, July 21-24, 2002 Monterey, California, USA*, pages 118–126. ACM Press, 2002.
38. M. Fitzi and M. Hirt. Optimally efficient multi-valued Byzantine Agreement. In Ruppert E and Malkhi D, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006*, pages 163–168, 2006.
39. M. Fitzi and U. M. Maurer. From partial consistency to global broadcast. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 494–503. ACM Press, 2000.
40. Z. Galil, A. J. Mayer, and M. Yung. Resolving message complexity of Byzantine Agreement and beyond. In *Proceedings of 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995*, pages 724–733. IEEE Computer Society, 1995.
41. J. A. Garay and K. J. Perry. A continuum of failure models for distributed computing. In A. Segall and S. Zaks, editors, *Distributed Algorithms, 6th International Workshop, WDAG '92, Haifa, Israel, November 2-4, 1992, Proceedings*,

volume 647 of *Lecture Notes in Computer Science*, pages 153–165. Springer Verlag, 1992.

42. 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.
43. O. Golderich, S. Micali, and A. Wigderson. How to play a mental game– a completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM Press, 1987.
44. M. Hirt, U. Maurer, and B. Przydatek. Efficient secure multiparty computation. In T. Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 143–161. Springer Verlag, 2000.
45. 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.
46. J. Katz and C. Y. Koo. On expected constant-round protocols for Byzantine Agreement. In C. Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, Lecture Notes in Computer Science, pages 445–462. Springer Verlag, 2006.
47. J. Katz and C. Y. Koo. Round-efficient secure computation in point-to-point networks. In M. Naor, editor, *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*, pages 311–328. Springer Verlag, 2007.
48. L. Lamport. The weak Byzantine generals problem. *Journal of ACM*, 30(3):668–676, 1983.
49. Y. Lindell, A. Lysyanskaya, and T. Rabin. On the composition of authenticated Byzantine Agreement. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montral, Qubec, Canada*, pages 514–523. ACM Press, 2002.
50. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
51. 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.
52. A. Patra, A. Choudhary, and C. Pandu Rangan. Efficient asynchronous multiparty computation with optimal resilience. Cryptology ePrint Archive, Report 2008/425, 2008.
53. A. Patra, A. Choudhary, and C. Pandu Rangan. Round efficient unconditionally secure multiparty computation protocol. In D. R. Chowdhury, V. Rijmen, and A. Das, editors, *Progress in Cryptology - INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008. Proceedings*, volume 5365 of *Lecture Notes in Computer Science*, pages 185–199. Springer Verlag, 2008.
54. A. Patra, A. Choudhary, and C. Pandu Rangan. Information theoretically secure multi party set intersection re-visited. In M. J. Jacobson Jr., V. Rijmen, and R. Safavi-Naini, editors, *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*. Springer Verlag, 2009.
55. 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.
56. M. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *JACM*, 27(2):228–234, 1980.
57. B. Pfitzmann and M. Waidner. Unconditional Byzantine Agreement for any number of faulty processors. In A. Finkel and M. Jantzen, editors, *STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, February 13-15, 1992, Proceedings*, volume 577 of *Lecture Notes in Computer Science*, pages 339–350. Springer Verlag, 1992.
58. 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.
59. T. Rabin. Robust sharing of secrets when the dealer is honest or cheating. *Journal of ACM*, 41(6):1089–1109, 1994.
60. 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.
61. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
62. K. Srinathan, A. Narayanan, and C. Pandu Rangan. Optimal perfectly secure message transmission. In M. K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 545–561. Springer Verlag, 2004.
63. S. Toueg. Randomized Byzantine Agreements. In *Proceedings of the Third Annual ACM Symposium on Princiles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 163–178. ACM Press, 1984.
64. S. Toueg, K. J. Perry, and T. K. Srikanth. Fast distributed agreement. *SIAM Journal of Computing*, 16(3):445–457, 1987.
65. R. Turpin and B. A. Coan. Extending binary Byzantine Agreement to multivalued Byzantine Agreement. *Information Processing Letters*, 18(2):73–76, 1984.
66. A. C. Yao. Protocols for secure computations. In *Proceedings of 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, 3-5 November 1982*, pages 160–164. IEEE Computer Society, 1982.

APPENDIX A: Analysis of the Communication Complexity of the AVSS, ABA Scheme of [17]

The communication complexity analysis of the AVSS and ABA protocol of [17] 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 [17] given in Canetti's Thesis [16]. To bound the error probability by ϵ, all the communication and computation in the protocol of [17] is done over a finite field $\mathbb{F}$, where $|\mathbb{F}| = GF(2^\kappa)$ and $\epsilon = 2^{-\Omega(\kappa)}$. Thus each field element can be represented by $\mathcal{O}(\kappa) = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

To begin with, in the ICP protocol of [17], D gives $\mathcal{O}(\kappa)$ field elements to INT and $\mathcal{O}(\kappa)$ field elements to verifier R. Though the ICP protocol of [16] is presented with a *single* verifier, it is executed with n verifiers in protocol A-RS. In order to execute ICP with n verifiers, D gives $\mathcal{O}(n\kappa)$ field elements to INT and $\mathcal{O}(\kappa)$ field elements to each of the n verifiers. So the communication complexity of ICP of [16] when executed with n verifiers is $\mathcal{O}(n\kappa)$ field elements and hence $\mathcal{O}(n\kappa^2)$ bits.

Now by incorporating their ICP protocol with n verifiers in Shamir secret sharing [61], the authors in [17] designed an asynchronous primitive called A-RS, which consists of two sub-protocols, namely A-RS-Share and A-RS-Rec. In the A-RS-Share protocol, D generates n shares (Shamir shares) of a secret s and for each of the n shares, D executes an instance of ICP protocol with n verifiers. So the A-RS-Share protocol of [17] involves a private communication of $\mathcal{O}(n^2\kappa^2)$ bits. In addition to this, the A-RS-Share protocol also involves an A-cast of $\mathcal{O}(\log(n))$ bits. In the A-RS-Rec protocol, the IC signatures given by D in A-RS-Share are revealed, which involves a private communication of $\mathcal{O}(n^2\kappa^2)$ bits. In addition, the A-RS-Rec protocol involves A-cast of $\mathcal{O}(n^2 \log(n))$ bits.

Proceeding further, by incorporating their A-RS protocol, the authors in [17] designed an AWSS scheme. The AWSS protocol consists of two sub-protocols, namely AWSS-Share and AWSS-Rec. In the AWSS-Share protocol, D generates n shares (Shamir shares [61]) of the secret and instantiate n instances of the ICP protocol for each of the n shares. Now each individual party A-RS-Share all the values that it has received in the n instances of the ICP protocol. Since each individual party receives a total of $\mathcal{O}(n\kappa)$ field elements in the n instances of ICP, the above step incurs a private communication of $\mathcal{O}(n^4\kappa^3)$ bits and A-cast of $\mathcal{O}(n^2\kappa \log(n))$ bits. In the AWSS-Rec protocol, each party P_i tries to reconstruct the values which are A-RS-Shared by each party P_j in a set $\mathcal{E}_i$. Here $\mathcal{E}_i$ is a set which is defined in the AWSS-Share protocol. In the worst case, the size of each $\mathcal{E}_i$ is $\mathcal{O}(n)$. So in the worst case, the AWSS-Rec protocol privately communicates $\mathcal{O}(n^5\kappa^3)$ bits and A-cast $\mathcal{O}(n^5\kappa \log(n))$ bits.

The authors in [17] then further extended their AWSS-Share protocol to Two&Sum AWSS-Share protocol, where each party P_i has to A-RS-Share $\mathcal{O}(n\kappa^2)$ field elements. So the communication complexity of Two&Sum AWSS-Share is $\mathcal{O}(n^4\kappa^4)$ bits and A-cast of $\mathcal{O}(n^2\kappa^2 \log(n))$ bits.

Finally using their Two&Sum AWSS-Share and AWSS-Rec protocol, the authors in [17] have deigned their AVSS scheme, which consists of two sub-protocols, namely AVSS-Share and AVSS-Rec. In the AVSS-Share protocol, the most communication expensive step is the one where each party has to AWSS-Rec $\mathcal{O}(n^3\kappa)$ field elements. So in total, the AVSS-Share protocol of [17] involves a communication complexity of $\mathcal{O}(n^9\kappa^4)$ bits and A-cast $\mathcal{O}(n^9\kappa^2 \log(n))$ bits. The AVSS-Rec protocol involves n instances of AWSS-Rec, resulting in a communication complexity of $\mathcal{O}(n^6\kappa^3)$ bits and A-cast of $\mathcal{O}(n^6\kappa \log(n))$ bits.

Now in the *common coin* protocol, each party in $\mathcal{P}$ acts as a dealer and invokes n instances of AVSS-Share to share n secrets. So the communication complexity of the common protocol of [17] is $\mathcal{O}(n^{11}\kappa^4)$ bits of private communication and $\mathcal{O}(n^{11}\kappa^2 \log(n))$ bits of A-cast. Now in the ABA protocol of [17], AVSS-Share protocol is called for $\mathcal{C} = \mathcal{O}(1)$ expected time. Hence the ABA protocol of [17] involves a private communication of $\mathcal{O}(n^{11}\kappa^4)$ bits and A-cast of $\mathcal{O}(n^{11}\kappa^2 \log(n))$ bits. As mentioned earlier, $\mathcal{O}(\kappa) = \mathcal{O}(\log \frac{1}{\epsilon})$. Thus the ABA protocol of [17] involves a private communication of $\mathcal{O}(n^{11} \log(\frac{1}{\epsilon})^4)$ bits and A-cast of $\mathcal{O}(n^{11} \log(\frac{1}{\epsilon})^2 \log(n))$ bits.