# Simple and Efficient Asynchronous Byzantine Agreement with Optimal Resilience

Arpita Patra [*], Ashish Choudhary [†], and C. Pandu Rangan [‡]
Dept. of Computer Science and Engineering, IIT Madras
Chennai, India 600036
arpitapatra10@gmail.com, partho_31@yahoo.co.in, prangan55@gmail.com

## ABSTRACT

Consider a completely asynchronous network consisting of $n$ parties where every two parties are connected by a private channel. An adversary $\mathcal{A}_t$ with *unbounded computing power* actively controls at most $t = (\lceil \frac{n}{3} \rceil - 1)$ out of $n$ parties in Byzantine fashion. In this setting, we say that $\pi$ is a $t$-resilient, $(1-\epsilon)$-terminating *Asynchronous Byzantine Agreement* (ABA) protocol, if $\pi$ satisfies all the properties of Byzantine Agreement (BA) in asynchronous settings tolerating $\mathcal{A}_t$ and terminates (i.e every honest party terminates $\pi$) with probability at least $(1-\epsilon)$. In this work, we present a new $t$-resilient, $(1-\epsilon)$-terminating ABA protocol which *privately* communicates $\mathcal{O}(\mathcal{C}n^6\kappa)$ bits and A-casts[1] $\mathcal{O}(\mathcal{C}n^6\kappa)$ bits, where $\epsilon = 2^{-\Omega(\kappa)}$ and $\mathcal{C}$ is the *expected running time* of the protocol. Moreover, conditioned on the event that our ABA protocol terminates, it does so in constant expected time; i.e., $\mathcal{C} = \mathcal{O}(1)$. Our ABA protocol is to be compared with the *only known* $t$-resilient, $(1-\epsilon)$-terminating ABA protocol of [5] in the same settings, which *privately* communicates $\mathcal{O}(\mathcal{C}n^{11}\kappa^4)$ bits and A-casts $\mathcal{O}(\mathcal{C}n^{11}\kappa^2\log(n))$ bits, where $\epsilon = 2^{-\Omega(\kappa)}$ and $\mathcal{C} = \mathcal{O}(1)$. So our ABA achieves a huge gain in communication complexity in comparison to the ABA of [5], while keeping all other properties in place. In another landmark work, in PODC 2008, Abraham et. al [1] proposed a $t$-resilient, 1-terminating (called as *almost-surely terminating* in [1]) ABA protocol which privately communicates $\mathcal{O}(\mathcal{C}n^6\log n)$ bits and A-casts $\mathcal{O}(\mathcal{C}n^6\log n)$ bits. But ABA protocol of Abraham et. al. takes polynomial ($\mathcal{C} = \mathcal{O}(n^2)$) expected time to terminate. Hence the merits of our ABA protocol over the ABA of Abraham et. al. are: (i) For any $\kappa < n^2\log n$, our ABA is better in terms of communication complexity (ii) conditioned on the event that our ABA protocol terminates, it does so in constant expected time (the constant is independent of $n$, $t$ and $\kappa$), whereas ABA of Abraham et. al. takes polynomial expected time. Summing up, in a practical scenario where a faster and communication efficient ABA protocol is required, our ABA fits the bill better than ABA protocols of [5, 1].

For designing our ABA protocol, we present a novel and simple *asynchronous verifiable secret sharing* (AVSS) protocol which significantly improves the communication complexity of the only known AVSS protocol of [5] in the same settings. We believe that our AVSS can be used in many other applications for improving communication complexity and hence is of independent interest.

**Categories and Subject Descriptors:** D. 4. 6 [Security and Protection]: Cryptographic Controls

**General Terms:** Security, Reliability.

**Keywords:** Unbounded Computing Power, VSS, Byzantine Agreement, Asynchronous Networks.

[*]Financial Support from Microsoft Research India Acknowledged.

[†]Financial Support from Infosys Technology India Acknowledged.

[‡]Work supported by Project No. CSE/05-06/076/DITX/CPAN on Protocols for Secure Communication and Computation, sponsored by Department of Information Technology, Govt. of India.

[1]A-Cast is a primitive in asynchronous world, allowing a party to send the same value to all the other parties. Hence A-Cast in asynchronous world is the parallel notion of broadcast in synchronous world.

## 1. INTRODUCTION

Byzantine Agreement (BA) [14] is one of the most fundamental problems in distributed computing. 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 that is the input value of one of the honest parties. 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 [9, 6, 4, 13]. An interesting and practically motivated variant of BA is *Asynchronous* BA (ABA) tolerating a *computationally unbounded* malicious adversary.

In ABA, the communication channels between the parties have arbitrary, yet finite delay (i.e the messages are guaranteed to reach eventually) and an adversary $\mathcal{A}_t$ with *unbounded computing power* can corrupt at most $t$ out of $n$ parties in Byzantine fashion. It is well known that ABA tolerating $\mathcal{A}_t$ is possible iff $n \geq 3t+1$ [13, 14]. Any ABA protocol designed with $n = 3t+1$ is therefore called as *optimally resilient*. By the seminal result of [10], any optimally resilient ABA protocol must have *non-terminating* runs, where some honest party(ies) may not output any value. So we say an ABA protocol to be $(1-\epsilon)$-terminating, if in the protocol, each honest party eventually terminates with probability $(1-\epsilon)$, where $\epsilon$ is the probability of not terminating. As a special case, we call an ABA protocol to be *almost-surely terminating*, a term coined by Abraham et. al in [1], if the set of non-terminating executions in the protocol has *probability zero*. The BA problem has been studied extensively over the past three decades and tight bounds have been established on resilience, round complexity and communication complexity for several variants of the problem in synchronous settings (see [13]). However, very little attention has been paid in designing fast, communication efficient, optimally resilient ABA protocol. Our result in this paper marks a significant progress in this direction.

**Existing Results**: In the following table, we sum up the following properties of existing ABA protocols: (i) Resilience, (ii) Communication Complexity (CC), (iii) Expected Running Time (ERT) and (iv) Probability of Termination (POT) (whether $(1-\epsilon)$-terminating or 1-terminating i.e. *almost-surely terminating*).

| Ref. | Resilience | CC | ERT | POT |
|---|---|---|---|---|
| [3] | $3t+1$ | $\mathcal{O}(2^n)$ | $\mathcal{C} = \mathcal{O}(2^n)$ | 1 |
| [8] | $4t+1$ | $poly(n)$ | $\mathcal{C} = \mathcal{O}(1)$ | 1 |
| [5, 4] | $3t+1$ | $poly(n, \kappa)$ | $\mathcal{C} = \mathcal{O}(1)$ | $(1-\epsilon)$ |
| [1] | $3t+1$ | $poly(n)$ | $\mathcal{C} = \mathcal{O}(n^2)$ | 1 |

Though the *optimally resilient* ABA protocols of Canetti et.al [5, 4] and Abraham et. al. [1] provide polynomial communication complexity (the exact figure is provided in the sequel), they are fairly high. So, designing optimally resilient, communication efficient ABA protocol which runs in constant expected time is the next natural important and interesting problem. In this paper we focus on designing optimally resilient ABA protocol which is communication efficient, as well as runs in constant expected time.

**Our Contribution:** We present an optimally resilient, $(1-\epsilon)$-terminating ABA protocol which privately communicates $\mathcal{O}(\mathcal{C}n^6\kappa)$ bits and A-casts $\mathcal{O}(\mathcal{C}n^6\kappa)$ bits, where $\epsilon = 2^{-\Omega(\kappa)}$ and $\mathcal{C}$ is the *expected running time* of the protocol. Moreover, conditioned on the event that our ABA protocol terminates, it does so in constant expected time; i.e., $\mathcal{C} = \mathcal{O}(1)$. In the following table, we compare and contrast our ABA protocol with the most recent optimally resilient ABA protocols.

| Ref. | CC (in bits) | ERT | POT |
|---|---|---|---|
| [5] | Private– $\mathcal{O}(\mathcal{C}n^{11}\kappa^4)$ A-cast– $\mathcal{O}(\mathcal{C}n^{11}\kappa^2\log(n))$ | $\mathcal{C} = \mathcal{O}(1)$ | $(1-\epsilon)$ |
| [1] | Private– $\mathcal{O}(\mathcal{C}n^6\log(n))$ A-cast– $\mathcal{O}(\mathcal{C}n^6\log n)$ | $\mathcal{C} = \mathcal{O}(n^2)$ | 1 |
| This Article | Private– $\mathcal{O}(\mathcal{C}n^6\kappa)$ A-cast– $\mathcal{O}(\mathcal{C}n^6\kappa)$ | $\mathcal{C} = \mathcal{O}(1)$ | $(1-\epsilon)$ |

So our ABA achieves a huge gain in communication complexity in comparison to the ABA of [5], while keeping all other properties in place. Our ABA enjoys the following merits over the ABA of Abraham et. al. [1]: (i) For any $\kappa < n^2\log n$, our ABA is better in terms of communication complexity (ii) our ABA runs in constant expected time. However, we stress that our ABA is $(1-\epsilon)$-terminating whereas ABA of [1] is *almost-surely terminating*.

Our construction of ABA protocol, employs a novel *asynchronous verifiable secret sharing* (AVSS) scheme with $n = 3t+1$. Roughly speaking, an AVSS scheme consists of a sharing phase and reconstruction phase. Informally, the goal of the AVSS scheme is to share a secret among $n$ parties during the sharing phase in a way that would later allow *unique* reconstruction of the secret in the reconstruction phase. Our AVSS protocol is far better in terms of communication complexity than the AVSS protocol of [5] in the same setting, while having the same properties.

**A Brief Discussion on the Approaches Used in the ABA Protocols of [5] and [1] and Current Article:** Almost all the ABA protocols in the past have used Rabin's [15] idea of reducing the problem of ABA to that of implementing a *common coin*. While Rabin [15] assumed that all the parties have access to a common coin (namely, a common source of randomness), Bracha [3] provided a naive implementation of *common coin* for the first time in the literature. Feldman and Micali [8] reduced the problem of efficiently implementing a common coin to that of efficiently implementing AVSS. For a comprehensive account on the reduction from AVSS to ABA, reader may refer [4].

1. The ABA protocol of Canetti et.al [5, 4] 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 [5, 4]). To overcome these difficulties, the authors in [5] 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 another well known primitive called *Asynchronous Weak Secret Sharing* (AWSS). 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 [5] 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 becomes highly communication intensive as well as very much involved. The protocol privately communicates $\mathcal{O}(n^9\kappa^4)$ bits, A-Cast $\mathcal{O}(n^9\kappa^2\log(n))$ bits during *sharing phase* and privately communicates $\mathcal{O}(n^6\kappa^3)$ bits, A-Cast $\mathcal{O}(n^6\kappa\log(n))$ bits during *reconstruction phase* [2] for sharing a single secret $s$, where all the honest parties terminate the protocol with probability at least $1-2^{-\Omega(\kappa)}$.

2. The ABA protocol of [1] used the same reduction from AVSS to ABA as in [5], 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

[2]The exact communication complexity analysis of the AVSS scheme of [5] was not done earlier and we have carried out the same. The complete analysis will appear in the full version of the paper.

AVSS to ABA as in [5]. 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 [5] to design an AVSS scheme, we follow a shorter route for designing our AVSS: $ICP \rightarrow AWSS \rightarrow AVSS$. Beside this, we significantly improve each of these building blocks by employing new design approaches. Together, this lead to our efficient AVSS and ABA protocols which we believe are much simpler than the AVSS and ABA of [5].

## 2. NETWORK MODEL AND DEFINITIONS

Model: We follow the asynchronous network model of [5], where there are $n$ parties, denoted by the set $\mathcal{P} = \{P_1, \ldots, P_n\}$, who are connected by pairwise reliable and secure channel. An adversary $\mathcal{A}_t$ with *unbounded computing power* can corrupt at most $t$ parties during the protocol in Byzantine fashion. Once a party is corrupted, he remains so throughout the protocol. A message sent through a channel may have arbitrary (but finite) delay. To model this, we assume that $\mathcal{A}_t$ controls the delay in transmission of the messages flowing through different channels and hence he can arbitrarily (but finitely) delay their transmission. Though $\mathcal{A}_t$ can delay the messages sent by the honest parties, it has no access to these messages. The error probability of our protocol is $2^{-\Omega(\kappa)}$, where $\kappa(>0)$ is called the error parameter. To bound the error probability of our protocols by $2^{-\Omega(\kappa)}$, all our protocols work over a finite field $\mathbb{F}$ where $\mathbb{F} = GF(2^\kappa)$. Thus each field element can be represented by $\kappa$ bits. Without loss of generality, we assume $\kappa = poly(n)$.

Asynchronous Weak Secret Sharing (AWSS) [5]: Let (Sh, Rec) be a pair of protocols in which a dealer $D \in \mathcal{P}$ shares a secret $s \in \mathbb{F}$. We say that (Sh, Rec) is a $t$-resilient AWSS scheme for $n$ parties if the following hold for every possible $\mathcal{A}_t$:

• **Termination**: With probability at least $1 - 2^{-\Omega(\kappa)}$, 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 at least one honest party has terminated Sh and if all the honest parties invoke protocol Rec, then each honest party will eventually terminate Rec.

• **Correctness**: With probability at least $1 - 2^{-\Omega(\kappa)}$, the following requirements hold:

1. If $D$ is *honest* then each honest party upon terminating protocol Rec, outputs the shared secret $s$.
2. If $D$ is *faulty* and some honest party has terminated Sh, then there exists a unique $s' \in (\mathbb{F} \cup NULL)$, such that each honest party upon completing Rec, will output either $s'$ or $NULL$.

• **Secrecy**: If $D$ is honest and no honest party has begun executing protocol Rec, then the corrupted parties have no information about the shared secret.

Asynchronous Verifiable Secret Sharing (AVSS) [5]: The **Termination** and **Secrecy** conditions for AVSS are same as in AWSS. The only difference is in **Correctness** 2 property:

**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 $1 - 2^{-\Omega(\kappa)}$, each honest party upon terminating Rec, will output $s'$.

Asynchronous Byzantine Agreement (ABA)[5]: Let $\Pi$ be an asynchronous protocol in which each party has a binary input and let $\kappa > 0$ be the error parameter. We say that $\Pi$ is a $(1 - 2^{-\Omega(\kappa)})$-terminating, $t$-resilient ABA protocol if the following hold, for every possible $\mathcal{A}_t$:

• **Termination**: With probability at least $1 - 2^{-\Omega(\kappa)}$ all honest parties terminate the protocol.

• **Correctness**: All the honest parties who have terminated hold identical outputs. Furthermore, if all the honest parties had the same input, say $\rho$, then all the honest parties output $\rho$.

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

• **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 $m^* = m$.

A-Cast of $b$ bits incurs a private communication of $\mathcal{O}(n^2 b)$ bits [3]. Notice that the termination property of A-Cast is much weaker than that of ABA because for A-Cast, it is not required that the honest parties terminate the protocol when the sender is faulty. 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.*

## 3. INFORMATION CHECKING PROTOCOL

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 [16] who have designed an ICP in synchronous settings. The ICP of Rabin et. al. was also used as a tool by Canetti et. al. [5] for designing ABA with optimal resilience (i.e $n = 3t + 1$).

As described in [16, 5, 7], 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 $R$ [16, 7, 5]. 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/AVSS protocol. 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 [16, 5], our AICP is also structured into three phases:

1. **Generation Phase**: is initiated by $D$. Here $D$ hands over the secret $s$ to *intermediary* $INT$. In addition, $D$ sends some *authentication information* to $INT$ and *verification information* to individual parties (verifiers) in $\mathcal{P}$.

2. **Verification Phase**: is carried out by $INT$ and the set of verifiers $\mathcal{P}$. Here $INT$ decides whether to continue or abort the protocol depending upon the prediction whether in **Revelation Phase**, secret $s$ held by $INT$ will be (eventually) accepted/will be considered as valid by all the (honest) verifiers in $\mathcal{P}$. $INT$ achieves this by setting a boolean variable $\mathsf{Ver} = 0/1$, where $\mathsf{Ver} = 0$ implies abortion of the protocol, while $\mathsf{Ver} = 1$ implies the continuation of the protocol. If $\mathsf{Ver} = 1$, then the *authentication information*, along with $s$, which is held by $INT$ at the end of **Verification Phase** is called $D$'s IC *signature* on $s$.

3. **Revelation Phase**: is carried out by $INT$ and the verifiers in $\mathcal{P}$. Here $INT$ reveals his secret $s$ along with the *authentication information*. The verifiers publish their responses with respect to the revealed information of $INT$. Depending upon the responses by the verifiers, a verifier $P_i \in \mathcal{P}$ either accepts $INT$'s secret $s$ 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 [5]):

1. If $D$ and $INT$ are honest, then $s$ will be accepted in **Revelation Phase** by each honest verifier.

2. If $INT$ is honest and $\mathsf{Ver} = 1$, then $s$ held by $INT$ will be accepted in **Revelation Phase** by each honest verifier, except with probability $2^{-\Omega(\kappa)}$.

3. If $D$ is honest, then during **Revelation Phase**, with probability at least $1 - 2^{-\Omega(\kappa)}$, every $s' \neq s$ produced by a corrupted $INT$ will not be accepted by an honest verifier.

4. If $D$ and $INT$ are honest and $INT$ has not started **Revelation Phase**, then $\mathcal{A}_t$ will have no information about $s$.

Notice that unlike other asynchronous primitives (e.g. AWSS, AVSS, ABA), we do not concentrate on defining the termination property of AICP. The reason is that AICP will never be executed as a stand alone application in our ABA. Rather, AICP will act as a tool to design AWSS and AVSS, which have their own termination properties. This is in line with [5], where ICP is defined without termination property and is used as a tool in AWSS/AVSS protocol. We now present a novel AICP called $\mathsf{A\text{-}ICP}$ (given in Table. 1) with $n = 3t + 1$.

The high level idea of the protocol is as follows: $D$ sets the secret $s$ as the constant term of a random polynomial $F(x)$ of degree $t$ and gives it to $INT$. $D$ also hands over some *secret evaluation points* and the value of $F(x)$ at those points to individual verifiers. This ensures that at a later stage, a corrupted $INT$ cannot produce an incorrect $F(x)$ during **Revelation Phase**, without being caught by honest verifiers. This ensures property 3 of AICP. Now to satisfy property 2 of AICP, an honest $INT$ has to ensure that his polynomial $F(x)$ will be accepted by honest verifiers during **Revelation Phase**. For this, $INT$ interacts with the verifiers and finds whether his polynomial is 'consistent' with the secret evaluation points and the values possessed by the verifiers. In order to do so, $INT$ and verifiers follow a zero-knowledge strategy. Then in **Revelation Phase**, $INT$'s secret is accepted on the basis of the consistency between the responses of verifiers during **Verification Phase** and the responses of verifiers in **Revelation Phase**. *In the sequel, whenever we say that $D$ gives his IC signature on $s$ to $INT$, we mean that $D$ starts the* **Generation Phase** *of* $\mathsf{A\text{-}ICP}$.

LEMMA 1. *If $D$ and $INT$ are honest, then $s$ will be accepted in $\mathsf{Reveal}$ by every honest verifier.*

PROOF: If $D$ and $INT$ are honest, then at least $t + 1$ honest verifiers from *ReceivedSet* will $\mathsf{A\text{-}cast}$ `Accept` and `Re-accept` during **Verification** and **Revelation Phase** respectively and hence will be considered as *consistent*. So each honest verifier $P_i$ will eventually output $\mathsf{Reveal}_i = s$. □

LEMMA 2. *If $INT$ is honest and $\mathsf{Ver} = 1$ at the end of* **Verification Phase**, *then $s$ held by $INT$ will be accepted by every honest verifier, except with probability $2^{-\Omega(\kappa)}$.*

PROOF: Since $INT$ is honest and $\mathsf{Ver} = 1$ at the end of **Verification Phase**, $INT$ must have received `Accept` from the $\mathsf{A\text{-}cast}$ of at least $t + 1$ verifiers of *ReceivedSet*. Moreover, there are $2t + 1$ verifiers in *ReceivedSet*, of which at least $t + 1$ are honest. We now show that the *honest verifiers* in *ReceivedSet* will be eventually considered as *consistent* during $\mathsf{Reveal}$ with very high probability, implying that every honest verifier will accept $INT$'s secret $s$.

Let $P_i$ be an honest verifier in *ReceivedSet* who $\mathsf{A\text{-}casts}$ `Accept` during **Verification Phase**. This implies that $B(\alpha_i) = dv_i + r_i$ holds, which further implies that either (a) $F(\alpha_i) = v_i$ and $R(\alpha_i) = r_i$ OR (b) $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$ (for other combinations, $B(\alpha_i) = dv_i + r_i$ will not hold). If (a) holds, then $P_i$ will $\mathsf{A\text{-}cast}$ `Re-accept` and will be considered as *consistent* during $\mathsf{Reveal}$. But on the other hand if (b) holds, then $P_i$ will $\mathsf{A\text{-}cast}$ `Re-reject` and will be considered as *inconsistent* during $\mathsf{Reveal}$. We now show that $P_i$ will be considered as *inconsistent* during $\mathsf{Reveal}$ with negligible probability. The reason is that for every $F(x), R(x)$ held by an honest $INT$ and $v_i, r_i$ held by an honest verifier $P_i$ in *ReceivedSet*, there is *only one* non-zero $d$ for which $B(\alpha_i) = dv_i + r_i$, even if (b) is true. For otherwise, assume there exists another non-zero element $e \neq d$, for which $B(\alpha_i) = ev_i + r_i$ holds, even if (b) holds. This implies that $(d - e)F(\alpha_i) = (d - e)v_i$ or $F(\alpha_i) = v_i$, which is a contradiction to (b). Now since $d$ is randomly chosen by $INT$ only after $D$ hands over $F(x), R(x)$ to $INT$ and $v_i, r_i$ to $P_i$, a corrupted $D$ has to guess $d$ in advance during $\mathsf{Gen}$, in order to make $P_i$ *inconsistent* during $\mathsf{Reveal}$. However, $D$ can guess $d$ with probability at most $\frac{1}{|\mathbb{F}|-1} \approx 2^{-\Omega(\kappa)}$. Thus if $P_i$ has $\mathsf{A\text{-}cast}$ `Accept` during **Verification Phase**, then it will be considered as *consistent* during $\mathsf{Reveal}$ with very high probability.

On the other hand, let $P_i$ be an honest verifier in *Received Set* who $\mathsf{A\text{-}casts}$ `Reject` during **Verification Phase**. This implies that $B(\alpha_i) \neq dv_i + r_i$ which further implies either $F(\alpha_i) \neq v_i$ or $R(\alpha_i) \neq r_i$ or both. What ever may be the case, $P_i$ will $\mathsf{A\text{-}cast}$ `Re-reject` during $\mathsf{Reveal}$ and will be always considered as *consistent*. Hence the lemma. □

**Protocol A-ICP$(D, INT, \mathcal{P}, s)$**

**Generation Phase: Gen$(D, INT, \mathcal{P}, s)$**

1. The dealer $D$, having secret $s$, selects a random $t$ degree polynomial $F(x)$ over $\mathbb{F}$, such that $F(0) = s$.
2. $D$ selects another random $t$ degree polynomial $R(x)$ over $\mathbb{F}$, independent of $F(x)$.
3. $D$ selects $n$ distinct, random, secret evaluation points $\alpha_1, \alpha_2, \ldots, \alpha_n$ from $\mathbb{F} \setminus \{0, 1, \ldots, n-1\}$.
4. $D$ sends $F(x)$ and $R(x)$ to $INT$. To verifier $P_i \in \mathcal{P}$, $D$ sends $\alpha_i, v_i$ and $r_i$, where $v_i = F(\alpha_i)$ and $r_i = R(\alpha_i)$. We call $\alpha_i, v_i$ and $r_i$ as *verification information* for $P_i$.

**Verification Phase: Ver$(D, INT, \mathcal{P}, s)$**

1. Verifier $P_i$ sends a `Received-From-D` signal to $INT$ after receiving *verification information* from $D$.
2. $INT$ waits for $2t+1$ `Received-From-D` signal. Upon receiving them, $INT$ puts the identity of the verifiers (from whom it has obtained `Received-From-D` signal) in a list, say *ReceivedSet* (clearly $|ReceivedSet| = 2t+1$). $INT$ then chooses a random $d \in \mathbb{F} \setminus \{0\}$, computes $B(x) = dF(x) + R(x)$ and A-casts $(d, B(x), ReceivedSet)$.
3. A verifier $P_i$, on receiving $(d, B(x), ReceivedSet)$ from the A-cast of $INT$, checks whether the following holds: (i) $P_i \in ReceivedSet$; and (ii) $B(\alpha_i) \overset{?}{=} dv_i + r_i$. If yes (no) then $P_i$ A-casts `Accept` (`Reject`).
4. $INT$ sets $Ver = 1$, if he receives `Accept` from the A-cast of at least $t+1$ verifiers who belong to *ReceivedSet*. In this case, we say that $INT$ has 'successfully' completed the **Verification Phase**. Moreover, ($F(x)$ and $R(x)$) held by $INT$ is called $D$'s IC *signature* on secret $s$. On the other hand, $INT$ sets $Ver = 0$, if he receives `Reject` from the A-cast of at least $t+1$ verifiers who belong to *ReceivedSet*.

**Revelation Phase: Reveal$(D, INT, \mathcal{P}, s)$**

1. $INT$ A-casts $F(x)$ and $R(x)$.
2. A verifier $P_i \in ReceivedSet$, on receiving $F(x)$ and $R(x)$ from the A-cast of $INT$, checks whether $F(\alpha_i) \overset{?}{=} v_i$ and $R(\alpha_i) \overset{?}{=} r_i$. Now to ensure that $INT$ has indeed set `Ver` $= 1$ during **Verification Phase**, $P_i$ waits to receive `Accept` from the A-cast of at least $t+1$ verifiers who belong to *ReceivedSet*. Once he receives the same, $P_i$ A-casts `Re-accept` if both the above tests (namely, $F(\alpha_i) \overset{?}{=} v_i$ and $R(\alpha_i) \overset{?}{=} r_i$) pass. If either one of the test fails, $P_i$ A-casts `Re-reject`.
3. (a) A verifier $P_i \in ReceivedSet$ is called as *consistent* if
   i. he A-casted `Accept` in **Verification Phase** and `Re-accept` in **Revelation Phase**, Or
   ii. he A-casted `Reject` in **Verification Phase** and `Re-reject` in **Revelation Phase**.

   (b) Similarly, a verifier $P_i \in ReceivedSet$ is called as *inconsistent* if
   i. he A-casted `Accept` in **Verification Phase** and `Re-reject` in **Revelation Phase**, Or
   ii. he A-casted `Reject` in **Verification Phase** and `Re-accept` in **Revelation Phase**.
4. If a verifier $P_i \in \mathcal{P}$ finds at least $(t+1)$ *consistent* verifiers from *ReceivedSet*, then $P_i$ sets $\mathsf{Reveal}_i = s$ and terminates, where $s$ is the constant term of of $F(x)$ which is A-casted by $INT$ in the first step of current phase. We say that $INT$ is 'successful' in producing *IC signature* to $P_i$.
5. If a verifier $P_i \in \mathcal{P}$ finds at least $(t+1)$ *inconsistent* verifiers from *ReceivedSet*, then $P_i$ sets $\mathsf{Reveal}_i = NULL$ and terminates. We say that $INT$ fails to correctly produce IC signature to $P_i$.

**Table 1: Information Checking Protocol,** $n = 3t+1$

Lemma 3. *If $D$ is honest, then during Reveal, with probability at least $1-2^{-\Omega(\kappa)}$, every $s' \neq s$ produced by a corrupted $INT$ will not be accepted by an honest verifier.*

Proof: A corrupted $INT$ may produce incorrect $B'(x) \neq B(x)$ during **Verification Phase**. But since $D$ is honest, a corrupted $INT$ will have no knowledge about $\alpha_i$'s corresponding to honest verifiers in *ReceivedSet*. So all the honest verifiers in *ReceivedSet* (at least $t+1$) will A-cast `Reject` with very high probability during **Verification Phase**. In order that an honest verifier $P_i$ in *ReceivedSet* A-casts `Accept` for $B'(x)$, a corrupted $INT$ has to ensure that $B'(\alpha_i) = B(\alpha_i)$. For this, he has to correctly guess $\alpha_i$, which he can do with probability $\frac{t}{|\mathbb{F}-n-1|} \approx 2^{-\Omega(\kappa)}$. So, none of the honest verifiers in *ReceivedSet* will ever respond during Reveal, as they will be waiting indefinitely for the A-cast of $t+1$ `Accept` signals from the verifiers in *ReceivedSet*.

So assume that a corrupted $INT$ has produced correct $B(x)$ during **Verification Phase**. All honest verifiers from *ReceivedSet* will then A-cast `Accept` eventually. But if $INT$ A-casts $F'(x) \neq F(x)$ to produce a different $s' \neq s$ during Reveal, then all honest verifiers in *ReceivedSet* will eventually A-cast `Re-reject` with very high probability. To make an honest verifier $P_i$ A-cast `Re-accept` during Reveal, $INT$ must ensure $F'(\alpha_i) = F(\alpha_i)$. For this, he has to correctly guess $\alpha_i$ which he can do with probability at most $\frac{t}{|\mathbb{F}-n-1|} \approx 2^{-\Omega(\kappa)}$. Thus with very high probability, all honest verifiers from *ReceivedSet* will eventually A-cast `Re-reject` and hence they will become *inconsistent*. So each honest verifier $P_i$ will eventually set $\mathsf{Reveal}_i = NULL$. □

Lemma 4. *If $D$ and $INT$ are honest and $INT$ has not started Reveal, then $\mathcal{A}_t$ will have no information about $s$.*

Proof: At the end of **Verification Phase**, $\mathcal{A}_t$ will know $d, B(x) = dF(x) + R(x)$ and $t$ distinct points on $F(x), R(x)$. Since $F(x)$ and $R(x)$ are random polynomials of degree $t$, the secret $s = F(0)$ will remain unknown to $\mathcal{A}_t$. □

Lemma 5. *In A-ICP, Gen privately communicates $\mathcal{O}(n\kappa)$ bits. Ver requires A-cast of $\mathcal{O}(n\kappa)$ bits and private communication of $\mathcal{O}(n\kappa)$ bits. Reveal requires A-cast of $\mathcal{O}(n\kappa)$ bits.*

## 4. AWSS

We now present a novel AWSS protocol called AWSS with $n = 3t+1$. Protocol AWSS consists of a pair of sub-protocols, (AWSS-Share, AWSS-Rec). While AWSS-Share allows $D$ to share a secret $s$, AWSS-Rec enables public reconstruction of either $D$'s shared secret or $NULL$. If $D$ is *corrupted*, then $s$ can be either from $\mathbb{F}$ or $NULL$ (in a sense explained in the sequel). We follow the general idea of [2, 7, 12, 11] in synchronous settings for sharing the secret $s$ with a degree-$t$ symmetric bivariate polynomial $F(x, y)$, where each party $P_i$ gets the univariate polynomial $f_i(x) = F(x, i)$. In particular, protocol AWSS is somewhat inspired by the WSS protocol of [7] in synchronous settings, with several new ideas incorporated in it.

```
AWSS-Share($D, \mathcal{P}, s$)
DISTRIBUTION: CODE FOR $D$
  1. Select a random degree-$t$ symmetric bivariate polynomial $F(x, y)$ over $\mathbb{F}$, such that $F(0, 0) = s$.
  2. For $1 \leq i \leq n$, deliver $f_i(x) = F(x, i)$ to $P_i$, along with IC signature on each $f_i(j)$ by considering $P_i$ as $INT$ and executing $\mathsf{Gen}(D, P_i, \mathcal{P}, f_i(j))$ for every $j \in \{1, \ldots, n\}$.
VERIFICATION: CODE FOR $P_i$
  1. Wait until $\mathsf{Gen}(D, P_i, \mathcal{P}, f_i(j))$ is completed for every $j \in \{1, \ldots, n\}$.
  2. Check whether the points $(f_i(1), \ldots, f_i(n))$ lie on a unique $t$-degree polynomial. If yes, then acting as $INT$, execute $\mathsf{Ver}(D, P_i, \mathcal{P}, f_i(j))$ for every $j \in \{1, \ldots, n\}$.
  3. If $\mathsf{Ver}(D, P_i, \mathcal{P}, f_i(j))$ is completed with $\mathsf{Ver} = 1$ for all $j \in \{1, \ldots, n\}$, then hand over $f_i(j)$ to $P_j$, along with IC signature by acting as dealer and executing $\mathsf{Gen}(P_i, P_j, \mathcal{P}, f_i(j))$ for all $j \in \{1, \ldots, n\}$. In addition, participate in $\mathsf{Gen}(P_j, P_i, \mathcal{P}, f_j(i))$ for all $j \in \{1, \ldots, n\}$ by acting as $INT$.
  4. Wait until $\mathsf{Gen}(P_j, P_i, \mathcal{P}, f_j(i))$ is completed. If $f_i(j) = f_j(i)$ then execute $\mathsf{Ver}(P_j, P_i, \mathcal{P}, f_j(i))$ as an $INT$. If $\mathsf{Ver}(P_j, P_i, \mathcal{P}, f_j(i))$ is completed with $\mathsf{Ver} = 1$, then A-cast $\mathtt{OK}(P_i, P_j)$.
CORE CONSTRUCTION: CODE FOR $D$
  1. For each $P_j$, build a set $OKSetP_j = \{P_i | D \text{ receives } \mathtt{OK}(P_i, P_j) \text{ from the A-Cast of } P_i\}$. When $|OKSetP_j| = 2t+1$, conclude that $P_j$'s IC-Commitment on $f_j(x)$ is over and add $P_j$ in $WCORE$ (which is initially empty).
  2. Wait until $|WCORE| = 2t+1$. Then A-cast $WCORE$ and $OKSetP_j$ for all $P_j \in WCORE$.
CORE VERIFICATION & AGREEMENT ON CORE: CODE FOR $P_i$
  1. Wait to receive $WCORE$ and $OKSetP_j$ for all $P_j \in WCORE$ from $D$'s A-cast, such that $|WCORE| = 2t+1$ and $|OKSetP_j| = 2t+1$ for each $P_j \in WCORE$.
  2. Wait to receive $\mathtt{OK}(P_k, P_j)$ for all $P_k \in OKSetP_j$ and $P_j \in WCORE$. After receiving, accept $WCORE$ and $OKSetP_j$'s and terminate AWSS-Share.
```

Briefly, the high-level idea of AWSS is as follows: In AWSS-Share, $D$ first hands over $n$ points on $f_i(x)$ to $P_i$, with his IC signature on these values. Then $D$, in conjunction with all other parties, perform a sequences of communications and computations. As a result of this, at the end of the sharing phase, every party agrees on a set of $2t+1$ parties, called $WCORE$, such that every party $P_j \in WCORE$ is *confirmed* by a set of $2t+1$ parties, called as $OKSetP_j$. A party $P_k \in OKSetP_j$ provides the *confirmation* to $P_j$, only when it possesses proper IC signature of $D$ on $f_k(j)$ ($j^{th}$ point on polynomial $f_k(x)$, which $P_k$ is entitled to receive from $D$) as well as IC signature of $P_j$ on the point $f_j(k)$ ($k^{th}$ point on polynomial $f_j(x)$, which $P_j$ is entitled to receive from $D$), such that $f_j(k) = f_k(j)$ holds (which should ideally hold due to the selection and distribution of symmetric bivariate polynomial). In some sense, we may view these checkings as every $P_j \in WCORE$ is attempting to commit his received (from $D$) polynomial $f_j(x)$ among the parties in $OKSetP_j$ (by giving his *IC Signature* on one point of $f_j(x)$ to each party) and the parties in $OKSetP_j$ 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 refer this commitment as $P_j$'s *IC-Commitment* on $f_j(x)$.

Achieving the agreement (among the parties) on $WCORE$ and corresponding $OKSet$s is a bit tricky in asynchronous network. Even though the *confirmations* are A-casted by parties, parties may end up with different versions of $WCORE$ and $OKSet$'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 $OKSet$s after receiving *confirmations* and ask $D$ to A-cast the same. After receiving $WCORE$ and $OKSet$s from the A-cast of $D$, individual parties ensure the validity of (verifies) these sets by receiving the same *confirmations* from the parties in the received $OKSet$s. A similar approach was used in [1].

In AWSS-Rec, the parties in $WCORE$ and corresponding $OKSet$'s are used for reconstructing the secret. Precisely, in the reconstruction phase, $P_j$'s *IC-Commitment* on $f_j(x)$ is revealed by reconstructing it with the help of the parties in $OKSetP_j$, for every $P_j \in WCORE$. Then the polynomials $f_j(x)$'s are used to construct the symmetric bivariate polynomial (if possible) $F(x, y)$ that is committed by $D$ during sharing phase. Since $f_j(x)$ is a degree-$t$ polynomial, any $t+1$ points on it are enough to interpolate $f_j(x)$. The points on $f_j(x)$ are obtained by requesting each party $P_k$ in $OKSetP_j$ to reveal IC signature of $D$ on $f_k(j)$ and IC signature of $P_j$ on $f_j(k)$ such that $f_j(k) = f_k(j)$ holds. Asking $P_k \in OKSetP_j$ to reveal $D$'s 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. This helps our AWSS protocol to satisfy Correctness 1 property of AWSS. Now asking $P_k$ in $OKSetP_j$ 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 the sharing phase. This ensure correctness 2 property. Summing up, when at least one of $D$ and $P_j$ is honest, $P_j$'s *IC-Commitment* on $f_j(x)$ is revealed properly. But when both $D$ and $P_j$ are corrupted, $P_j$'s *IC-Commitment* on $f_j(x)$ can be revealed as any $\overline{f_j(x)} \neq f_j(x)$. It is the later property that makes our protocol to qualify as a AWSS protocol rather than a AVSS protocol.

Lemma 6. *Protocol AWSS satisfies termination property.*

Proof: **Termination 1:** If $D$ is honest then he will eventually include all the $2t+1$ honest parties in $WCORE$ as well as in corresponding $OKSet$'s and will A-Cast the same. By the property of A-Cast, every honest party will receive $WCORE$ and $OKSet$'s, confirm their validity and terminate AWSS-Share eventually.

**Termination 2:** If an honest $P_i$ has terminated AWSS-Share, then he must have received $WCORE$ and $OKSetP_j$'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:** By Lemma 2, if an *honest* $P_i$ sets $\mathsf{Ver} = 1$, then IC signature produced by $P_i$ will be accepted in Reveal, except with probability $2^{-\Omega(\kappa)}$. For every $P_j \in WCORE$, $|OKSetP_j| = 2t + 1$. So there are at least $t + 1$ honest parties in $OKSetP_j$ who will be included in $ValidSetP_j$ with high probability. So for every $P_j \in WCORE$, $P_j$'s *IC-Commitment* will be reconstructed. Thus with very high probability, each honest party will terminate AWSS-Rec. □

```
AWSS-Rec($D, \mathcal{P}, s$)
SIGNATURE REVELATION: CODE FOR $P_i$
  1. If $P_i$ belongs to $OKSetP_j$ for some $P_j \in WCORE$, then participate (as an $INT$) in
     Reveal($D, P_i, \mathcal{P}, f_i(j)$) and Reveal($P_j, P_i, \mathcal{P}, f_j(i)$).
  2. Participate (as a verifier) in Reveal($D, P_k, \mathcal{P}, f_k(j)$) and Reveal($P_j, P_k, \mathcal{P}, f_j(k)$),
     for all $P_k \in OKSetP_j$ and $P_j \in WCORE$.
LOCAL COMPUTATION: CODE FOR $P_i$
  1. For every $P_j \in WCORE$, reconstruct $P_j$'s IC-Commitment on $f_j(x)$ by executing the following steps:
     (a) Construct a set $ValidSetP_j = \emptyset$.
     (b) Add party $P_k \in OKSetP_j$ to $ValidSetP_j$ if the following condition is true:
         Reveal($D, P_k, \mathcal{P}, f_k(j)$), Reveal($P_j, P_k, \mathcal{P}, f_j(k)$) are successfully completed,
         with outputs, Reveal$_i = \overline{f_k(j)}$ and Reveal$_i = \overline{f_j(k)}$ respectively
         and $\overline{f_k(j)} = \overline{f_j(k)}$.
     (c) Wait until $|ValidSetP_j| = t + 1$. Construct a polynomial $\overline{f_j(x)}$ passing through the points
         $(k, \overline{f_j(k)})$ where $P_k \in ValidSetP_j$. Associate $\overline{f_j(x)}$ with $P_j \in CORE$.
  2. Wait until IC-Commitment is reconstructed with a polynomial $\overline{f_j(x)}$ for each $P_j$ in $WCORE$.
  3. For each $(P_\gamma, P_\delta) \in WCORE$, check $\overline{f_\gamma(\delta)} \overset{?}{=} \overline{f_\delta(\gamma)}$.
     If the test passes for each pair of parties in $WCORE$, then construct the bivariate polynomial $\overline{F(x,y)}$
     using the polynomials $\overline{f_j(x)}$ associated with $P_j \in WCORE$, compute $\overline{s} = \overline{F(0,0)}$
     and terminate. Else set $\overline{s} = NULL$ and terminate.
```

LEMMA 7. *Protocol AWSS satisfies secrecy property.*

PROOF: Follows from the secrecy of A-ICP and properties of symmetric bivariate polynomial. □

LEMMA 8. *Protocol AWSS satisfies correctness property.*

PROOF: **Correctness 1:** Here we have to consider the case when $D$ is *honest*. We first prove that if $D$ is honest, then with very high probability, for every $P_j \in WCORE$, $P_j$'s *IC-Commitment* will be reconstructed correctly. In other words, $\overline{f_j(x)}$ associated with *every* $P_j \in WCORE$ during reconstruction phase, is same as $f_j(x)$ selected by $D$. From the property of A-ICP, for an *honest* $P_j \in WCORE$, a corrupted $P_k \in OKSetP_j$ can produce $P_j$'s valid signature on $\overline{f_j(k)} \neq f_j(k)$ with negligible probability (from Lemma 3). Similarly, for a *corrupted* $P_j \in WCORE$, a corrupted $P_k \in OKSetP_j$ can produce $P_j$'s valid signature on $\overline{f_j(k)} \neq f_j(k)$ but $P_k$ will fail to produce honest $D$'s signature on $\overline{f_k(j)} = \overline{f_j(k)}$ with very high probability. Thus with very high probability, corresponding to each $P_j \in WCORE$, the parties in $ValidSetP_j$ have produced correct points on $f_j(x)$. So $F(x, y)$ and $s = F(0, 0)$ will be reconstructed correctly by every party with very high probability.

**Correctness 2:** Here we consider the case, when $D$ is *corrupted*. Since in AWSS-Share, every honest party agrees on $WCORE$ and $OKSetP_j$ for $P_j \in WCORE$, a unique secret $s' \in \mathbb{F} \cup NULL$ is defined by (at least $t + 1$) honest parties in $WCORE$ at the end of AWSS-Share. We say that $s'$ is $D$'s *committed secret*. If the univariate polynomials of the honest parties in $WCORE$ define a degree-$t$ symmetric bivariate polynomial, say $F'(x, y)$, then $s'$ is the constant term of $F'(x, y)$. Otherwise $s' = NULL$.

We first consider the case when $s' = F'(0, 0)$. This implies that univariate polynomials corresponding to honest $P_j$'s in $WCORE$ define a symmetric $t$-degree bivariate polynomial $F'(x, y)$. We claim that with very high probability, for an honest $P_j \in WCORE$, $P_j$'s *IC-Commitment* will be reconstructed correctly. In other words, $\overline{f_j(x)}$ associated with an *honest* $P_j \in WCORE$ during reconstruction phase, is same as the one, which was received by $P_j$. This claim follows from the argument given in CORRECTNESS 1. But for a *corrupted* $P_j$ in $WCORE$, $P_j$'s *IC-Commitment* can be revealed as any degree-$t$ polynomial $\overline{f_j(x)}$. This is because a corrupted $P_k \in OKSetP_j$ 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 $P_k \in OKSetP_j$ are revealed (to parties) before the values of honest parties in $OKSetP_j$. Now if reconstructed $\overline{f_j(x)}$'s corresponding to *corrupted* $P_j$'s in $WCORE$, along with reconstructed $\overline{f_j(x)}$'s corresponding to *honest* $P_j$'s in $WCORE$ define $F'(x, y)$, 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 (who in turn define $F'(x, y)$), no other secret (other than $s'$) can be reconstructed.

On the other hand, let $D$'s committed secret $s' = NULL$. In this case irrespective of the behavior of corrupted parties, $NULL$ will be reconstructed by each honest party. □

LEMMA 9. *Both AWSS-Share and AWSS-Rec privately communicates $\mathcal{O}(n^3\kappa)$ bits and A-cast $\mathcal{O}(n^3\kappa)$ bits.*

THEOREM 1. *The pair (AWSS-Share, AWSS-Rec) constitutes a valid AWSS scheme with $n = 3t + 1$, which shares a single secret and satisfies all the properties of AWSS.*

IMPORTANT NOTE: In AWSS-Share, the degree-$t$ univariate polynomial $F(x, 0) = f_0(x)$ is used to share the secret $s$. *In the following when ever we say that $D$ shares a degree-$t$ polynomial $f(x)$ by executing AWSS-Share($D, \mathcal{P}, f(x)$)*, we mean that $D$ executes AWSS-Share with a degree-$t$ symmetric bivariate polynomial $F(x, y)$, such that $F(x, 0) = f(x)$. It should be noted that $f(x)$ is not completely random but only preserves the secrecy of the constant term which is the secret $s = f(0)$. Yet, this distribution of polynomials is sufficient to provide the secrecy requirements needed by our AVSS where AWSS is used as a building block. If $D$ indeed selects the bivariate polynomial in the above way and follows the protocol steps correctly, then as a result of the above execution, $P_i$ in $WCORE$ will hold the $i^{th}$ share $f(i) = F(i,0) = F(0,i)$, the polynomial $f_i(x) = F(x,i)$ and the share-share $f_j(i) = F(i,j) = f_i(j)$ corresponding to every other $P_j$. Similarly, AWSS-Rec$(D, \mathcal{P}, f(x))$ can be used for the reconstruction of $f(x)$ and secret $s = f(0)$.

## 5. AVSS

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

We now explain the high level idea used in AVSS. In AVSS-Share, $D$ selects a degree-$t$ symmetric bivariate polynomial $F(x,y)$, 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* his received polynomial $f_i(x)$. 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 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 party $P_j \in VCORE$, the instance of AWSS-Share where $P_j$ is dealer, 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 only when both $D$ and $P_j$ are corrupted in which case (a) *AWSS-Commitment* ensures that reconstruction of *AWSS-Commitment* can not be changed to some other polynomial other than $NULL$, but (b) *IC-Commitment* can not ensure the same. The agreement on $VCORE$ and corresponding $WCORE^{P_j}$'s is achieved using similar mechanism, as used in AWSS-Share for achieving agreement on $WCORE$ and corresponding $OKSets$.

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

DISTRIBUTION: CODE FOR $D$

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

COMMITMENT UPON VERIFICATION: CODE FOR $P_i$

1. Wait to obtain $f_i(x)$ from $D$.
2. If $f_i(x)$ is a degree-$t$ polynomial then as a dealer, execute AWSS-Share$(P_i, \mathcal{P}, f_i(x))$ by selecting a degree-$t$ symmetric bivariate polynomial $Q^{P_i}(x,y)$ such that $Q^{P_i}(x,0) = q_0^{P_i}(x) = f_i(x)$. 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$^{P_j}$.

CORE CONSTRUCTION: CODE FOR $D$

1. Wait to terminate AWSS-Share$^{P_j}$ with $WCORE^{P_j}$ and $OKSetP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$. Then add $P_j$ in a set $TempCORE$ (initially empty). Here $j \in \{1, \ldots, n\}$.
2. Even after terminating AWSS-Share$^{P_j}$, update (include new parties in) $WCORE^{P_j}$ and $OKSetP_k^{P_j}$'s upon receiving new A-casts of the form OK(.,.) as a part of AWSS-Share$^{P_j}$. Also update $TempCORE$ upon terminating AWSS-Share for new parties.
3. After every update, perform the following computations:
   (a) Assign $VCORE = TempCORE$ and check whether $|VCORE \cap WCORE^{P_j}| \geq 2t+1$ for every $P_j \in VCORE$. If not then remove $P_j$ from $VCORE$ and keep on repeating this until no more party can be removed from $VCORE$.
   (b) Check whether $|VCORE| \geq 2t+1$. If not, then delete $VCORE$ and wait for more updates. Otherwise, A-cast $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKSetP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$. Conclude that each $P_j \in VCORE$ is *AWSS-committed* to $f_j(x)$.

CORE VERIFICATION & AGREEMENT ON CORE : CODE FOR $P_i$

1. Wait to receive $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKSetP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$ from $D$'s A-cast, such that $|VCORE| \geq 2t+1$ and for each $P_j \in VCORE$, $|VCORE \cap WCORE^{P_j}| \geq 2t+1$ and $|OKSetP_k^{P_j}| \geq 2t+1$ for every $P_k \in WCORE^{P_j}$.
2. Wait to terminate AWSS-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 $P_m \in OKSetP_k^{P_j}$ and $P_k \in WCORE^{P_j}$. After receiving all OKs, accept the $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKSetP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$ and terminate **AVSS-Share**.

In AVSS-Rec, $D$'s committed secret is recovered with the help of the parties in $VCORE$ and $WCORE^{P_j}$'s. Precisely, 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.

```
AVSS-Rec(D, P, s)

SECRET RECONSTRUCTION: CODE FOR $P_i$
  1. For every $P_j \in VCORE$, reconstruct $P_j$'s AWSS-Commitment on $f_j(x)$ as follows:
     (a) Participate in AWSS-Rec($P_j, \mathcal{P}, f_j(x)$) with $WCORE^{P_j}$ and $OKSetP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$. 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.
```

**NULL Commitment in AVSS**: In AVSS, we say that $D$'s committed secret is the constant term of the symmetric degree-$t$ bivariate polynomial $F'(x,y)$ which is defined by the univariate polynomials of the honest parties in $VCORE$. For an *honest* $D$, the polynomials of the honest parties in $VCORE$ will always define a symmetric bivariate polynomial and thus the committed secret is always a *valid field element* and hence considered as *meaningful*. But for a *corrupted* $D$, this may not hold. In this case, we say that $D$ is committed to $NULL$. The reconstruction phase of our AVSS protocol ensures the reconstruction of *the committed secret*.

LEMMA 10. *Protocol AVSS satisfies termination property.*

PROOF: **Termination 1:** If $D$ is honest, then corresponding to every honest $P_j$, AWSS-Share$^{P_j}$ will eventually terminate with $2t+1$ honest parties in $WCORE^{P_j}$. Thus eventually all the $2t+1$ honest parties will be included in $TempCORE$ and $D$ will eventually get $VCORE = TempCORE$, such that $|VCORE| \geq 2t+1$ and $|VCORE \cap WCORE^{P_j}| \geq 2t+1$ for each $P_j \in VCORE$. From similar argument given in **Termination 1** of Lemma 6, all honest parties will eventually agree on $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKSetP_k^{P_j}$ and will terminate AVSS-Share.

**Termination 2:** The proof follows from the similar argument given in **Termination 2** of Lemma 6.

**Termination 3:** Follows from the fact that corresponding to each $P_j \in VCORE$, every honest $P_i$ will terminate AWSS-Rec$^{P_j}$ (from **Termination 3** of Lemma 6). □

LEMMA 11. *Protocol AVSS satisfies correctness property.*

PROOF: **Correctness 1:** We have to consider the case when $D$ is honest. If $D$ is *honest* then we prove that with very high probability, for every $P_i \in FINAL$, $P_i$'s *AWSS-Commitment* will be reconstructed correctly. In other words, AWSS-Rec$^{P_i}$ will disclose $\overline{f_i(x)}$ which is same as $f_i(x)$ selected by honest $D$. For every *honest* $P_i \in FINAL$ this is trivially true. We have to prove the above statement for a corrupted $P_i \in FINAL$. If a corrupted $P_i$ belongs to $FINAL$, it implies AWSS-Rec$^{P_i}$ is successful (i.e., the output is a symmetric degree-$t$ bivariate polynomial) and AWSS-Share$^{P_i}$ had terminated during AVSS-Share. Now termination of AVSS-Share ensures $|VCORE \cap WCORE^{P_i}| \geq 2t+1$. The above statements have the following implications together: (a) With very high probability, $P_i$ must have given consistent polynomials to the honest parties in $WCORE^{P_i}$ (during AWSS-Share$^{P_i}$) such that they induce valid degree-$t$ symmetric bivariate polynomial (see **Correctness 2** of Lemma 8). (b) $P_i$ 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 statements in (a) and (b) together imply that $P_i$ must have committed (to the honest parties in $WCORE^{P_i}$ which are 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 $\overline{Q^{P_i}(x,y)} = Q^{P_i}(x,y)$ 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 $\overline{F(x,y)} = F(x,y)$. Thus $s = \overline{F(0,0)} = F(0,0)$ will be recovered.

**Correctness 2:** Since in AVSS-Share, every honest party agrees on a common $VCORE$, a unique secret $s' \in (\mathbb{F} \cup NULL)$ is defined by (at least $t+1$) honest parties in $VCORE$. If the univariate polynomials of the honest parties in $VCORE$ define a degree-$t$ symmetric bivariate polynomial, say $F'(x,y)$, then $s'$ is the constant term of $F'(x,y)$. Otherwise $s' = NULL$. When $s' = NULL$, we say that committed $s'$ is not *meaningful*.

Let $s' = NULL$. Now for every honest $P_i \in VCORE$, AWSS-Rec$^{P_i}$ will disclose $\overline{f_i(x)}$ which is same as the one received by $P_i$ in sharing phase. Hence, all honest parties from $VCORE$ will be added to $FINAL$ with very high probability. 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 $(P_\gamma, P_\delta)$ of honest parties and $NULL$ will be reconstructed.

On the other hand, let $s'$ be *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 completely resembles with the case when $D$ is honest and hence the proof follows from the proof of **Correctness 1**. □

LEMMA 12. *Protocol AVSS satisfies secrecy property.*

PROOF: Follows from Lemma 7, Lemma 4 and properties of symmetric bivariate polynomial. □

LEMMA 13. *Both Protocol AVSS-Share and Protocol AVSS-Rec communicate $\mathcal{O}(n^4\kappa)$ bits and A-cast $\mathcal{O}(n^4\kappa)$ bits.*

IMPORTANT NOTE: Protocol AVSS does not force *corrupted* $D$ to commit some *meaningful* secret (i.e., an element from $\mathbb{F}$). Hence, the secret $s$, committed by a *corrupted* $D$ can be either from $\mathbb{F}$ or $NULL$. We may assume that if $D$'s committed secret is $NULL$, then $D$ has 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 *commit* secret from $\mathbb{F}$.

## 6. EFFICIENT ABA

Using our AVSS protocol, we design an ABA protocol with optimal resilience. The first step is to get a *common coin* [4]. In the common coin protocol of [4], every party shares $n$ random secrets using $n$ different instances of AVSS protocol of [5, 4]. We replace the AVSS scheme of [5, 4] with protocol AVSS to obtain an efficient common coin protocol, which we call as Efficient-Common-Coin.

LEMMA 14. *Protocol Efficient-Common-Coin privately communicates $\mathcal{O}(n^6\kappa)$ bits and A-cast $\mathcal{O}(n^6\kappa)$ bits.*

PROOF: Easy, as Efficient-Common-Coin executes $n^2$ instances of AVSS-Share and AVSS-Rec. □

Efficient-Common-Coin will satisfy all the properties of the common coin protocol of [4] (see Lemmas 5.27-5.31 of [4]). In [4, 5], the authors have used their common coin protocol (that terminates with probability $(1 - 2^{-\Omega(\kappa)})$) to get a $(1-2^{-\Omega(\kappa)})$-terminating, $t$-resilient ABA protocol (see Figure 5.11 of [4]). We replace the common coin protocol of [4] by Efficient-Common-Coin to obtain our efficient $(1 - 2^{-\Omega(\kappa)})$-terminating, $t$-resilient ABA with $3t + 1$, which we call as Efficient-ABA. Similar to the ABA protocol of [5], conditioned on the event that Efficient-ABA terminates, it does so in constant expected time. The proof for this follows from same arguments given in [4].

THEOREM 2. *Protocol Efficient-ABA privately communicates $\mathcal{O}(\mathcal{C}n^6\kappa)$ bits and A-casts $\mathcal{O}(\mathcal{C}n^6\kappa)$ bits, where $\mathcal{C} = \Theta(1)$ is the expected running time of the protocol.*

PROOF: Since Efficient-ABA terminates in $\mathcal{C} = \Theta(1)$ expected time, Efficient-Common-Coin will be called constant expected number of times. This follows from the similar argument as given in [5, 4]). Hence the theorem. □

## 7. CONCLUSION AND OPEN PROBLEMS

We have presented a novel, constant expected time, $(1 - 2^{-\Omega(\kappa)})$-terminating, optimally resilient ABA protocol whose communication complexity is significantly better than existing ABA protocols of [5, 1] (though the ABA protocol of [1] has a strong property of being *almost surely terminating*). The key factors that have contributed to this gain in the communication complexity are: (a) a shorter route: $ICP \rightarrow AWSS \rightarrow AVSS \rightarrow ABA$, (b) Improving each of the building blocks by introducing new techniques.

Though all our primitives, namely, A-ICP, AWSS, AVSS, are designed to deal with a *single* secret, they can be extended for dealing with *multiple secrets* concurrently. We can modify Efficient-Common-Coin appropriately to incorporate AVSS dealing with multiple secrets *concurrently.* Thus by exploiting and harnessing the advantages of dealing with multiple secrets concurrently in each of the above primitives, we can further reduce the communication complexity of our ABA to $\mathcal{O}(n^4\kappa)$ bits (private communication and A-cast). The details will be available in the full version of the paper.

An interesting open problem is to further improve the communication complexity of ABA protocols. Also one can try to design an *almost surely terminating*, optimally resilient, constant expected time ABA protocol whose communication complexity is less than the ABA protocol of [1].

**Acknowledgements**: We would like to thank the anonymous referees of PODC 2009 for many helpful suggestions.

## 8. REFERENCES

[1] I. Abraham, D. Dolev, and J. Y. Halpern. An Almost Surely Terminating Polynomial Protocol for Asynchronous Byzantine Agreement with Optimal Resilience. In *PODC*, pages 405–414, 2008.

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *STOC*, pages 1–10, 1988.

[3] G. Bracha. An Asynchronous $\lfloor (n-1)/3 \rfloor$-Resilient Consensus Protocol. In *PODC*, pages 154 – 162, 1984.

[4] R. Canetti. *Studies in Secure Multiparty Computation and Applications.* PhD thesis, Weizmann Institute, Israel, 1995.

[5] R. Canetti and T. Rabin. Fast Asynchronous Byzantine Agreement with Optimal Resilience. In *STOC*, pages 42–51, 1993.

[6] B. Chor and C. Dwork. Randomization in Byzantine Agreement. *Advances in Computing Research*, 5:443–497, 1989.

[7] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient Multiparty Computations Secure Against an Adaptive Adversary. In *EUROCRYPT*, pages 311–326, 1999.

[8] P Feldman and S. Micali. An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement. *SIAM Journal of Computing*, 26(4):873–933, 1997.

[9] M. Fischer. The Consensus Problem in Unreliable Distributed System. Technical Report, Department of Computer Science, Yale University, 1983.

[10] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *JACM*, 32(2):374–382, 1985.

[11] M. Fitzi, J. Garay, S. Gollakota, C. Pandu Rangan, and K. Srinathan. Round-Optimal and Efficient Verifiable Secret Sharing. In *TCC*, pages 329–342, 2006.

[12] R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin. The Round Complexity of Verifiable Secret Sharing and Secure Multicast. In *STOC*, pages 580–589, 2001.

[13] N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[14] M. Pease, R. E. Shostak, and L. Lamport. Reaching Agreement in the Presence of faults. *JACM*, 27(2):228–234, 1980.

[15] M. Rabin. Randomized Byzantine Generals. In *FOCS*, pages 403–409, 1983.

[16] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In *STOC*, pages 73–85, 1989.