Fast Asynchronous Byzantine Agreement with Optimal Resilience *

Ran Canetti † Tal Rabin ‡

IBM T.J. Watson Research Center

September 10, 1998

Abstract

It is known that, in both asynchronous and synchronous networks, no Byzantine Agreement (BA) protocol for n players exists if $\lceil\frac{n}{3}\rceil$ of the players are faulty (in other words, no BA protocol is $\lceil\frac{n}{3}\rceil$-resilient). The only known *asynchronous* $(\lceil\frac{n}{3}\rceil-1)$-resilient BA protocol runs in expected exponential time, and the best resilience achieved by an asynchronous protocol with polynomial complexity is $(\lceil\frac{n}{4}\rceil-1)$. The question whether there exists an asynchronous $(\lceil\frac{n}{3}\rceil-1)$-resilient BA protocol with polynomial complexity remained open.

We answer this question in the affirmative. Consider a completely asynchronous network of n players, in which every two players are connected via a private channel. Furthermore, let the faulty players behave maliciously and have unlimited computational power. In this setting, we describe an $(\lceil\frac{n}{3}\rceil-1)$-resilient BA protocol. Except for negligible probability all the non-faulty players complete the execution of the protocol. Conditioned on the event that all the non-faulty players complete the execution of the protocol, they do so in *constant* expected time, and polynomial complexity.

Our construction employs an $(\lceil\frac{n}{3}\rceil-1)$-resilient Asynchronous Verifiable Secret Sharing (AVSS) protocol. The best resilience previously achieved by an AVSS protocol is $(\lceil\frac{n}{4}\rceil-1)$. Our AVSS protocol employs new asynchronous tools which are of independent interest.

*An extended abstract of this work appeared in the proceedings of the 25th Symposium on Theory of Computation, 1993 [CR93].

†email:canetti@theory.lcs.mit.edu

‡email: talr@theory.lcs.mit.edu. Supported by the Eshkol Fellowship of the Israel Ministry of Education.

1 Introduction

The problem of reaching agreement in the presence of faults is one of the most fundamental problems in distributed computing. A particularly interesting variant of this problem, introduced by Pease, Shostak and Lamport [PSL80] allows Byzantine faults (namely, the faulty players may collaborate in any malicious way). This variant, called the Byzantine Agreement (BA) problem, is formulated as follows. Design a protocol that allows the non-faulty players to agree on a common value. The agreed value should equal the input value of one of the non-faulty players.

The BA problem was investigated extensively in various models, characterized by the synchrony of the network, the privacy of the channels, the computational power of the players, and other parameters. We refer the interested reader to the early surveys of Fischer [Fis83] and Chor and Dwork [CD89]; see also a quick overview below. Yet, despite extensive research a few important questions have remained open. One of these questions is the subject of this paper.

State of the art and our result. Lower bounds on the resilience of BA protocols were proven in [PSL80]. (Informally, a protocol is t-resilient if the outputs of the non-faulty players meet the specifications as long as up to t of the players are faulty.) They showed that agreement cannot be reached by a *deterministic* protocol in an n-player *synchronous* network with $\lceil \frac{n}{3} \rceil$ Byzantine faults. Karlin and Yao generalized this result to randomized protocols [KY86]. Clearly, these results apply to *asynchronous* networks as well. Furthermore, Fischer, Lynch and Paterson's impossibility result for *deterministic, asynchronous* agreement protocols [FLP85] implies that any (randomized) asynchronous protocol reaching BA must have non-terminating runs. (Roughly, a network is *synchronous* if the computation proceeds in rounds, where the messages sent in a round arrive at the beginning of the next round. In an *asynchronous* network the players have no global clock and messages may incur arbitrary, yet finite, delays.)

On the positive side, in a synchronous network Garay and Moses present a *deterministic* BA protocol that is $(\lceil \frac{n}{3} \rceil - 1)$-resilient, runs in $t+1$ rounds (which is optimal for deterministic protocols), and has polynomial complexity [GM98]. The best known *randomized, synchronous* BA protocol, in a setting where the players are connected via *private* channels, is the Feldman-Micali protocol [FM88]. This protocol is $(\lceil \frac{n}{3} \rceil - 1)$-resilient, runs in expected constant time and has polynomial complexity. If the adversary is limited to probabilistic polynomial time then the private channels can be 'implemented' by means of encryption [GM84]. (All randomized BA protocols discussed here have zero error probability. That is, whenever the players complete the execution, then they output a correct value.)

In a model where private channels are unavailable and the adversary is computationally unbounded (this is often called the full information model) the best known protocol is due to Chor and Coan [CC85]. This protocol is $(\lceil \frac{n}{3} \rceil - 1)$-resilient and runs in $O(t/\log t)$ rounds. Very recently Ben-Or has reported an improvement of that result to $O(\sqrt{t})$ rounds for $t = \Theta(n)$, together with a matching lower bound on the number of rounds for that model.

In *asynchronous* networks the best known protocol with optimal (i.e., $(\lceil \frac{n}{3} \rceil - 1)$) resilience is Bracha's protocol that runs in $\Theta(2^n)$ expected time [Bra84]. Feldman generalizes the [FM88] construction to the asynchronous setting, yielding a constant expected time, $(\lceil \frac{n}{4} \rceil - 1)$-resilient asynchronous BA protocol [Fel89]. Like [FM88], Feldman's protocol requires private channels. It remained an open question (cf. [FM88, CD89]) whether there exists an $(\lceil \frac{n}{3} \rceil - 1)$-resilient, asynchronous BA protocol with polynomial complexity.

We answer this question in the affirmative. Consider a completely asynchronous network of n players, in which each two players are connected via a private channel. Furthermore, let the adversary controlling the faulty players have unlimited computational power. In this setting, we describe a Byzantine Agreement protocol that is $(\lceil\frac{n}{3}\rceil - 1)$-resilient. With overwhelming probability all the non-faulty players complete the execution of the protocol. Conditioned on the event that all the non-faulty players have completed the execution of the protocol, they do so in constant expected time and with polynomial message complexity. (We do not know whether there exist $(\lceil\frac{n}{3}\rceil - 1)$-resilient BA protocols that terminate with probability 1 and have polynomial complexity.)

Previous results and techniques. Let us describe the chain of results and techniques leading to our result. Rabin describes an $(\lceil\frac{n}{8}\rceil - 1)$-resilient Byzantine Agreement protocol that runs in constant expected time, provided that all the players have access to a 'common coin' (namely, a common source of randomness) [Rab83]. Rabin's construction can be used in synchronous as well as asynchronous networks. Bracha improved the resilience of Rabin's protocol in an *asynchronous* network, given a 'common coin', to $(\lceil\frac{n}{3}\rceil - 1)$ [Bra84]. Implicitly, Bracha also suggests a simple protocol for implementing a 'common coin', and shows that the running time of his BA protocol is inversely proportional to the probability that all players compute the same value of the coin (call this probability the success probability of a common coin protocol). The success probability of Bracha's common coin protocol is exponentially small in the number of players. (Basically, in Bracha's protocol each player tosses a coin locally and the players hope that they all got the same value.) Consequently, Bracha's BA protocol has expected exponential running time.

The essence of the [FM88] synchronous protocol is a common coin protocol with constant success probability. Once a common-coin is achieved, the players proceed in a conceptually similar manner to Bracha's protocol. At the heart of the [FM88] common coin protocol is an $(\lceil\frac{n}{3}\rceil - 1)$-resilient *Verifiable Secret Sharing* (VSS) protocol (a notion defined by Chor *et al.* [CGMA85]). Feldman has generalized the [FM88] common coin protocol to asynchronous networks. Given an r-resilient Asynchronous VSS (AVSS) protocol, Feldman's asynchronous common coin protocol is $min(r, \lceil\frac{n}{3}\rceil - 1)$-resilient. AVSS protocols which are $(\lceil\frac{n}{4}\rceil - 1)$-resilient are presented in [Fel89, BCG93]. Yet, up to now, no known AVSS protocol has been more than $(\lceil\frac{n}{4}\rceil - 1)$-resilient.

The technical difficulty. We offer an intuitive exposition of the difficulties encountered in trying to devise an $(\lceil\frac{n}{3}\rceil - 1)$-resilient AVSS protocol. An AVSS protocol is composed of two phases: (1) a sharing phase, in which a dealer shares a secret among the players, and each player verifies for itself that a unique secret has been defined by the shares, and (2) a reconstruction phase in which the players reconstruct the secret from their shares. A player P must be able to complete the execution of the sharing phase even if it never got any message from up to t players (since these players may be faulty and never send messages). This means that in the sharing phase P may have communicated only with a subset (denoted C_1) of $n - t$ players. When P proceeds to the reconstruction phase it again can wait to receive messages from at most $n-t$ of the players. Denote this set by C_2. The set C_2 might not contain all of C_1, and in addition may contain players with whom P has not communicated in the sharing phase. The shares of the players in $C_2 - C_1$ may not be in accordance with the secret defined by the shares of the players in C_1. (In fact, the players not in C_1 may have not receive shares at all.) Thus, P can depend only on the shares of the players in the intersection of C_1 and C_2, denoted by C, for reconstructing the secret. Still, t out of the players in C may be faulty. As long as $n \geq 4t+1$, it holds that $|C| \geq 2t+1$; thus, the majority of the players in C are uncorrupted. However, when $n = 3t+1$, it is possible that $|C| = t+1$. In this case, there might be only a single uncorrupted player in C, thus correct reconstruction seems implausible.

Our solution. We overcome these difficulties by developing a mechanism that allows each player P to make sure, when it adds a player Q to the set C_1, that in the reconstruction phase P will have *some* value to associate with Q's share. (I.e. in a way we "force" Q to communicate.) If Q is faulty then this value is arbitrary and chosen at the time of reconstruction, but if Q is uncorrupted then the associated value is the correct one. This is achieved using the Information Checking Protocol of [Rab94, RB89]. This mechanism facilitates to achieves a 'weakened' version of AVSS, which we call A-RS. A-RS only guarantees that the value reconstructed by each player will belong to a fixed and small set of values. Using A-RS as a primitive we construct a slightly stronger protocol, called Asynchronous Weak Secret Sharing (AWSS). In an AWSS protocol, the value reconstructed by each player will be either the shared value or *null*. Using AWSS, we construct our AVSS protocol. (Both AWSS and AVSS generalize synchronous constructs introduced in [Rab94, RB89].)

For self-containment, and since the Feldman protocol [Fel89] was never published, we also present a protocol for reaching BA given an AVSS protocol. Our protocol is based on [Rab83, Bra84, Fel89]. Put together, our protocols constitute an asynchronous $(\lceil \frac{n}{3} \rceil - 1)$-resilient BA protocol.

We note that our $(\lceil \frac{n}{3} \rceil - 1)$-resilient AVSS protocol has been used also to achieve asynchronous secure computation with optimal resilience [BKR94]. Another usage is for strengthening the results of Ben-or and El-Yaniv for bounding the total running time of many parallel invocations of a BA protocol [BE91].

Organization. In Section 2 we describe the asynchronous model and formally define Byzantine Agreement and Asynchronous Verifiable Secret Sharing. In Section 3 we state our main theorems, and present an overview of our protocols. In Section 4 we describe tools used in our construction. In Sections 5 through 7 we describe our A-RS, AWSS, and AVSS protocols. In Sections 8 and 9 we describe our BA protocol given an AVSS protocol.

2 Definitions

2.1 The model of computation

We describe the asynchronous model of computation, define the running time of protcols. We also briefly discuss a different formalization of the asynchronous model, namely the I/O automata model. Finally, we set some conventions for presenting asynchronous protocols in a modular and hierarchical way.

Asynchronous protocols. An n-player asynchronous protocol is a collection of n players $P_1, ..., P_n$, where each player is a probabilistic interactive Turing machine (see [GMR89]). Each machine is equipped, as usual, with input tape, random input tape, and output tape. (The input tape contains 1^n, followed by any input the player may have.) In addition a machine is equipped with an incoming message tape and an outgoing messages tape. Initially the machine is in a special start state and with empty work tape. Upon activation the machine proceeds in a computation based on the current state, the current contents of the work, input, random input, and incoming message tapes. During the computation the contents of the output and outgoing messages tapes are updated. The activation is completed when the machine enters a special activation done state. Alternatively the machine can also move into a halt state. (Intuitively, an activation is the computation that follows the receipt of a message. The 'activation done' state signifies that the processing of a message is completed and the player is now waiting for the next message. The 'halt' state means that the player has halted and will not process more messages.) The contents of the outgoing messages tape is partitioned via standard encoding to a sequence of messages. Each message $m = (i, j, \mu)$ consists of the sender's identity i (that is, i is the machine's identity), the intended recipient's identity j, and the message body μ. Let the frame of a message contain only the sender and recipient identity, and the number of bits in the body.

In the sequel we set some conventions for describing complex interactive Turing machines in a modular way.

Executions and the adversary. Executing a protocol is a process defined via an additional interactive Turing machine called the adversary and denoted $\mathcal{A}$. Without loss of generality the adversary is deterministic; also, it has no input. In the sequel we describe the adversary in slightly higher level terms than those of a Turing machine. A description in terms of Turing machines can be extracted from this one.

An execution, parameterized by some input and random input values for the players (input x_i and random input r_i for each player P_i), consists of a sequence of atomic steps. In each atomic step the adversary chooses a single player, P_i, that has not halted. The adversary then chooses a message m to be written on P_i's incoming message tape. (In this case we say that message m is delivered. Below we place restrictions on the values of the delivered messages.) Next player P_i is activated in the special start state, with x_i written on its input tape, with r_i written on its random input tape, and with the contents of the work tape being identical to its contents at the completion of the previous activation (or blank if this is the first activation). Upon completion of the activation, the adversary learns the *frames* of the messages in P_i's outgoing messages tape. We stress that the adversary does not learn the bodies of these messages.

The adversary is restricted as follows in deciding which messages will appear on the incoming message tape of an activated player P_i. In the first activation of each player the message is a special `wakeup` message. In each subsequent activation the message must be one that was previously sent to P_i (ie, it appeared on the outgoing messages tape of some player with intended recipient P_i). In addition each message must be delivered at most once. We stress that messages need not be delivered in the order in which they were sent.

In addition to activating players, the adversary may corrupt players in an adaptive way. That is, at the end of each atomic step the adversary may choose to corrupt some player P_j. (The choice is made based on the information gathered so far by the adversary.) In this case the adversary learns the entire computational history of P_j, including all past contents of all of P_j's tapes. In addition, from this point on the adversary is allowed to deliver any arbitrary messages as long as the sender of these messages is P_j. Also, P_j is no longer activated and has no output. A t-adversary is an adversary that in any execution corrupts at most t players. (We remark that in the sequel the term uncorrupted player always refers to a certain step in an execution. Clearly, an uncorrupted player can become corrupted at a later step.)

We say that a player has completed an execution of the protocol when it has halted (ie, it moved to the 'halt' state). An execution is complete if all the uncorrupted players have completed. The output of a complete execution is the concatenation of the contents of the output tapes of all (uncorrupted) players.[1]

An execution is legal if each player that remained uncorrupted throughout was activated at least once, and each message sent is eventually delivered (unless its recipient is corrupted or has already halted). Notice that executions that complete do so after a finite number of steps. (These executions are also legal.) Yet, legal executions that do not complete can be either finite or infinite.

A time measure. Let us briefly present a definition of the running time of a protocol in our asynchronous model. (For more general definitions of asynchronous time, see, for instance, [AFL83, Lyn96].)

Informally, imagine an 'external clock' measuring time in the network (of course, the players 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 this execution. The duration of a finite execution is the total time measured by the external clock divided by the period of this execution. (Infinite executions have infinite duration.)

More formally, consider an execution of an asynchronous protocol with some inputs and random inputs for the players, and some adversary. Recall that this execution is a sequence of atomic steps, where in each atomic step some player is activated. The first atomic step is a 'fictitious' step where the `wakeup` messages are sent to all players. We assign round-numbers to atomic steps, as follows. The first ('fictitious') atomic step, denoted l_0, is the only step at round 0. Next, for each $i > 0$ let l_i be the last atomic step where an $(i-1)$-message is delivered. (A message is an $(i-1)$-message if it was sent in an atomic step that belongs to round $i-1$.) All the steps *after* step l_{i-1} *until* (and including) step l_i are in round i. The duration of the execution is the round number of the last atomic step.

In the sequel we also discuss the durations of sub-sequences of an execution (say, from a given event until a given event). The definition is similar. Notice that if an execution is partitioned into consecutive sub-sequences then the duration of the entire execution is at most the sum of the durations of the sub-sequences.

The expected running time of a protocol, relative to an adversary and some input values for the players, and conditioned on an event, is the expected value of the duration of a *legal* execution, where the expectancy is taken over the random inputs of the players in which this event occurs. The (non relative) expected running time $T(\pi|v)$ of protocol π, conditioned on event v, is the maximum over all inputs $\vec{x} = x_1, ...x_n$ and applicable adversaries $\mathcal{A}$, of the expected running time of the protocol relative to input $\vec{x}$ and adversary $\mathcal{A}$ and conditioned on event v:

$$T(\pi|v) = \text{Max}_{\vec{x},\mathcal{A}}\{\text{Exp}_{\vec{r}}[D(\pi, \mathcal{A}, \vec{x}, \vec{r})|v]\}$$

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

[1] In scenarios where it is important to capture the information gathered by the adversary from the computation it is convenient to let the adversary have its own output, and include the adversary output in the output of the computation (see for instance [BCG93]). Here we do not care about this information, thus we do not include the adversary's output in the output of the computation.

On the I/O automata model. A very general formal model for describing asynchronous protocols is the I/O automata model [LT89, Lyn96]. Of special interest to this work is the version that allows for randomized protocols, namely the probabilistic I/O automata model [SL95]. In the I/O automata model the system is modeled via an automaton that consists of states and transitions. The states model configurations of the network and the transitions model (local) computational steps between configurations. (A transition can model, say, an internal computational step within some player or the delivery of a message.) Typically an I/O automaton is partitioned into several sub-automata called modules. Some modules model the programs run by the communicating players, and other modules model the behavior of the communication channels. An execution of an I/O automaton is a legal sequence of transitions. A probabilistic I/O automaton introduces probability distributions over the allowed transitions going out of a certain state.

The I/O automata model is very general and allows for formal (and modular) presentation and analysis of protocols in several computational models. Yet we choose to present our protocols in the more specific (and limited) interactive Turing machines model. We find this model convenient for capturing the allowed behavior and capabilities of the adversary, and for presenting the complex randomized protocols of this paper in a clear way. In addition, the interactive Turing machine model seems to facilitate future incorporation of computational limitations of the adversary. Naturally, the protocols of this paper can be presented and analyzed also in the I/O automata model.

Writing asynchronous protocols. We describe some writing conventions for asynchronous protocols. It is convenient to describe an asynchronous protocol as a sequence of enumerated items. Upon activation, the player scans all items in the specified order. An item has one of three possible forms:

- "<instruction>." Here the instruction is carried out once at the first activation of the protocol.
- "`Wait until` <condition>. `Then` <instruction>." Here, if the condition is satisfied, *and the instruction was not yet executed in a previous activation,* then the instruction is executed. Otherwise the instruction is ignored.
- "`If` <condition> `then` <instruction>." Here, if the condition is satisfied, the corresponding instruction is executed, regardless of whether it was executed in a previous activation.

For ease of presentation we partition protocols into several modules, called sub-protocols. Each sub-protocol is first presented and analyzed as if it were the only protocol run by the player. Sub-protocols are combined by letting one sub-protocol 'call' other sub-protocols during its execution. This writing style should be interpreted as follows. Within each protocol there is an implicit protocol manager code, that operates as follows. The protocol manager keeps track of the different invocations of sub-protocols that are currently 'in execution'. The current local state and contents of the work-tape of each invocation is kept separately. We assume that each message is addressed only to a single invocation, and that an identifier for this invocation is specified within m, using some standard encoding. Upon activation of a player with incoming message m, the protocol manager decides, based on information specified in m, which invocation to activate. The invocation then runs as if it were the only protocol run by the player (with some exceptions described below.) Once the activation of this invocation is completed, the protocol manager writes the outgoing messages generated by this invocation on the outgoing messages tape of the player. Note that this structure is hierarchical; that is, each sub-protocol may have its own protocol manager and partitioning to yet other sub-protocols. If an incoming message is addressed to an invocation that was completed then the message is discarded. If an incoming message is addressed to an invocation that is not yet in execution then this message is 'kept aside'. When this invocation is called for the first time then the manager activates the invocation on all the kept messages addressed to it.[2] If in some activation the activated invocation I generates output (or completes its execution) then the protocol manager immediately activates the invocation that 'called' invocation I with an appropriate notice.

Many of our variables contain sets that only get values added to them. Here we let $U^{(l)}$ denote the set held in variable U when the number of elements in U is l for the first time.

We make the following exception to the partitioning of the protocol into sub-protocols. We use a special type of variable, called shared variables. Unlike regular variables, a shared variable used by some invocation can be updated by another invocation that is in execution at the same time. Consequently, shared variables may have different values in different activations of an invocation without being modified by this invocation. (In this case, the variable has been updated by another invocation.) The shared variable we use are typically updated by a single invocation, and are read by one or more other invocations. (Typically, the reading invocation will 'wait until' a set held in a shared variable reaches some size.) Consequently, if in some activation an invocation I updates a shard variable, then the relevant protocol manager is notified. When the activation of invocation I is complete, the protocol manager immediately activates the other invocations that use (or "wait on") this shared variable with a special message indicating that the shared variable has been updated. To avoid confusion, we use calligraphic letters (e.g., $\mathcal{U}$) to denote shared variables.

2.2 Asynchronous Byzantine Agreement

Definition 1 *Let π be an asynchronous protocol for which each one of the n players has binary input. We say that π is a $(1-\epsilon)$-*completing, *t-*resilient Byzantine Agreement *protocol if the following requirements hold, for every t-adversary and every input vector for the players.*

• **Completion.** *With probability $1-\epsilon$ all the players complete the protocol.*[3]

• **Correctness.** *All the uncorrupted players that have halted have identical outputs. Furthermore, if all the players that remain uncorrupted throughout the execution have the same input, σ, then all these players output σ upon completion.*

2.3 Asynchronous Verifiable Secret Sharing

A secret sharing protocol (either synchronous or asynchronous) consists of two protocols: a sharing protocol, in which a dealer "shares" a secret among the players, and a reconstruction protocol, in which the players "reconstruct" the secret from its shares. An uncorrupted dealer should be able to share a secret while making sure that the reconstructed value will be the correct secret. Furthermore, if the dealer is uncorrupted then the shared secret should remain unknown to the adversary, as long as no uncorrupted player has started the reconstruction protocol. We say that a secret sharing protocol is verifiable if the following requirements hold: as soon as *one* uncorrupted player

[2] An alternative convention may be to invoke the corresponding sub-protocol immediately upon the receipt of the message. We prefer the convention that sub-protocols must be explicitly invoked by other sub-protocols.

[3] Here and in the sequel the probabilities are taken over the random inputs of the players.

successfully completes the sharing protocol, a value r is fixed; furthermore, *all* the uncorrupted players will eventually successfully complete the sharing protocol. The value reconstructed by the players in the reconstruction protocol will be r, even if the dealer is corrupted. In Definition 2 we formalize the requirements from a verifiable secret sharing protocol for our asynchronous setting.

More precisely, let a Share-Reconstruct (SR) protocol be a protocol that consists of two sub-protocols, denoted Sh and Rec. It will be convenient to regard these sub-protocols as independent sub-protocols. Also, there is a distinguished player called the dealer and denoted D. An execution of an SR protocol proceeds as follows:

1. A player first runs sub-protocol Sh on its given input. (The dealer has input value $s \in S$ where S is some pre-defined set. All other players have no input.) At a certain point, protocol Sh may output a special symbol `` `sharing succeeded´ ``. In the sequel, when we colloquially say that protocol Sh completes, we mean that the player outputs `` `sharing succeeded´ ``. We stress that sub-protocol Sh remains active even after this symbol has been output.

2. Once a special symbol `` `start running Rec´ `` appears on the player's input tape, the player invokes sub-protocol Rec. From this point on, sub-protocols Sh and Rec have a common state; in particular, all the local variables are common to the two sub-protocols. In particular, Sh completes when Rec does. The output of the SR protocol is the output of Rec. (We stress that a party can start running Rec even before it has output `` `sharing succeeded´ ``.)

In the sequel we consider also a more complex situation where a single Sh sub-protocol is followed by several invocations of Rec, which share the same local state. The interpretation is the same as here; we elaborate within.

In order to formalize the secrecy requirement of Definition 2 we need the following notion. The adversary view of an execution is the entire information available to the adversary regarding that execution. This information includes the identities, inputs and internal information of the corrupted players, the messages received by the corrupted players, the frames of the messages sent by the uncorrupted players, and the sequence of activations of the players. (It is convenient to organize this information in events, that correspond to the events of an execution. Each event contains the message delivered and the frames of the messages sent; messages sent to corrupted parties appear in full.)

Definition 2 *An n-player SR protocol* $\langle \mathrm{Sh}, \mathrm{Rec} \rangle$ *is* $(1-\epsilon)$-correct, t-resilient AVSS *if the following requirements hold, for every t-adversary. Let the dealer's input for* Sh *is a value $s \in S$ for some finite globally known set S. Then:*

• **Completion.** *With probability $1-\epsilon$ the following requirements hold.*

1. If the dealer is uncorrupted throughout protocol Sh *then all uncorrupted players eventually output a special symbol* `` `sharing succeeded´ ``

2. If some uncorrupted player outputs `` `sharing succeeded´ `` *then all uncorrupted players eventually output* `` `sharing succeeded´ ``.

3. If the uncorrupted players output `` `sharing succeeded´ `` *and all uncorrupted players have the* `` `start running Rec´ `` *on their input tapes, then all the uncorrupted players eventually complete both protocols* Sh *and* Rec.

• **Correctness.** *Once some uncorrupted player outputs* `` `sharing succeeded´ ``, *there exists a* fixed *value, r, such that the following requirements hold with probability $1-\epsilon$:*

1. Each uncorrupted player outputs r upon completion of Rec*. (Namely, r is the reconstructed secret.)*

2. If the dealer is uncorrupted throughout protocol Sh *then r is the shared secret, i.e. $r = s$.*

• **Secrecy.** *Let* $\text{VIEW}(s)$ *denote the random variable (over the random inputs of the players) describing the adversary view of an execution of* Sh *where the dealer has input s. If the dealer is uncorrupted throughout then the distribution of* $\text{VIEW}(s)$ *is the same for all values $s \in S$.*

Remarks: 1. Definition 2 does not require a player to recognize a situation where the sharing of a secret has failed. In other words, an uncorrupted player is in one of two "states": either it knows that the sharing has succeeded, or it does not yet know.

This characteristic of the definition, although seemingly unnatural, appears to be essential for our needs. A definition that requires uncorrupted players to always complete protocol Sh with a success/failure decision is a considerably stronger one. In fact, this stronger definition trivially implies Byzantine agreement, thus it seems unfit for primitives used to construct a Byzantine agreement protocol.

2. Another characteristic of the definition is that protocol Sh is not completed until protocol Rec completes. This is needed in our protocols since a player will have to keep participating in Sh even after it outputs `` `sharing succeeded´ ``, in order to help other players to output `` `sharing succeeded´ ``as well.

3 Overview of the protocols

First, let us state our main results.

Theorem 1 (AVSS). *Let $n \geq 3t+1$. For every $\epsilon > 0$ there exists a $(1-\epsilon)$-correct, t-resilient AVSS protocol for n players. Conditioned on the event that the uncorrupted players complete, they do so in constant time. Furthermore, the computational resources required of each player are polynomial in n and $\log \frac{1}{\epsilon}$.*

Theorem 2 (BA). *Let $n \geq 3t+1$. For every $\epsilon > 0$ there exists a $(1-\epsilon)$-completing, t-resilient, asynchronous Byzantine Agreement protocol for n players. Conditioned on the event that the uncorrupted players complete, they do so in constant expected time. Furthermore, the computational resources required of each player are polynomial in n and $\log \frac{1}{\epsilon}$.*

Our Byzantine Agreement protocol is complex and involves many layers. To facilitate the reading we first present a "top-down" overview of our protocol. Let $\mathbf{F}$ be a field of size greater than n. All the computations in the sequel are done in $\mathbf{F}$. The BA protocol employs the idea of using 'common coins' to reach agreement, as follows.

BA using Common Coin. This part of our protocol follows the constructions of Rabin, Bracha, Ben Or, Feldman-Micali, and Feldman [Ben83, Rab83, Bra84, FM88, Fel89]. The protocol proceeds in rounds. In each round, each player has a 'modified input' value. In the first round, the modified input of each player is its local input. In each round the players invoke two protocols, called Vote and Common Coin. Protocol Common Coin has the following property. Each player has a random input, and binary output. For every value $\sigma \in \{0,1\}$, with probability at least $\frac{1}{4}$ *all* the uncorrupted players output σ. In protocol Vote each player tries to establish whether there exists a 'distinct majority' for some value amongst the players' modified inputs of this round (we define distinct majority in the sequel). If a player recognizes a 'distinct majority' for some value it takes this value to be its modified input for the next round. Otherwise, it sets its modified input for the next round to be the output of protocol Common Coin. We show that no two uncorrupted players ever recognize a distinct majority for two different values. This is used to show that in each round, with probability at least $\frac{1}{4}$ all players have the same modified input. Furthermore, if all the uncorrupted players have the same modified input in some round, then all the uncorrupted players will recognize this and complete outputting this value. It follows that, conditioned on the even that all the uncorrupted players complete all the Vote and Common coin protocols they invoke, the BA protocol completes within a constant expected number of rounds.

Protocol Vote is deterministic and always completes in constant time. Except for an event that occurs with negligible probability, all the invocations of Common Coin complete in constant time. Consequently, except for negligible probability, each round of the BA protocol is completed within constant time.

Common Coin using AVSS. Our construction follows Feldman, and Feldman-Micali [Fel89, FM88]. The protocol proceeds roughly as follows. First, each player shares n random secrets using our AVSS protocol. Once a player has completed a large enough number out of the n^2 invocations of the sharing protocol (thus enough values-to-be-reconstructed have been fixed) to ensure sufficient randomness in the outcome of the global coin, it invokes the corresponding reconstruction protocols. Once all the relevant secrets are reconstructed, the player locally computes its output based on the reconstructed secrets.

AVSS from scratch. Our AVSS protocol is constructed in three 'layers'. Each layer consists of a different secret sharing protocol (with an allowed-error parameter, ϵ). The protocol of the lowest layer is called Asynchronous Recoverable Sharing (A-RS). The next protocol is called Asynchronous Weak Secret Sharing (AWSS). The last ('top') layer is an AVSS protocol (as in Definition 2). Each protocol is used as a building block for the next. All three sharing protocols satisfy the *completion* and *secrecy* requirements of Definition 2. In all three protocols, if the dealer is uncorrupted then the uncorrupted players always reconstruct the secret shared by the dealer. The correctness property for the case that the dealer is corrupted is upgraded from A-RS to AWSS and finally to AVSS:[4]

A-RS- Once an uncorrupted player completes the reconstruction protocol, a subset $S \subseteq \mathbf{F}$, of size $2t+1$, is fixed. With probability $1-\epsilon$, each uncorrupted player will reconstruct a value $r \in S \cup \{null\}$. (We stress that it is not required that all uncorrupted players output the same reconstructed value.)

AWSS- Once an uncorrupted player completes the sharing protocol, a value s is fixed. With probability $1-\epsilon$, each uncorrupted player will reconstruct either s or *null.*

AVSS- Once an uncorrupted player completes the sharing protocol, a value s is fixed. With probability $1-\epsilon$, each uncorrupted player will reconstruct s.

[4]As usual, the probabilities are taken over the entire random inputs of the players.

4 Tools

4.1 Information Checking Protocol- ICP

The Information Checking Protocol (ICP) is a tool for authenticating messages in the presence of computationally unbounded corrupted players. The ICP was introduced by T. Rabin and Ben-Or [Rab94, RB89]. For self-containment, we state the properties and constructions of the ICP.

The protocol is executed by three players: a dealer D, an intermediary T, and a receiver R. The dealer hands a secret value s over to T. At a later stage, T is required to hand this value over to R, and to convince R that s is indeed the value which T received from D. More precisely, the protocol is carried out in three phases:

Generation(s) is initiated by D. In this phase D hands the secret s to T and some auxiliary data to both T and R.

Verification is initiated by T and carried out by T and R. In this phase T tries to predict (for itself) whether in the Authentication phase, R will accept s, the secret held by T. If it predicts that R will (resp. will not) accept s he sets **Ver**$= s$ (resp., *null*).

Authentication is initiated by T and carried out by T and R. In this phase R receives a value s' from T, along with some auxiliary data, and either outputs or rejects it. We denote the acceptance, (resp., rejection) by **Auth**$= s'$ (resp., *null*).

The ICP has the following properties, with an allowed error parameter ϵ.

Correctness:

1. If D, T and R are uncorrupted, and D has a secret s, then **Ver**$= s$ and **Auth**$= s$.[5]

2. If T and R are uncorrupted, and T has set **Ver**$= s$, then with probability $(1-\epsilon)$, R will output s in the Authentication phase.

3. If D and R are uncorrupted, and the dealer has a secret s, then the probability that R outputs a value s' in the Authentication phase, such that $s' \neq s$ is at most ϵ.

Secrecy:

4. The information that D hands R in the Generation phase is distributed independently of the secret s. (Consequently, if D and T are uncorrupted, and T has not executed the Authentication phase, R has no information about the secret s.)

For self containment we sketch the [Rab94, RB89] construction in Figure 1.

4.2 Broadcast

The asynchronous broadcast primitive we use (denoted a-cast) was introduced and elegantly implemented by Bracha [Bra84].

Definition 3 *Let π be an n-player protocol initiated by a special player (called the sender), having input m (the message to be broadcast). Protocol π is a t-*resilient asynchronous broadcast* protocol if the following requirements hold, for every input, and every t-adversary.*

[5]Here and in the sequel, statements of the form "with probability p, if <condition> then <event>" should be interpreted as follows: For any input for the players, and with any adversary, the event that <condition> occurs and <event> does not occur has probability at most p, where the probability is taken over the random inputs of the players.

ICP$[\epsilon]$

Set k such that $\epsilon \geq \frac{k}{2^k}$. Let $\mathbf{F}$ be a field with $|\mathbf{F}| > 2^k$.

Generation phase (Gen):

1. The dealer D, having a secret $s \in \mathbf{F}$, chooses random values $y_1, ..., y_{2k}$ and $b_1, ..., b_{2k}$ uniformly distributed in $\mathbf{F}$, and computes $c_i \triangleq b_i s + y_i$ for $1 \leq i \leq 2k$.
2. D sends s and $\vec{y}_s \triangleq y_1, ..., y_{2k}$ to T .
3. D sends the check vectors $(b_1, c_1), ..., (b_{2k}, c_{2k})$ to R.

Verification (Ver):

1. T chooses k distinct random indices $\mathcal{I} \triangleq \{i_1, ..., i_k\}$, $1 \leq i_j \leq 2k$, and asks R to reveal (b_{i_1}, c_{i_j}) for all $i_j \in \mathcal{I}$.
2. R reveals these values. The remaining indices will be the *unrevealed indices.*
3. For each one of the revealed indices $i_j \in \mathcal{I}$, T tests whether $s \cdot b_{i_j} + y_{i_j} = c_{i_j}$. If *all* k indices satisfy this requirement then T sets **Ver**$= s$. Otherwise, it sets **Ver**$=$ *null.*

Authentication (Auth):

1. T sends to R the secret value s and y_{i_j} for $1 \leq i_j \leq 2k$, $\ i_j \notin \mathcal{I}$ (i.e. the unrevealed indices).
2. If there exists an unrevealed index i_j which satisfies $s \cdot b_{i_j} + y_{i_j} = c_{i_j}$ then R sets **Auth**$= s$, otherwise **Auth**$=$*null.*

Figure 1: The Information Checking Protocol of [Rab94, RB89]

• **Termination:**

1. If the sender is uncorrupted then each uncorrupted player will eventually complete the protocol.

2. If any uncorrupted player completes the protocol, then all the uncorrupted players will eventually complete the protocol.

• **Correctness:** *All the uncorrupted players who have terminated have identical outputs. Furthermore, if the sender is uncorrupted then all players output m.*

We stress that the Termination property of a-cast is much weaker than the Termination property of BA: for a-cast, we do not require that the uncorrupted players complete the protocol when the sender is corrupted.

Conventions: In the sequel, we say that a player P_i 'a-casts a message m', meaning that P_i invokes a-cast as the sender with input m. We also say that player P_j completes the a-cast of player P_i with value m, meaning that P_j completes the execution of P_i's a-cast and has locally set the value of the broadcast to m. Furthermore, we often do not explicitly mention that a player invokes a-cast as a regular player. It can be easily deduced when a player invokes an a-cast. We also assume that the identity of the sender, as well as an invocation-identifier, appear in m.

For self-containment, we present a slight modification of Bracha's a-cast protocol in Figure 2, and prove its correctness.

Protocol A-cast

Code for the sender (on input m):

1. send message $(\mathtt{MSG}, m)$ to all the players.

Code for player P_i:

2. Wait until receiving a message $(\mathtt{MSG}, m)$. Then send $(\mathtt{ECHO}, m)$ to all the players.
3. Wait until receiving $n - t$ messages $(\mathtt{ECHO}, m')$ that agree on the value of m'. Then send $(\mathtt{READY}, m')$ to all the players.
4. Wait until receiving $t + 1$ messages $(\mathtt{READY}, m')$ that agree on the value of m'. Then send $(\mathtt{READY}, m')$ to all the players.
5. Wait until receiving $2t + 1$ messages $(\mathtt{READY}, m')$ that agree on the value of m'. Then output m' and complete the protocol.

Figure 2: Bracha's broadcast protocol(sightly modified)

Lemma 1 *Protocol A-cast is an $(\lceil \frac{n}{3} \rceil - 1)$-resilient asynchronous broadcast protocol. Furthermore, the protocol has constant running time.*

Proof: *Termination:* If the sender is uncorrupted then all players will complete steps 2—5 and complete the protocol. If an uncorrupted player has completed the protocol then it has received $n - t$ $(\mathtt{READY}, m')$ messages. At least $t + 1$ of those come from uncorrupted players. Thus all uncorrupted players will receive $t+1$ $(\mathtt{READY}, m')$ messages, and will execute the instruction of Step

4. Consequently, all uncorrupted players will receive $n-t$ $(\texttt{READY}, m')$ messages and complete the protocol.

Correctness: If the sender is uncorrupted (with input m) then at most t players will send $(\texttt{ECHO}, m')$ with $m' \neq m$. Thus no uncorrupted player will send $(\texttt{READY}, m')$, and no uncorrupted player will output m'. Now, assume that an uncorrupted player has output some value m. It follows that at least $t+1$ uncorrupted players have sent $(\texttt{READY}, m)$. These players have *not* sent $(\texttt{ECHO}, m')$ for $m' \neq m$. Thus, at most $n-t-1$ players have sent $(\texttt{ECHO}, m')$, and no uncorrupted player has sent (in Step 3) $(\texttt{READY}, m')$ with $m' \neq m$. Consequently, at most t players have sent $(\texttt{READY}, m')$, and no uncorrupted player outputs m'. □

5 Asynchronous Recoverable Sharing — A-RS

Unlike synchronous systems, in an asynchronous system it is not possible to decide whether a player whose messages do not arrive is corrupted or just slow. As argued in the introduction, this difficulty causes known techniques for AVSS to fail when $n \leq 4t$. We overcome this difficulty by using the Information Checking Protocol (ICP) described in the previous section. It will aid us in constructing a 'weakened version' of an AVSS protocol. This primitive, called Asynchronous reconstructible sharing (A-RS), has a "synchronizing effect" on the network, in the sense that it guarantees a bounded wait for receiving values, *even from corrupted players*. A-RS is a Share-Reconstruct protocol. Very roughly, it will have the property that if the sharing has been completed then there is a guarantee that eventually *some* value will be reconstructed by each proper player, not necessarily the same one. Even though A-RS does not seem to offer much in ways of consistency, it does provide a great service as it eliminates the question of whether to wait for a value from a player or not.

More formally: an A-RS protocol satisfies the *termination* and *secrecy* requirements of AVSS (Definition 2). Also, if the dealer is uncorrupted then the uncorrupted players always reconstruct the secret shared by the dealer. The difference from AVSS lies in the case where the dealer is corrupted. For A-RS we only require the following:

Correctness of A-RS:

1. *Uncorrupted dealer: All the uncorrupted players output the value s which is the dealer's shared secret.*
2. *Corrupted dealer: Once the first uncorrupted player completes the protocol, a subset $S \subseteq \mathbf{F}$, of size $2t+1$, is defined. With probability $1-\epsilon$, each uncorrupted player will reconstruct a value $r \in S \cup \{null\}$. (We stress that it is not required that all uncorrupted players output the same value.)*

We describe our construction. Basically, we use Shamir's classic secret sharing protocol [Sha79] while using ICP to verify the shares. This way, we can make sure that the following two properties hold simultaneously (with overwhelming probability): (a) the shares of at least $t+1$ uncorrupted players will always be available at reconstruction, and (b) if the dealer is uncorrupted then all the shares available at reconstruction are the originally dealt shares. A more detailed description of our construction follows.[6]

[6]Here, as well as in all subsequent secret sharing protocols, we use the standard convention that the dealer, in addition to executing his specific code, also participates as one of the players (and will eventually have a share of the secret). Naturally, the dealer and the player are one entity, that is if the adversary corrupts the dealer (resp.

Sharing protocol. The dealer, having a secret s, chooses values $a_1, ..., a_t \in_R \mathbf{F}^t$, and defines the polynomial $f(x) = a_t x^t + ... + a_1 x + s$.[7] (We call this process choosing a random polynomial $f(x)$ for s.) Next, for each i, the dealer sends $\phi_i \triangleq f(i)$ to P_i. We say that ϕ_i is P_i's share of s. In addition, for each share ϕ_i and for each player P_j, the dealer executes the Generation phase of ICP (described in Section 4.1), with player P_i as the intermediary, and player P_j as the receiver. The ICP protocol will have the appropriate allowed error parameter ϵ' (we set $\epsilon' = \frac{\epsilon}{n^2}$). We denote this execution of the Generation phase by $\mathbf{Gen}_{D,i,j}[\epsilon'](\phi_i)$. (In the sequel, we omit the parameter ϵ' from the notation.) Next, player P_i, as T , initiates the Verification phase of ICP with P_j, as R, denoted $\mathbf{Ver}_{i,j}$. Success of $\mathbf{Ver}_{i,j}$ assures P_i that, with probability $1 - \epsilon'$, P_j will later output its share ϕ_i. Once P_i successfully (i.e. with non-null output) completes $2t+1$ invocations of $\mathbf{Ver}_*$, it a-casts `(OK,` P_i`)` . A player outputs `‘sharing succeeded’` and completes the sharing protocol upon completing $2t + 1$ a-casts of the form `(OK,` P_k`)` . (We remark that in the case of A-RS a player can in fact complete the sharing protocol before the reconstruction protocol ends. Yet for uniformity with later constructions a player completes the sharing protocol only when the reconstruction protocol is completed.) Our protocol, denoted A-RS Share, is presented in Figure 3.

Protocol A-RS-Share[ϵ]

Code for the dealer, with secret s: Let $\epsilon' = \frac{\epsilon}{n^2}$.

1. Choose a random polynomial, $f(x)$, for s.
2. Set $\phi_i \triangleq f(i)$ for $1 \le i \le n$.
   Let $\epsilon' \triangleq \frac{\epsilon}{n^2}$. Execute $\mathbf{Gen}_{D,i,j}[\epsilon'](\phi_i)$ for $1 \le i, j \le n$, with player P_i as T and player P_j as R.

Code for player P_i:

3. For all $j \in [n]$: Wait until $\mathbf{Gen}_{D,i,j}[\epsilon']$ is completed; then, initiate as T $\mathbf{Ver}_{i,j}[\epsilon']$ with player P_j as R.
4. Wait until $\mathbf{Ver}_{i,j}[\epsilon']$ completes with a non-null output for $2t + 1$ players P_j. Then a-cast `(OK,` P_i`)` .
5. Wait until the a-casts of `(OK,` P_k`)` is completed for $2t + 1$ different players P_k then raise the flag `‘sharing succeeded’`.

Figure 3: The A-RS sharing protocol

Definition 4 *Let* $\mathbf{F}$ *be a field, and let* $r \ge t + 1$. *A set* $\{(i_1, \phi_{i_1}), \ldots, (i_r, \phi_{i_r})\}$ *where* $\phi_{i_j} \in \mathbf{F}$ *is said to* define a secret s *if there exists a (unique) polynomial* $f(x)$ *of degree at most* t, *such that* $f(0) = s$, *and* $f(i_j) = \phi_{i_j}$, *for* $1 \le j \le r$. *(We shall interchangeably say that the* shares $\phi_{i_1}, \ldots, \phi_{i_r}$ *define the polynomial* $f(x)$.*)*

player) then the player (resp. dealer) is corrupted as well, and this will constitute one corruption on the part of the adversary. As seen in the analyses of the protocols, this does not affect the allowed number of corruptions. (Intuitively, if the dealer is not corrupted then nothing is lost by having the dealer double up as a player. If the dealer has been corrupted then all secrecy issues are mute since the secret itself is known to the adversary.)

[7] We let $e \in_R D$ denote an element e chosen uniformly at random from domain D.

Reconstruction protocol. First, player P_i initiates an a-cast of its share, ϕ_i. It then initiates the Authentication phase of ICP as T , with every other player, P_j as R, with respect to the share ϕ_i which it holds. We denote this execution by $\mathbf{Auth}_{i,j}$. Furthermore, if player P_j initiated $\mathbf{Auth}_{j,i}$ then player P_i participates in **Auth** as R with player P_j as T . For each such invocation $\mathbf{Auth}_{j,i}$ where P_i outputted ϕ_j, it a-casts (P_i `authenticates` P_j) . In this case we shall say that player P_i has authenticated player P_j.

Player P_i considers a share, ϕ_j, legal, if P_j has been authenticated by at least $t+1$ players. Once P_i has $t+1$ legal shares, he computes the secret which they define, and a-casts it. (Checking whether a given set of shares define a secret, and computing this secret, can be done efficiently using interpolation.) If there exists a value which is a-casted by at least $n-2t$ players, then P_i output this value. Otherwise, P_i outputs *null*. (This extra a-cast is required to limit the size of the set of possible values reconstructed by the different players.) The reconstruction protocol, denoted A-RS-Rec, is presented in Figure 4.

Protocol A-RS-Rec$[\epsilon]$

Code for player P_i:

1. A-cast your share ϕ_i.
2. Invoke $\mathbf{Auth}_{i,j}[\epsilon']$ with every player P_j, where P_i plays the role of T and P_j plays the role of R. (The invocation $\mathbf{Auth}_{i,j}[\epsilon']$ corresponds to the invocation $\mathbf{Ver}_{i,j}[\epsilon']$ in Step 3 of A-RS-Share.)
3. If the a-cast of P_j's share is completed, with value ϕ'_j, then participate as R in P_j's initiation of $\mathbf{Auth}_{j,i}$. If $\mathbf{Auth}_{j,i} = \phi'_j$, then a-cast ($P_i$ `authenticates` P_j) .
4. Consider a share ϕ'_j legal if P_j's a-cast of his share from Step 1 was completed with output ϕ'_j, and at least $t+1$ a-casts of the form (P_k `authenticates` P_j) have been completed. Create a set called IS_i (standing for Interpolation Set).

   For each $j \in [n]$, wait until ϕ'_j becomes legal. Then, add ϕ'_j to IS_i.
5. Wait until $|IS_i| = t+1$. Then, interpolate the polynomial defined by IS_i. Let s denote the constant term of the polynomial. A-cast (P_i `suggests secret` s) .
6. Wait until completing $n-t$ (P_* `suggests secret` s') a-casts of different players.
   If there is a value, s', which appears in at least $n-2t$ such a-casts, then output s'. Otherwise, output *null*. Complete the execution of A-RS-Share, raise the flag \``sharing succeeded`´ and A-RS-Rec.

Figure 4: The A-RS reconstruction protocol

Theorem 3 *Let* $n \geq 3t+1$*. Then for every* $\epsilon > 0$*, the pair* (A-RS-Share[ϵ],A-RS-Rec[ϵ]) *is a* $(1-\epsilon)$*-correct,* t*-resilient A-RS protocol for* n *players.*

Proof: Fix a t-adversary. We show that the Completion and Secrecy requirements of Definition 2 (AVSS) are met, as well as the Correctness property for A-RS stated above.

Completion (1): When the dealer is uncorrupted, the Verification phase of ICP between every two uncorrupted players will terminate with non-null output (see the definition of ICP on page 12).

Therefore, each uncorrupted player will have $2t + 1$ successful verifications for its share ϕ_i. Thus, each uncorrupted player P_i will initiate an a-cast of `(OK,` P_i`)`. Consequently, each uncorrupted player will complete participation in $2t + 1$ a-casts of the form `(OK,` P_j`)`, and terminate protocol A-RS-Share.

Completion (2): If one uncorrupted player has outputed `` `sharing succeeded' ``, then it has completed participation in $2t+1$ `(OK,` P_j`)` a-casts. By the properties of a-cast (Definition 3), every uncorrupted player will complete these $2t + 1$ a-casts, and will thus output `` `sharing succeeded' ``.

For the rest of the proof, let E be the event that no errors occur in all the invocations of ICP. (That is, all the requirements listed in Section 4 hold.) We know that event E occurs with probability at least $1 - n^2\epsilon' = 1 - \epsilon$.

Completion (3): If some uncorrupted player, P_i, has outputed `` `sharing succeeded' ``, then $2t+1$ players have a-casted `(OK, ...)` in Step 3 of A-RS-Share. Out of these a-casts, at least $t+1$ have originated with uncorrupted players. Let G_i be this set of at least $t + 1$ uncorrupted players, relative to player P_i.

For every uncorrupted player $P_j \in G_i$ the following holds: P_j a-casted `(OK,` P_j`)` in A-RS-Share since it had complete $2t + 1$ invocations of **Ver** with non-null output. That is, $\mathbf{Ver}_{j,j_l} \neq \mathit{null}$ for players in $A_j \stackrel{\triangle}{=} \{P_{j_1}, ..., P_{j_{2t+1}}\}$. Furthermore, as P_j is uncorrupted every player in A_j will complete the a-cast of P_j's share ϕ_j in the first step of A-RS-Rec. Out of the players in A_j at least $t + 1$ are uncorrupted, and due to the properties of ICP, these uncorrupted players will complete **Auth** and output the value ϕ_j, which in return would cause them to initiated an a-cast of the form `(`P_{j_l} `authenticates` P_j`)`. Thus, the share of player P_j will be considered legal by each uncorrupted player. Therefore every uncorrupted player will have $t + 1$ legal shares in Step 3 of A-RS-Rec and will suggest some secret. Consequently, each uncorrupted player will have $n - t$ suggested secrets, and will complete protocol A-RS-Rec.

Note that all the Completion properties hold with no probability of error. Furthermore, both A-RS-Share and A-RS-Rec complete in constant time.

Correctness (1): If the dealer is uncorrupted and event E occurs, then each share ϕ_j in the set IS_i of each uncorrupted P_i, has originated with D (namely, $\phi_j = f(j)$) due to the propertiesof ICP. Therefore, the shares in IS_i define the secret, s, shared by the dealer, and P_i will a-cast `(`P_i `suggests secret` s`)`. Consequently, each uncorrupted player will complete $n - t$ such a-casts, out of which at least $n - 2t$ suggest the secret s. Hence, each uncorrupted player will output s.

Correctness (2): Let P_i be the first uncorrupted player to complete $n - t$ a-casts of the form `(`P_k `suggests secret ...)`. Let S denote the (multi)set of secrets suggested in these $n - t$ a-casts. Now, consider another uncorrupted player, P_j that has a non-null output. At least $n - 2t$ out of the $n - t$ a-casts completed by P_j in Step 5 of A-RS-Rec have the same output value, denoted v. It follows that at least $n - 3t$ out of the values in S equal v. Consequently, there are at most $1 + (n - t) - (n - 3t) = 2t + 1$ different values in S, and no uncorrupted player outputs a non-null value that is not in S.

Secrecy: Informally, as long as the dealer is uncorrupted and no uncorrupted player has initiated protocol A-RS-Rec, then any set of $t' \leq t$ players have only t' shares of the secret, along with the data received during the ICP Generation and Verification Protocols with respect to the other players' shares of the secret. The secrecy properties of ICP and Shamir's secret sharing protocol imply that the shared secret is independent of this data.

More precisely, recall the definition of the adversary view from Section 2.1. For concreteness, assume that the view consists of a sequence of events, where each event contains the information gathered by the adversary either in an atomic step (ie, in an activation of some player), or in corrupting some player. Let $v^{(i)}(\mathcal{A}, s, \vec{r})$ denote the first i events in the view of adversary $\mathcal{A}$ of an execution of protocol A-RS-Share with input secret s and random inputs $\vec{r} = r_1, ..., r_n$ for the players. Let $\nu^{(i)}(\mathcal{A}, s)$ denote the random variable having the distribution of $v^{(i)}(\mathcal{A}, s, \vec{r})$ where R is uniformly chosen, conditioned on the event that the dealer remains uncorrupted throughout.

We show by induction on i that the distribution of $\nu^{(i)}(\mathcal{A}, s)$ is the same for all $s \in \mathbf{F}$. The induction base ($i = 0$) is trivial. For the induction step, let $i \geq 0$ and fix some value v in the support of $\nu^{(i)}(\mathcal{A}, 0)$. (That is, v is a sequence of the first i events in an execution of A-RS-Share. By the induction hypothesis the support of $\nu^{(i)}(\mathcal{A}, s)$ for all $s \in \mathbf{F}$ is the same.) Let $\xi_s^v(i+1)$ be a random variable having the distribution of the $(i+1)$st event in an execution with secret s, given that the first i events agree with v. We complete the proof by demonstrating that for any $s, s' \in \mathbf{F}$ the random variables $\xi_s^v(i+1)$ and $\xi_{s'}^v(i+1)$ are equally distributed.

We distinguish two cases: **(I).** $\xi_s^v(i+1)$ describes the activation of some player on some message delivered by $\mathcal{A}$. In this case $\xi_s^v(i+1)$ contains the identity of the activated player, the frame of the delivered message, and the frames of the messages sent by this player. For messages sent to and from corrupted players the body of the message is also included. It follows from the induction hypothesis that the distribution of the delivered message, as well as the identity of the activated player, is the same in $\xi_s^v(i+1)$ and $\xi_{s'}^v(i+1)$. It can be seen by inspecting the protocol that the frames of the messages sent by the activated player are uniquely determined by v and the delivered message. In fact, the only variable parts are the field elements in the messages sent to the corrupted players. These values are up to t shares of a random polynomial of degree t with some fixed free coefficient, as well as random field elements used in ICP. These field elements are uniformly and independently distributed, regardless of the value of the secret.

(II). $\xi_s^v(i+1)$ describes the corruption of some player P_j. In this case $\xi_s^v(i+1)$ contains the identity of P_j, and the internal memory contents of P_j. Also here, it follows from the induction hypothesis that j, the identity of the corrupted player, has the same distribution in $\xi_s^v(i+1)$ and $\xi_{s'}^v(i+1)$. The memory contents of P_j is uniquely determined by v, except for the share ϕ_j and the field elements pertaining to the invocations of ICP. Again, these values are uniformly distributed regardless of the value of the secret. □

6 Asynchronous Weak Secret Sharing — AWSS

We construct a secret sharing protocol called Asynchronous Weak Secret Sharing (AWSS). An AWSS protocol satisfies the *termination* and *secrecy* requirements of AVSS (Definition 2 on page 9). The correctness requirement is as follows. If the dealer is uncorrupted then the uncorrupted players always reconstruct the secret shared by the dealer, as in AVSS. The difference from AVSS lies in the case where the dealer is corrupted. For AWSS we only require that:

Correctness of AWSS: *Once some uncorrupted player completes protocol* Sh, *there exists a* fixed *value,* $r \in \mathbf{F} \cup \{null\}$, *such that the following requirements hold:*

1. *If the dealer is uncorrupted throughout protocol* Sh *then* r *is the shared secret, i.e.* $r = s$, *and each uncorrupted player outputs* r *at the end of the reconstruction protocol.*

2. *If the dealer is corrupted then each uncorrupted player outputs either* r *or null upon completing*

the reconstruction protocol.

Remarks:

- We stress that even if $r \neq$ *null* then some of the uncorrupted players may output r and some may output *null.* The adversary can decide, during the execution of the reconstruction protocol, which players will output *null.*
- Our construction of A-RS, described in the previous section, does not satisfy the requirements of an AWSS protocol. There, in case that the dealer is corrupted, the adversary has the power to decide, *at the reconstruction stage,* which value each uncorrupted player reconstructs (out of the predefined set of size $2t+1$).

Our construction. Our construction follows the synchronous WSS protocol of [Rab94, RB89]. The idea is to make sure that, at the end of the sharing protocol, each player P_i will have an interpolation set, IS_i, which constitutes of players whose shares will be available in the reconstruction stage. (The availability of these shares is guaranteed using A-RS, as described below.) The interpolation sets of every two uncorrupted players P_i and P_j will have an intersection of at least $t+1$ players. This way, if the shares of the players in the interpolation sets of P_i and P_j define some secret, then they define the same secret.

In the reconstruction stage the shares of all the players are jointly reconstructed. Next, each player computes and outputs the secret defined by the shares of the players in its interpolation set. If these shares do not define a secret then the player outputs *null.*

Sharing protocol. The dealer, sharing a secret $s \in \mathbf{F}$, chooses a random polynomial $f(x)$ of degree t for s. For each i, the dealer sets $\phi_i \triangleq f(i)$. In addition, for each share ϕ_i and for each player P_j, the dealer executes the generation phase of ICP with player P_i as the intermediary, T, and player P_j as the Receiver. (This part is the same as in A-RS-Share.)

Next, each player shares its share of the secret and all the values which it got as T in the ICP-Generation Protocol, using A-RS-Share. Furthermore, it shares using A-RS-Share the values which it got as R , in each of the invocations of ICP-Gen. (Each of these values is shared using a different invocation of A-RS-Share.) Player P_i creates a variable C_i. (C_i will contain a set of players). Once P_i completes all the A-RS-Shares of P_j's values, P_j is added to C_i. Next player P_i initiates the ICP-Verification Protocol for ϕ_i with each player $P_j \in C_i$, and with the appropriate allowed-error parameter, ϵ'. (We denote this execution by $\mathbf{Ver}_{i,j}[\epsilon''](\phi_i)$).

Let A_i be P_i's set of players P_j such that $\mathbf{Ver}_{i,j}[\epsilon''](\phi_i) = \phi_i$. Once $|A_i| = 2t+1$, player P_i a-casts A_i. (This *acceptance set,* A_i, is the set of players who will later accept ϕ_i with high probability.)

Let $\mathcal{E}_i$ be P_i's set of players $P_j \in C_i$ whose a-cast of A_j has been completed, and $A_j \subseteq C_i$. Once $|\mathcal{E}_i| \geq 2t+1$, player P_i a-casts $IS_i \triangleq \mathcal{E}_i^{(2t+1)}$. (This *eligibility set,* $\mathcal{E}_i$, is the set of players whose share will be later considered either legal or *null.* The *interpolation set,* IS_i, is the set of players whose shares will later define the secret associated with P_i.)

Let F_i be P_i's set of players P_j whose IS_j a-cast has been completed, and $IS_j \subseteq \mathcal{E}_i$. Once $|F_i| = n-t$, player P_i completes the sharing protocol. (This *final set,* F_i, is the set of players, with whom P_i will later associate a secret.)

Reconstruction protocol. Initially, each player P_i reconstructs, using A-RS-Share, the share of player $P_j \in \mathcal{E}_i$ and the values which P_j received as T in ICP. Once this reconstruction is completed for every player $P_l \in A_j$ the values which P_l received as R with respect to P_j are reconstructed using A-RS-Rec.

For each player $P_j \in \mathcal{E}_i$ and for each player $P_l \in A_j$, once all the necessary values are reconstructed, P_i computes P_l's expected result of $\mathbf{Auth}_{j,l}[\epsilon'']$ (by 'doubling up' as P_l). Next, P_i associates a value with player P_j as follows. If P_i finds out that $t+1$ players in A_j have outputted $\mathbf{Auth}_{j,l}[\epsilon''] = \phi_j$ then P_i associates ϕ_j with P_j. (We then say that ϕ_j is *legal*). Otherwise, P_i associates the value "null" with P_j.

Now, for each player $P_k \in F_i$, once the values associated with all the players in IS_k are computed, if all the legal shares in IS_k define a secret, s, then P_i considers s to be the secret *suggested* by P_k. Otherwise, the secret suggested by P_k is *null.*

Once the values shared by all the players in $\mathcal{E}_i$ are reconstructed, P_i computes the secrets suggested by all the players in F_i. (Again, this is done by 'doubling up' as each player in F_i.) If all the players in F_i suggest the same secret, s, then P_i outputs s. Otherwise, P_i outputs *null.* (Equivalently, P_i can output the secret defined by the reconstructed shares of the players in $\cup_{P_j \in F_i} IS_j$. If these shares do not define a secret then P_i outputs *null.*)

The AWSS-Share and AWSS-Rec are described in Figures 5 and 6 respectively.

Theorem 4 *Let $n \geq 3t+1$. Then for every $\epsilon > 0$, the pair* (AWSS-Share[ϵ],AWSS-Rec[ϵ]) *is a $(1-\epsilon)$-correct, t-resilient AWSS protocol for n players.*

As in the proof of Theorem 3, let E be the event that no errors occur in all the invocations of ICP. (That is, all the requirements listed in Section 4.1 hold with no probability of error.) In proving Theorem 4 we assume that event E occurs. Note that at most n^2 invocations of ICP occur, each with error probability of $\epsilon'' = \frac{\epsilon}{2n^2}$. Each of these invocations incurs less than $7 \log \frac{1}{\epsilon}$ invocations of A-RS. Thus, altogether there are less than $7n^2 \log \frac{1}{\epsilon}$ invocations of A-RS, each with error probability of $\frac{\epsilon}{14n^2 \log \frac{1}{\epsilon}}$. Consequently the probability that an error occurs in any of the invocations of ICP or A-RS is at most ϵ. For convenience and clarity, we partition the proof of Theorem 4 to several short lemmas. (The Completion and Secrecy properties are taken from Definition 2.)

Lemma 2 *Given that no uncorrupted player has completed AWSS-Rec then for every uncorrupted P_i, P_j eventually $C_i = C_j$.*

Proof: Let us look at some $P_l \in C_i$. If player P_i has placed P_l in C_i (Step 4) then P_i has completed P_l's A-RS-Share of Step 3. Due to the Completion property of the A-RS protocol we know that eventually P_j will also conclude that A-RS-Share has ended and then P_j will add P_l to C_j. □

Lemma 3 *Given that no uncorrupted player has completed AWSS-Rec then for every uncorrupted P_i, P_j eventually $\mathcal{E}_i = \mathcal{E}_j$.*

Proof: Let us look at some $(P_l, A_l) \in \mathcal{E}_i$. If player P_i placed (P_l, A_l) in $\mathcal{E}_i$ then P_i has completed the a-cast (`Acceptance set`,P_l,A_l) and $P_l \in C_i$ and $A_l \subseteq C_i$. Due to the a-cast property P_j will also complete the above a-cast, and eventually we will have (due to Lemma 2) $P_l \in C_i = C_j$, and $A_l \subseteq C_i = C_j$. Then, P_j will add (P_l, A_l) to $\mathcal{E}_j$. □

Lemma 4 (Completion (1):) *If the dealer is uncorrupted then each uncorrupted player will eventually output* `‘sharing succeeded’`.

Protocol AWSS-Share[ϵ]

Set $\epsilon' = \frac{\epsilon}{14n^2 \log \frac{1}{\epsilon}}$ and $\epsilon'' = \frac{\epsilon}{2n^2}$.

Code for the dealer, on parameter ϵ and secret s:

1. Choose a random polynomial $f(x)$ for s.
2. Set $\phi_i \triangleq f(i)$. For each two players P_i and P_j Execute $\mathbf{Gen}_{D,i,j}[\epsilon''](\phi_i)$ with player P_i as T and player P_j as R .

Code for player P_i, on parameter ϵ:

3. Wait until the values from Step 2 have been received. Then, invoke A-RS-Share[ϵ'] (as a dealer) for sharing the following values:
   (a) The share ϕ_i and all the values received as T from the dealer in the invocations of $\mathbf{Gen}_{D,i,j}[\epsilon''](\phi_i)$.
   (b) For each j, the values received as R from the dealer, in the invocations of $\mathbf{Gen}_{D,j,i}[\epsilon''](\phi_j)$. We shall refer to the values shared for j as R_i-$info_j$.
4. Initialize a Communication Set $C_i := \emptyset$.
   For each j, wait until all the invocations of A-RS-Share[ϵ'] initiated by P_j in Step 3 are (locally) completed. Then, add player P_j to C_i.
5. Initialize an Acceptance Set, $A_i := \emptyset$.
   (a) For each j, wait until $P_j \in C_i$. Then, invoke as T the protocol $\mathbf{Ver}_{i,j}[\epsilon''](\phi_i)$ with player P_j as R .
   (b) For each j, wait until $\mathbf{Ver}_{i,j}[\epsilon''](\phi_i)$ is completed. Then, if $\mathbf{Ver}_{i,j}[\epsilon''](\phi_i) = \phi_i$ then add player P_j to A_i.
   (c) Wait until $|A_i| = 2t+1$. Then A-cast `(Acceptance Set,`P_i`,`A_i`)`.
6. Initialize a *shared variable* called the Eligibility Set, $\mathcal{E}_i := \emptyset$.
   For each j, wait until P_j's a-cast of the form `(Acceptance Set,`P_j`,`A_j`)` has been completed, and $P_j \in C_i$ and $A_j \subseteq C_i$. Then, add the pair (P_j, A_j) to $\mathcal{E}_i$.
7. Wait until $|\mathcal{E}_i| = n-t$. Define the Interpolation Set $IS_i \triangleq \mathcal{E}_i^{(n-t)}$. A-cast `(Interpolation set,`$P_i$`,`$IS_i$`)`.
8. Initialize a shared variable called a Final Set, $\mathcal{F}_i := \emptyset$
   For each j, wait until P_j's a-cast of the form `(Interpolation Set,`P_j`,`IS_j`)` has been completed, and for each $(P_l, A_l) \in IS_j$ it holds that $(P_l, IS_l) \in \mathcal{E}_i$. Then, add the pair (P_j, IS_j) to $\mathcal{F}_i$.
9. Wait until $|\mathcal{F}_i| = n-t$. Then, stop adding players to $\mathcal{F}_i$ and output `` `sharing succeeded' ``.

Figure 5: The AWSS Sharing Protocol

Protocol AWSS-Rec$[\epsilon]$

Code for player P_i, on parameter ϵ:

1. For each pair $(P_j, A_j) \in \mathcal{E}_i$:
   (a) Execute A-RS-Rec$[\epsilon']$ of the value shared in AWSS-Sh Step 3a (this is P_j's share ϕ_j and values needed for P_j as T with respect to ϕ_j to carry out $\mathbf{Auth}_{j,l}[\epsilon'']$ with each player P_l as R).
   (b) Wait until all the invocations of A-RS-Rec$[\epsilon']$ of the values shared in Step 3a of P_j have been completed. Then, for each $P_l \in A_j$ execute A-RS-Rec$[\epsilon']$ of P_l's value shared in Step 3b of AWSS-Share associated with P_j (i.e. the values in R_l-*info*$_j$). These are the values that P_l needs needed to carry out $\mathbf{Auth}_{j,l}[\epsilon'']$ as R .
   (c) Wait until all the invocations of A-RS-Rec$[\epsilon']$ of values shared in step 3 of AWSS-Share, with P_j as the dealer, are reconstructed. Then, using the reconstructed values, and for each $P_l \in A_j$, simulate $\mathbf{Auth}_{j,l}[\epsilon'']$ acting both as P_j and P_l. If at least $t+1$ $\mathbf{Auth}_{j,*}[\epsilon''] = (\phi_j)$ then associate the share ϕ_j with player P_j, else associate *null* with P_j.
2. For each j such that $(P_j, IS_j) \in \mathcal{F}_i$, wait until some value is associated with all the players P_l such that $P_l \in, IS_j$. Then, define
   $S_j = \{\phi \mid \phi \neq null,\ \phi \text{ is associated with player } P_l, \text{and } (P_l, A_l) \in IS_j\}$
   and set
   $$b_j = \begin{cases} v_j & \text{if the shares of } S_j \text{ define the secret } v_j \\ null & \text{otherwise} \end{cases}$$
3. Wait until b_j is set for all players P_j such that $(P_j, IS_j) \in \mathcal{F}_i$. Then, if all the values $\{b_j | (P_j, IS_j) \in \mathcal{F}_i\}$ are identical then set c_i to this value. Otherwise set $c_i = null$. A-cast $(P_i$,`Reconstructed-AWSS`$,c_i)$.
4. Wait until receiving $2t+1$ a-casts of the form $(P_j$,`Reconstructed-AWSS`$,c_j)$. Set *final*$_i$ to the value c_j if this value appears in $t+1$ of the a-casts, otherwise *final*$_i$=*null*. Output *final*$_i$. If Step 9 of AWSS-Share was not executed then output `` `sharing succeeded' ``. Complete both AWSS-Sh and AWSS-Rec.

Figure 6: The AWSS - Reconstruction Protocol

Proof: All uncorrupted players will eventually be in the sets C_i and A_i of every uncorrupted player P_i. Thus A_i will eventually be of size $2t+1$, and every uncorrupted player P_i will a-cast (`Acceptance Set`,P_i, A_i) in Step 5 of AWSS-Share. As it holds for an uncorrupted player P_j that $A_j \subseteq C_j$ then due to the A-RS completion property player P_i will also have $A_j \subseteq C_i$, and hence all uncorrupted players will eventually be in the set $\mathcal{E}_i$ of every uncorrupted player. Consequently, every uncorrupted player P_j will a-cast (`Interpolation Set`,P_j,IS_j) in Step 7. As it holds for an uncorrupted P_j that $IS_j \subseteq \mathcal{E}_j$ it will hold that for each uncorrupted P_i that $IS_j \subseteq \mathcal{E}_j = \mathcal{E}_i$. Thus every uncorrupted player P_i will add this pair (P_j, IS_j) to $\mathcal{F}_i$. Thus an uncorrupted player P_i will eventually have $|\mathcal{F}_i| \geq n-t$ and will set output \``sharing succeeded`'. □

Lemma 5 (Completion (2)): *If some uncorrupted player, P_i, has output* \``sharing succeeded`', *then each uncorrupted player $P_j$ will eventually output* \``sharing succeeded`'.

Proof: If some uncorrupted player has completed AWSS-Rec then this means that he has completed $2t+1$ a-casts of the form (P_i,`Reconstructed-AWSS`,b_i). Due to the properties of a-cast each other uncorrupted player will complete these a-cast and output \``sharing succeeded`'. Thus, we can assume that no uncorrupted player has completed protocol AWSS-Rec, and that player $P_i$ has outputted \``sharing succeeded`'. Thus, $|\mathcal{F}_i| = n-t$, which means that $P_i$ has completed $n-t$ a-casts of the form (`Interpolation set`,P_l, IS_l), and $IS_l \subseteq \mathcal{E}_i$. Due to the properties of a-cast P_j will eventually complete these invocations of a-cast. Due to Lemma 3, we have that eventually $\mathcal{E}_i = \mathcal{E}_j$, hence, the requirement that $IS_l \subseteq \mathcal{E}_j$ will be satisfied and P_j will output \``sharing succeeded`'. □

Lemma 6 (Completion (3)): *If the uncorrupted players output* \``sharing succeeded`' *and all uncorrupted players have* \``start running Rec`' *on their input tapes, then all the uncorrupted players will complete protocol AWSS-Rec.*

Proof: As no uncorrupted player has completed AWSS-Rec, we have that $\mathcal{E}_i = \mathcal{E}_j$ for uncorrupted players P_i, P_j. Thus, all uncorrupted players will participate in A-RS-Rec of the values in Steps 1a and 1b. Due to the properties of A-RS-Rec all these invocations will be completed, and an uncorrupted P_i will have the necessary values to complete Step 1c. Thus, in Step 2 each uncorrupted player P_i will a-cast the appropriate message, and thus each uncorrupted player will complete $2t+1$ such a-cast and complete protocol AWSS-Rec. □

Correctness (1) and (2) are proven in Lemma 7 through Lemma 10.

Lemma 7 *Let ϕ_j be the share that player P_j received from the dealer in Step 2 of AWSS-Share. With overwhelming probability, the value which an uncorrupted player P_i associates with a player $(P_j, A_j) \in \mathcal{E}_i$ in Step 1(c) of AWSS-Rec is as follows:*

	uncorrupted dealer	corrupted dealer
P_j uncorrupted	ϕ_j	ϕ_j
P_j corrupted	ϕ_j *or null*	—

Proof: Player P_i will associate a value ϕ'_j with player P_j if the following two requirements hold:

1. ϕ'_j is the value which was reconstructed by the A-RS-Rec of P_j's share.

2. There are at least $t+1$ players in A_j for whom $\mathbf{Auth}_{j,*}[\epsilon''] = \phi'_j$.

Consider first the case where P_j is uncorrupted. In this case P_j shared the value ϕ_j in Step 3a of AWSS-Share. From the correctness of A-RS we have that P_i will reconstruct, in Step 1a of AWSS-Rec, the correct values of ϕ_j and of P_j's data needed for executing $\mathbf{Auth}_{j,l}[\epsilon'']$ for each player $P_l \in A_j$. Furthermore, at least $t+1$ out of the players in A_j are uncorrupted. The data of these $t+1$ players will also be correctly reconstructed, in Step 1b of AWSS-Rec. Thus it follows from the properties of ICP that P_i will successfully verify, in Step 1c, that $\mathbf{Auth}_{j,l}[\epsilon''] = \phi_j$ for at least $t+1$ players $P_l \in A_j$, and will associate ϕ_j with P_j.

Consider the case where P_j is corrupted and the dealer is uncorrupted, and assume that the value ϕ'_j reconstructed in the A-RS-Rec of P_j's share is different than ϕ_j. Player P_i will, in Step 1c of AWSS-Rec, associate a non-null value with P_j only if P_i has $\mathbf{Auth}_{j,l}[\epsilon''] = \phi'_j$ for at least $t+1$ players P_l; one of these $t+1$ players must be uncorrupted. We show that P_i computes $\mathbf{Auth}_{j,l}[\epsilon''] = \phi'_j$ for some uncorrupted player P_l only with negligible probability.

Let I_j denote the values that P_i reconstructs, in Step 1a of AWSS-Rec which includes P_j's share value ϕ'_j and all the values required for $\mathbf{Auth}_{j,l}, \quad 1 \leq l \leq n$ where P_j is T . Let $\mathrm{R}_{j,l}$ denote the values that P_i reconstructs in Step 1b of AWSS-Rec which includes P_l's data relevant for the invocation of $\mathbf{Auth}_{j,l}[\epsilon'']$ where P_l is R.

Given that P_l is uncorrupted, A-RS ensures that $\mathrm{R}_{j,l}$ is indeed P_l's data for $\mathbf{Auth}_{j,l}[\epsilon'']$. Therefore, if it were true that I_j is determined by the adversary independently of $\mathrm{R}_{j,l}$ for all l, then Property 3 of ICP (see Section 4) would imply that P_i sets $\mathbf{Auth}_{j,l}[\epsilon''] = \phi'_j$ for some uncorrupted P_l with negligible probability. However, since P_j is corrupted, A-RS ensures only a weaker requirement on I_j. We show that this property suffices.

A-RS allows I_j to be set by the adversary during the reconstruction stage. However, once the first uncorrupted player completes A-RS-Rec, a small set S of $2t+1$ possible values for I_j is fixed. By the time that S is fixed no uncorrupted player has yet started the reconstruction of $\mathrm{R}_{j,l}$ in Step 1b, (i.e., of the data shared by an uncorrupted P_l in Step 3b of AWSS-Share). Thus, $\mathrm{R}_{j,l}$ for all uncorrupted P_l is unknown to the adversary when S is fixed. The adversary can change its values to any value in S, thus it has $n-t$ options. It will succeed in foiling P_i's computation if for any uncorrupted $P_l \in A_j$ it would hold that $\mathbf{Auth}_{j,l}[\epsilon''] = \phi'_j$. Assuming that there are t corrupted players in A_j (which is the best case for the adversary as in this case it needs $\mathbf{Auth}$ to succeed for a single additional player) then there are $t+1$ uncorrupted players in A_j. This results in the adversary's probability of having a single invocation of $\mathbf{Auth}_{j,*}$ succeed is $(t+1)(n-t)\epsilon$, where ϵ is the probability of success for the adversary in a single invocation of $\mathbf{Auth}_{j,*}$. Thus, P_i will output $\phi'_j \neq \phi_j$ with negligible probability. □

In order to proceed with our proof we define the set V and derive from it the value r used for the Correctness requirement. Assume that player P is the first uncorrupted player to complete AWSS-Share. Let

$$V = \{(P_i, v_i) \mid v_i \text{ is the secret defined by the shares of the uncorrupted players in } IS_i, \ \ (P_i, IS_i) \in \mathcal{F}_P\}.$$

Let

$$r = \begin{cases} v_i & \text{if } v_i \text{ appears in at least } n-2t \text{ pairs of } V \\ null & \text{otherwise} \end{cases}$$

We stress that P does not know V. Still, V is a useful tool in our proof.

For each uncorrupted player P_i who has completed AWSS-Share we define the following:

$V_i = \{(P_j, v_j) \mid v_j$ is the secret defined by the shares of the uncorrupted players in IS_j, $(P_j, IS_j) \in \mathcal{F}_i\}$.

Let

$$r_i = \begin{cases} v_j & \text{if } v_j \text{ appears in all } n-t \text{ pairs of } V_i \\ null & \text{otherwise} \end{cases}$$

Lemma 8 *If the values r is null than r_i is null for all uncorrupted player P_i.*

Proof: The intersection of $|\mathcal{F}_P \cap \mathcal{F}_i| \geq n-2t$, thus $|V \cap V_i| \geq n-2t$. As r is *null* this means that there is no value which appears at least $n-2t$ times in V. Thus, there are at least two different values in V_i, which in return means that r_i is *null*.

Lemma 9 *If the value r and the value r_i of an uncorrupted player P_i do not equal null, then $r = r_i$.*

Proof: As $r \neq null$ there are $n-2t$ pairs $(P_l, r) \in V$, denote this set by D. $|V \cap V_i| \geq n-2t$, this guarantees that $|D \cap V_i| \geq 1$. Hence, there exists a pair $(P_l, r) \in V_i$. The value r_i is not *null* thus all pairs $(P_l, r_l) \in V_i$ are of the form (P_l, r_i). Thus $r_i = r$. □

Lemma 10 (Correctness (1)): *If the dealer is uncorrupted, sharing a secret s, then every uncorrupted player P_i outputs s.*

Proof: We need to prove that an uncorrupted player P_i the value $b_i = s$, i.e. that the shares of S_j for each $(P_j, IS_j) \in \mathcal{F}_i$ define the secret s. From Lemma 7 we know that given that the dealer is uncorrupted an uncorrupted P_i will associate with a corrupted player P_l either the proper share given to P_l by the dealer or *null*. All the shares given out by the dealer define the secret s, and thus the shares of S_j define s. It now follows from Lemma 11 that P_i outputs s. □

Lemma 11 Correctness (2). *Every uncorrupted player P_i outputs, upon completing AWSS-Rec, either $r_i = r$ or null.*

Proof: If player P_i outputs *null* then we are done, thus assume that it outputs a non-null value r'_i, and we shall prove that $r'_i = r_i$. Player P_i outputs r'_i if it is the value defined by all S_j, where $(P_j, IS_j) \in \mathcal{F}_i$, where S_j is the set of non-null shares associated with the players in IS_j. The set of uncorrupted players in IS_j is a subset of size at least $t+1$, thus this subset defines the secret r'_i as well. We also have by definition that r_i is the value defined by the subset of uncorrupted players in IS_j. Thus, $r'_i = r_i$.

□

Lemma 12 (Secrecy): *If the dealer is uncorrupted then the distribution of the adversary view of AWSS-Share does not depend on the shared secret.*

Proof: Informally, the relevant information in the adversary view of the computation is shares of the various polynomials from Step 1 of AWSS-Share and the different invocations of A-RS-Share, together with the field elements from the invocations of ICP. Since the shared of at most t players are included in the adversary view, all these values are distributed uniformly and independently in $\mathbf{F}$, regardless of the value of the shared secret.

More formally, the proof of security proceeds by induction similarly to the proof of secrecy of A-RS (Theorem 3). We omit further details. □

6.1 Two&Sum-AWSS

In our AVSS protocol we do not use AWSS as described in the previous section. Instead, we use a slight variation of AWSS, called Two&Sum-AWSS. A dealer shares a secret s, together with a set $\{a_1, \ldots, a_k\}$ of *auxiliary secrets,* in a way that allows the players to separately reconstruct the secret, any one of the auxiliary secrets, and the sum $s + a_i$ of the secret with any one of the auxiliary secrets. (That is, *several* invocations of the reconstruction protocol will use the shares obtained in a *single* invocation of the sharing protocol.) For each i, reconstructing any single value out of $\{s, a_i, s + a_i\}$ reveals no information on the other two. The correctness and secrecy properties of Two&Sum-AWSS is stated more formally as follows.

Correctness of Two&Sum-AWSS: *Once some uncorrupted player completes protocol* Sh, *there exist* fixed *values,* $r_s, \{r_{a_1}, \ldots, r_{a_k}\}, \{r_{s+a_1}, \ldots, r_{s+a_k}\} \in \mathbf{F} \cup \{null\}$, *such that the following requirements hold:*

1. *If the dealer is uncorrupted, sharing* $s, \{a_1, \ldots, a_k\}$, *then,* $r_s = s$, $r_{a_j} = a_j$ *and* $r_{s+a_j} = s + a_j$, *and when reconstructing the secret (resp., the* i*th auxiliary secret, or the sum of the secret and the* i*th auxiliary secret), each uncorrupted player outputs* s *(resp.,* a_i, *or* $s + a_i$*).*
2. *If the dealer is corrupted then at the end of AWSS-Rec*
   (a) *each uncorrupted player outputs either* r_s *(resp.,* r_{a_i}, *or* r_{s+a_i}*) or null.*
   (b) *If* r_s *is null then, for each* i, *at least at the end of one of the following reconstructions the uncorrupted players will output null, AWSS-Rec of* r_{a_j} *or AWSS-Rec of* r_{s+a_j}.
   (c) *If* r_s, r_{a_i} *and* r_{s+a_i} *are not null then* $r_{s+a_i} = r_s + r_{a_i}$.

Secrecy: *As long as the dealer remains uncorrupted, and even if for each* i *one value out of* $\{a_i, s + a_i\}$ *are reconstructed, the distribution of the adversary view does not depend on the value of the shared secret.*

Our construction of Two&Sum-AWSS, presented in Figure 7, is a straightforward generalization of our AWSS protocol. (The protocol is based on the *synchronous* Two&Sum-WSS presented in [Rab94].)

The reconstruction protocol is identical to AWSS-Rec, with the addition of parameters specifying which value should be reconstructed. We denote by AWSS-Rec$_{\mathrm{S}}$ (resp., AWSS-Rec$_{\mathrm{A}_i}$, AWSS-Rec$_{\mathrm{S}+\mathrm{A}_i}$) the invocation for reconstructing the secret (resp., the ith auxiliary secret, the sum of the secret and the ith auxiliary secret).

Theorem 5 *Let* $n \geq 3t + 1$. *Then for every* $\epsilon > 0$, *the pair* (Two&Sum-AWSS-Share[ϵ],AWSS-Rec[ϵ]) *is a* $(1 - \epsilon)$*-correct, t-resilient Two&Sum AWSS protocol for n players. &Sum*

Proof: The proof of the Completion properties is identical to the proof of the Completion properties of AWSS (Lemmas 2 through 6). **Correctness (1) and Correctness (2a)** follow immediately from Lemma 10, this is due to the fact that each individual sharing is by itself an AWSS-Sh.

The rest of the proof is presented in Lemmas 13 and 14 below. We set the values r_s and r_{a_j}, r_{s+a_j} in the same manner as defined in the analysis of AWSS, with the exception that P_i is the first uncorrupted player to complete Two&Sum-AWSS-Sh. □

Lemma 13 Correctness (2b) *If* r_s *is null then, for each* j, *at least one of* r_{a_j} *and* r_{s+a_j} *is null.*

Protocol Two&Sum AWSS-Share$[\epsilon, k]$

Set $\epsilon' = \frac{\epsilon}{14kn^2 \log \frac{1}{\epsilon}}$ and $\epsilon'' = \frac{\epsilon}{2kn^2}$.

Code for the dealer: (on input $s, a_1, \ldots, a_k$ and ϵ)

1. Choose random polynomials $f(\cdot)$ for s and $g_l(\cdot)$ for each a_l. Let $\phi_i \triangleq f(i)$ and $\gamma_{l,i} \triangleq g_l(i)$ for $0 \leq l \leq k$ and $1 \leq i \leq n$. Let $\sigma_{l,i} \triangleq \phi_i + \gamma_{l,i}$.
   Send $\phi_i, \{\gamma_{1,i} \ldots, \gamma_{k,i}\}$ to each player P_i.
2. For every two players P_i, P_j, invoke: $\mathbf{Gen}_{D,i,j}[\epsilon''](\phi_i)$, $\mathbf{Gen}_{D,i,j}[\epsilon''](\gamma_{1,i}), \ldots \mathbf{Gen}_{D,i,j}[\epsilon''](\gamma_{k,i})$, and $\mathbf{Gen}_{D,i,j}[\epsilon''](\sigma_{1,i}), \ldots \mathbf{Gen}_{D,i,j}[\epsilon''](\sigma_{k,i})$, with P_i as the Intermediary and P_j as the Receiver.

Code for player P_i

3. Wait until the values from Step 2 have been received. Then, invoke A-RS-Share$[\epsilon']$ (as a dealer) for sharing the following values:
   (a) The share ϕ_i and all the values received as T from D in the invocations of $\mathbf{Gen}_{D,i,j}([\epsilon'']\phi_i)$ in Step 2.
   (b) For each l, $1 \leq l \leq k$ share the value $\gamma_{l,i}$ and all the values received as T from D in the invocations of $\mathbf{Gen}_{D,i,j}[\epsilon''](\gamma_{l,i})$ in Step 2.
   (c) For each l, $1 \leq l \leq k$ share the value $\sigma_{l,i}$ and all the values received as T from D in the invocations of $\mathbf{Gen}_{D,i,j}[\epsilon''](\sigma_{l,i})$ in Step 2.
   (d) For each j share the values received as R from D, in the invocations of $\mathbf{Gen}_{D,j,i}[\epsilon'']$ for ϕ_i and $\gamma_{l,i}, \sigma_{l,i}$ for $1 \leq l \leq k$ of Step 2.
4. Initialize a Communication Set $C_i := \emptyset$.
   For each j, wait until all the A-RS-Shares initiated by P_j in Step 3 are (locally) completed. Then, add player P_j to C_i.
5. Create an Acceptance Set, A_i.
   Until $|A_i| = 2t + 1$ do:
   Invoke as T the protocols $\mathbf{Ver}_{i,j}[\epsilon''](\phi_i)$, $\mathbf{Ver}_{i,j}[\epsilon''](\gamma_{1,i}), \ldots \mathbf{Ver}_{i,j}[\epsilon''](\gamma_{k,i})$, and $\mathbf{Ver}_{i,j}[\epsilon''](\sigma_{1,i}), \ldots \mathbf{Ver}_{i,j}[\epsilon''](\sigma_{k,i})$, for each player $P_j \in C_i$ as R . If all the invocations are successful then add player P_j to A_i.
   A-cast `(Acceptance Set,`P_i`,`A_i`)`.
   Initialize an Acceptance Set, $A_i := \emptyset$.
   (a) For each j, wait until $P_j \in C_i$. Then, invoke as T the protocols $\mathbf{Ver}_{i,j}[\epsilon''](\phi_i)$, $\mathbf{Ver}_{i,j}[\epsilon''](\gamma_{1,i}), \ldots \mathbf{Ver}_{i,j}[\epsilon''](\gamma_{k,i})$, and $\mathbf{Ver}_{i,j}[\epsilon''](\sigma_{1,i}), \ldots \mathbf{Ver}_{i,j}[\epsilon''](\sigma_{k,i})$, with player P_j playing the role of R .
   (b) For each j, wait until all the invocations of $\mathbf{Ver}_{i,j}[\epsilon''](\phi_i)$, $\mathbf{Ver}_{i,j}[\epsilon''](\gamma_{1,i}), \ldots \mathbf{Ver}_{i,j}[\epsilon''](\gamma_{k,i})$, and $\mathbf{Ver}_{i,j}[\epsilon''](\sigma_{1,i}), \ldots \mathbf{Ver}_{i,j}[\epsilon''](\sigma_{k,i})$, are completed. Then, if all these invocations have non-null output then add player P_j to A_i.
   (c) Wait until $|A_i| = 2t + 1$. Then A-cast `(Acceptance Set,`P_i`,`A_i`)`.
6. Execute the rest of the code of AWSS-Share (i.e., Steps 6 through 9 of Figure 5).

Figure 7: The Two&Sum AWSS Sharing Protocol

Proof: If r_{a_j} is *null* then we are done. Thus, assume that $r_{a_j} \neq$ *null*, and we shall prove that r_{s+a_j} is *null*. Since $r_{a_j} \neq$ *null* there are at least $n-2t$ sets IS_l in which the shares of the uncorrupted players define r_{a_j}. As the intersection of good players in two IS is greater than $t+1$, we have that the rest of the Interpolation Sets do not define a secret, i.e. they interpolate to a polynomial of degree greater than t. On the other hand r_s is *null* this means that at most $n-2t-1$ sets IS_l in which the shares of the uncorrupted players define the same secret. Due to the same argument concerning the Interpolation Set the rest define a polynomial of degree greater than t. This in return gives that there are at most $n-2t-1$ sets which define r_{s+a_j}, and all the rest define polynomials of degree greater than t. Hence, r_{s+a_j} is *null*. □

Lemma 14 *As long as the dealer remains uncorrupted, and even if for each i one value out of $\{a_i, s+a_i\}$ are reconstructed, the distribution of the adversary view does not depend on the value of the shared secret.*

Proof: Informally, on top of the information included in the adversary view of AWSS-Share, the adversary view now includes k reconstructed values where each of these values is uniformly and independently distributed in $\mathbf{F}$. Thus, the relevant information in the adversary view is still uniformly distributed, regardless of the value of the shared secret.

Also here, a formal proof is similar to the proof of the secrecy property of A-RS (Theorem 3). Details are omitted. □

7 Asynchronous Verifiable Secret Sharing — AVSS

The idea of the AVSS protocol is sketched as follows. First, the dealer sends each player a share of the secret, as in the previous protocol. The players then 'commit' to their shares by re-sharing them using AWSS. Next, the players make sure, using a cut-and-choose method (as in [CCD88, Fel89]), that enough AWSS-Sharings have been successful and that the 'committed upon shares' define a secret. We describe the protocol in some more detail. Full details appear in Figure 8.

Sharing protocol. The dealer, sharing a secret s, chooses a random polynomial $f(x)$ for s. For each i, the dealer sends $f(i)$ to P_i. In addition, the dealer chooses $k \cdot n \cdot t$ random polynomials $g_{1,1,1}(x), \ldots, g_{k,n,t}(x)$ of degree t, where k is an appropriate security parameter (we later set $k = O(n + \log \frac{1}{\epsilon})$). For each such polynomial $g(x)$, and for each i, the dealer sends $g(i)$ to player P_i. Upon receiving all the expected values from the dealer, each P_i re-shares the values $f(i), \{g_{1,1,1}(i), \ldots, g_{k,n,t}(i)\}$ using Two&Sum-AWSS-Share with the appropriate allowed-error parameter (see Section 6.1).

Next, the dealer proves to the players that their shares indeed define a secret. Each player P_i is waiting to have a set IS_i of at least $n-t$ players, such that the Two&Sum-AWSS-Share's of these players have been completed, and the corresponding values define a polynomial. For this purpose P_i participates in up to t 'iterations', as follows.[8] Initially, all players are valid. At the beginning of each iteration r, P_i waits until it has completed the Two&Sum-AWSS-Share of $n-t$ valid players. Let $IS_{i,r}$ denote this set of players. Next P_i verifies, using cut-and-choose as described in the code, that the share of each player in $IS_{i,r}$ is valid (in a sense defined within). P_i removes all the players

[8]Partitioning the protocol into t iterations is done for clarity of exposition. All these iterations are completed in a constant number of asynchronous time steps.

in $IS_{i,r}$ whose validation failed from the set of valid players. If the validity of the shares of at least $n-t$ players in $IS_{i,r}$ is confirmed, then the iteration was successful. In this case, P_i a-casts (P_i `confirms` P_j) for each player $P_j \in IS_{i,r}$. (It will be seen that in this case the shares of the players in $IS_{i,r}$ define a polynomial.) When P_i has completed the Two&Sum-AWSS-Share of at least one other player, P_i initiates the next iteration.

Let F_i be P_i's set of players who were confirmed by at least $t+1$ players. Player P_i outputs `` `sharing succeeded' ``once $|F_i| \geq 2t+1$.

Reconstruction protocol. The reconstruction protocol is simple: The share of each player P_i is reconstructed using AWSS-Rec$_{i,\mathrm{S}}$.[9] Once $t+1$ shares of players in F_i are reconstructed with a non-null value, player P_i computes the secret defined by these shares, and terminates.

Protocols AVSS-Share and AVSS-Rec are presented in Figures 8 and 9, respectively.

Theorem 6 *Let $n \geq 3t+1$. Then for every $\epsilon > 0$, the pair* (AVSS-Share[ϵ],AVSS-Rec[ϵ]) *is a $(1-\epsilon)$-correct, t-resilient AVSS protocol for n players.*

For convenience and clarity we partition the proof of Theorem 6 to several lemmas. Throughout the proof we assume that the following event, denoted E, occurs: All invocations of Two&Sum-AWSS have been properly completed. That is, if an uncorrupted player has completed Two&Sum-AWSS-Share then all uncorrupted players will complete all corresponding invocations of AWSS-Rec, outputting either the required value or *null*. (See the Correctness requirement of Two&Sum-AWSS, on page 27.) Event E occurs with probability at least $1 - n\epsilon' = 1 - \epsilon/2$.

Lemma 15 (Completion (1)): *If the dealer is uncorrupted then each uncorrupted player will complete protocol AVSS-Share.*

Proof: Each uncorrupted player P_i will complete the Two&Sum-AWSS-Share of the shares of each uncorrupted player P_j in Step 4. Thus P_j will eventually be in the set $IS_{i,r}$ of P_i for some iteration r. If the dealer is uncorrupted then P_j's reconstructed shares will agree, in Step 7c, with the polynomials a-casted by the dealer. Thus, P_j will not be removed from either $IS_{i,r}$ or V_i. Hence there will exist an iteration r where all $n-t$ uncorrupted players are in $IS_{i,r}$, causing each uncorrupted player P_i to a-cast (P_i,`confirms`,P_j). Consequently, each uncorrupted player will be in the final set F_k of each uncorrupted P_k. Thus P_k will have $|F_k| \geq 2t+1$ and will output `` `Sharing succeeded' ``. Furthermore all uncorrupted players complete the protocol within constant time. (This is argued as follows. Let P_i be the last player that completes AVSS-Share, and let P_m be the last player that enters the set F_j. Let P_i be the last player that a-casts (P_i,`confirms player`,P_m), at step e. Then, P_j outputs `` `Sharing succeeded' `` only a constant time after step e. Furthermore, step e occurs only a constant time after P_m is added to C_i. Finally, P_m is added to C_i only a constant time after P_i started running AVSS-Share.) □

Lemma 16 (Completion (2)): *If some uncorrupted player P_i completes protocol AVSS-Share, then each uncorrupted player will eventually complete protocol AVSS-Share.*

Proof: Assume an uncorrupted player P_i completed protocol AVSS-Share. Then $|F_i| \geq 2t+1$. It follows from the correctness properties of a-cast that, for any uncorrupted player P_j, any player in F_i will eventually be in F_j. Thus P_j outputs `` `Sharing succeeded' `` (within constant time). □

[9] We let AWSS-Rec$_{i,*}$ denote an invocation of AWSS-Rec$_*$ that corresponds to the Two&Sum-AWSS-Share of P_i.

Protocol AVSS-Share$[\epsilon]$

Set $\epsilon' = \epsilon/2n$.

Code for the dealer, on parameter ϵ and secret s:

1. Choose a random polynomial $f(x)$ for s. Send $\phi_i \triangleq f(i)$ to each player P_i.
2. Set $k \triangleq O(n, \log \frac{1}{\epsilon})$. For $1 \le l \le k$, for $1 \le j \le n$, and for $1 \le r \le t$ do:
   Pick a random polynomial $g_{l,j,r}(x)$ of degree t, and send $g_{l,j,r}(i)$ to each player P_i.
3. If completed an a-cast
   (`Random vector,`$b_{1,j,r}, ..., b_{k,j,r}$`,iteration` r`,player,`P_j) (see Step 7a), a-cast
   (`Polynomials of` P_j`, iteration` r`,` $g_{1,i,r}(\cdot) + f(\cdot) \cdot b_{1,i,r}, ..., g_{k,i,r}(\cdot) + f(\cdot) \cdot b_{k,i,r}$`)`.

Code for player P_i, on parameter ϵ:

4. Wait until the shares sent in Step 2 are received. Then, share $\phi_i, \{g_{1,1,1}(i), \ldots, , g_{k,n,t}(i)\}$ using Two&Sum-AWSS-Share$[\epsilon', n \cdot k \cdot t]$ (see Figure 7). Participate in the Two&Sum-AWSS-Share of other players.
5. Initialize a set $C_i := \emptyset$. For each j, wait until the invocation of Two&Sum-AWSS-Share where P_j is the dealer outputs `` `sharing succeeded' ``. Then, add P_j to C_i.
6. Let V_i be the set of valid players. Initially $V_i = \{P_1, \ldots, P_n\}$.
   For each $1 \le r \le t$, initialize a set $IS_{i,r} = \emptyset$.
   Initialize a final set $F_i = \emptyset$.
   Set a local variable ρ to 1.
7. If $|F_i| < 2t + 1$, do:
   (a) For each $1 \le r \le t$, wait until $|C_i \cap V_i| \ge n - t$ and $C_i \cap V_i \not\subset IS_{i,r-1}$, and $\rho = r$. Then:
   — Let $IS_{i,r} := C_i \cap V_i$
   — Set $\rho := \rho + 1$.
   — Choose $b_{1,i,r}, \ldots, b_{k,i,r} \in_{\mathrm{R}} \{0, 1\}$, and a-cast
   (`Random vector,`$b_{1,i,r}, ..., b_{k,i,r}$`, iteration` r`,player,`P_i).
   (b) For each r, wait until completing an a-cast (`Polynomials of` P_j`, iteration` r`, ...`) (see Step 3). Then, for $l = 1...k$, $m = 1...n$ do: If $b_{l,j,r} = 0$ execute AWSS-Rec$_{m,\mathrm{A}_{l,j,r}}$; If $b_{l,j,r} = 1$ then execute AWSS-Rec$_{m,\mathrm{S}+\mathrm{A}_{l,j,r}}$.
   (c) For each player P_m, for each l and for each r, wait until the corresponding AWSS-Rec of a share of player P_m is completed. Then, if the output value is *null*, or if any reconstructed share of P_m does not agree with the corresponding polynomial a-casted by the dealer, then P_m is removed from V_i and from $IS_{i,r}$.
   (d) For each r, wait until all the relevant invocations of AWSS-Rec of shares of players in $IS_{i,r}$ are completed. Then, if $|IS_{i,r}| \ge n - t$ then for each $P_m \in IS_{i,r}$ a-cast (P_i`,confirms player,`P_m`)`. (In this case we say that iteration r is successful.)
8. For each P_m, wait until $t + 1$ a-casts of the form (`...,confirms player,`P_m) (see Step 7c) are completed. Then, add P_m to F_i.
9. Wait until $|F_i| = 2t + 1$. Then output `` `sharing succeeded' ``.

Figure 8: The AVSS Sharing Protocol

Protocol AVSS-Rec[ϵ]

Code for player P_i, on parameter ϵ:

1. For each player P_j execute AWSS-Rec$_{j,\mathrm{S}}$. (P_j's share, ϕ_j, was shared by P_j in the Two&Sum-AWSS-Share of Step 4 of AVSS-Share.)
2. Wait until $t+1$ invocations of AWSS-Rec$_{j,\mathrm{S}}$, where $P_j \in F_i$, are completed with non-null outputs. Then, output the secret defined by the $t+1$ reconstructed values. Complete protocols AVSS-Share and AVSS-Rec.

Figure 9: The AVSS Reconstruction Protocol

Lemma 17 (Completion (3)): *If AVSS-Share has been completed by the uncorrupted players, then each uncorrupted player will complete protocol AVSS-Rec.*

Proof: Each uncorrupted player P_i that outputs `Sharing succeeded' has at least $t+1$ uncorrupted players in F_i. It follows from the correctness of Two&Sum-AWSS that the share of each uncorrupted player $P_j \in F_i$ will be successfully reconstructed by P_i in Step 1 of AVSS-Rec. Thus, P_i will have at least $t+1$ non-null shares in Step 2 of AVSS-Rec, and will complete the protocol. □

The Correctness property is proven via Lemmas 18 through 20.

Lemma 18 *Once an uncorrupted player P_i executes Step 7a of AVSS-Share in iteration r, a value $\phi'_m \in \mathbf{F} \cup null$ is fixed for each $P_m \in IS_{i,r}$. Furthermore, if iteration r was successful (as defined in Step 7c of AVSS-Share) then the following properties hold with overwhelming probability:*

1. *For each $P_m \in IS_{i,r}$, $\phi'_m \neq null$.*
2. *The set $C_{i,r} \triangleq \{(m, \phi'_m) | P_m \in IS_{i,r}\}$ defines a secret s'.*
3. *If the dealer is uncorrupted then s' is the secret s shared by the dealer.*
4. *When reconstructing P_m's share using AWSS-Rec$_{m,\mathrm{S}}$ in Step 1 of AVSS-Rec, each uncorrupted player outputs $\phi'_m \cup null$.*

Proof: Define the value ϕ'_m to be the value fixed for the secret shared by P_m in the Two&Sum-AWSS-Share of Step 4. Once P_i executes Step 7a, in iteration r, he has completed the Two&Sum-AWSS-Share of each $P_m \in IS_{i,r}$. It follows from the correctness of Two&Sum-AWSS that the value ϕ'_m is fixed. Part **4** of the lemma also follows.

Part 1. It follows from the correctness of Two&Sum-AWSS that if $\phi'_m = \{null\}$ then for each $l = 1..k$ at least one out of {AWSS-Rec$_{m,\mathrm{A}_{l,i,r}}$, AWSS-Rec$_{m,\mathrm{S}+\mathrm{A}_{l,i,r}}$} will have output *null*. Thus for each $l = 1..k$ P_i will have *null* output with probability at least $\frac{1}{2}$. The probability that P_i has, in Step 7c, non-null output of all the k invocations of AWSS-Rec is at most 2^{-k}. Thus, executions where ϕ'_m =*null* and iteration r is successful occur only with negligible probability.

Part 2. We use the same cut-and-choose argument as in part 1. Assume that $C_{i,r}$ does not define a secret. It follows that for each $l = 1..k$ there exists a P_m such that either the output of AWSS-Rec$_{m,\mathrm{A}_{l,i,r}}$ is not equal to $h_{l,i,r}(m)$ or the output of AWSS-Rec$_{m,\mathrm{S}+\mathrm{A}_{l,i,r}}$ is not equal to

$h_{l,i,r}(m)$ (where $h_{l,i,r}$ is the corresponding polynomial a-casted by the dealer). Thus for each $l = 1..k$ P_i will detect an error with probability at least $\frac{1}{2}$. The probability that P_i has not detected an error in all the k invocations of AWSS-Rec in Step 7c is at most 2^{-k}. Thus, iteration r is successful only with negligible probability.

Part 3. The correctness of Two&Sum-AWSS assures that the value reconstructed in AWSS-Rec$_{m,\mathrm{S}+\mathrm{A}_{l,i,r}}$ equals the sum of the values reconstructed in AWSS-Rec$_{m,\mathrm{A}_{l,i,r}}$ and in AWSS-Rec$_{m,\mathrm{S}}$. Since the dealer is uncorrupted, it a-casts, in Step 3 of AVSS-Share, the same polynomials that it shared in Step 1. Part 3 follows using the same cut-and-choose argument as in parts 1 and 2. $\square$

Lemma 19 *Let $C \stackrel{\triangle}{=} \{(m, \phi'_m)|P_m \in F_i$ and P_i is uncorrupted$\}$, where ϕ'_m is the value fixed for P_m (See Lemma 18). Then the following properties hold with overwhelming probability:*

1. *The set C defines a secret s'.*

2. *If the dealer is uncorrupted then s' is the shared secret, s.*

Proof: 1. Consider two players P_{m_1} and P_{m_2} such that $(m_1, \phi'_{m_1}), (m_2', \phi'_{m_2}) \in C$. Then both P_{m_1} and P_{m_2} were confirmed by uncorrupted players. Assume P_{m_1} was confirmed by uncorrupted player P_{j_1} in iteration r_1, and P_{m_2} was confirmed by uncorrupted player P_{j_2} in iteration r_2. Let $C_{j_1,r_1} \stackrel{\triangle}{=} \{(m, \phi'_m)|P_m \in IS_{j_1,r_1}\}$, and let $C_{j_2,r_2} \stackrel{\triangle}{=} \{(m, \phi'_m)|P_m \in IS_{j_2,r_2}\}$. Then the intersection $I \stackrel{\triangle}{=} C_{j_1,r_1} \cap C_{j_2,r_2}$ is of size at least $t+1$. Lemma 18 implies that the values in I are non-null with overwhelming probability, even if the corresponding players are corrupted. It follows that C_{j_1,r_1} and C_{j_2,r_2} define the same secret. Part 1 follows.
Part **2** follows from part 3 of Lemma 18. $\square$

Lemma 20 *Let r denote the secret defined by the set C (see Lemma 19). An uncorrupted player has output different than r of AVSS-Rec only with negligible probability.*

Proof: Consider an uncorrupted player P_i. The set $\{(m, \phi'_m)|P_m \in F_i\}$ is a subset of C. Thus it defines the same secret r. P_i will have at least $t+1$ uncorrupted players P_j in F_i; the corresponding reconstructed values ϕ'_j will be non-null. Thus P_i will be able to interpolate a value, and this value will be r. $\square$

Lemma 21 (Secrecy): *The Secrecy requirement of Definition 2 is satisfied.*

Proof: Informally, the relevant information in the adversary view of AVSS-Share consists of up to t shares of the polynomial $f(\cdot)$ shared by the dealer in Step 1, and up to t shares of each of the random polynomials shared in Step 2. Furthermore, for each random polynomial $g_*(\cdot)$, only one of $g_*(\cdot)$ or $f(\cdot) + g_*(\cdot)$ is known to the adversary. It can be verified that this information is distributed independently of the shared secret.

A more formal proof follows the lines of the proof of the secrecy property of A-RS-Share (Theorem 3). We omit further details. $\square$

8 Common Coin

We define an asynchronous common coin primitive, and describe a construction. Our construction is a modification of the construction of Feldman [Fel89]. We employ a $(\lceil \frac{n}{3} \rceil - 1)$-resilient AVSS protocol, say the one described in Section 7. A definition of a common coin protocol follows.

Definition 5 *Let π be a protocol where each player has local random input and binary output. We say that π is a $(1-\epsilon)$-*completing*, t-*resilient* p-*Common Coin* protocol if the following requirements hold for every t-adversary:*

- **Completion.** *With probability $1-\epsilon$, all the uncorrupted players complete the protocol.*
- **Correctness.** *For every value $\sigma \in \{0,1\}$, with probability at least p all the uncorrupted players output σ.*

Our construction (with 'completion parameter' ϵ). Roughly speaking, the protocol consists of two stages. First, each player shares n random secrets, using the AVSS-Share protocol of our AVSS protocol, with allowed error parameter $\epsilon' \stackrel{\triangle}{=} \frac{\epsilon}{n^2}$. Say that the ith secret shared by each player is assigned to player P_i. Once a player, P_i, has `sharing succeeded' outputs from $t+1$ AVSS-Share protocols of secrets assigned to it it a-casts the identity of the dealers of these secrets. We say that these $t+1$ secrets are attached to P_i. (Later, the value associated with P_i are computed based on the secrets attached to it.)

Upon having `sharing succeeded' output from the AVSS-Share of all the secrets attached to some P_j, player P_i is certain that a fixed (and yet unknown) value is attached to P_j. (The way in which this value will be computed is described in the protocol.) Once P_i is assured that the values attached to enough players has been fixed, he starts reconstructing the relevant secrets. (This process of ensuring that enough values have been fixed is at the heart of the protocol.) Once all the relevant secrets are reconstructed, each player locally computes its output based on the reconstructed secrets, in a special way described in the sequel. The protocol, given an AVSS protocol $\langle AVSS-Share, AVSS-Rec \rangle$, is presented in Figure 10 below.

Theorem 7 *Let $n \geq 3t+1$. Then, for every $0 < \epsilon \leq 0.2$ protocol* Common-Coin *is a $(1-\epsilon)$-completing, t-resilient $\frac{1}{4}$-common coin protocol for n players. Conditioned on the event that all uncorrupted players complete the protocol, they do so in constant time.*

Proof: The Completion property is asserted in Lemma 22. The Correctness property is asserted in Lemmas 23 through 25. Throughout the proof we assume that the following event E occurs. All invocations of AVSS have been properly completed. That is, if an uncorrupted player has output `sharing succeeded' from AVSS-Share then a value s is fixed. All uncorrupted players will complete the corresponding invocation of AVSS-Rec, outputting s. If the dealer is uncorrupted then s is the shared secret. (See Definition 2.) Event E occurs with probability at least $1 - n^2\epsilon' = 1 - \epsilon$.

Lemma 22 *Conditioned on event E, all the uncorrupted players complete protocol* Common-Coin$[\epsilon]$ *in constant time.*

Proof: Assume event E occurs, and let P_i be an uncorrupted player. Then, P_i will have output `sharing succeeded' from the AVSS-Share protocols initiated by uncorrupted players in Step 1. Thus, $C_i$ will eventually be of size $t+1$, and the message (`Attach` $A_i$ `to` P_i) will be a-casted.

For every uncorrupted player P_j, the a-cast (`Attach` A_j `to` P_j) will be received by P_i. Furthermore, since P_j completed the AVSS-Share$_{k,j}$ protocol for every $P_k \in c_j$, then P_i will complete these AVSS-Share$_{k,j}$ protocols as well. Therefore, every uncorrupted player will eventually be considered accepted by P_i (namely, added to the set G_i). Thus, G_i will eventually be of size $n-t$,

Protocol Common-Coin$[\epsilon]$

Code for player P_i, given 'completion' parameter, ϵ:

1. Let $\epsilon' \stackrel{\triangle}{=} \frac{\epsilon}{n^2}$. For $1 \leq j \leq n$, choose a random secret $x_{i,j} \in_{\mathrm{R}} \mathbf{F}$ and invoke AVSS-Share as a dealer for this value, with completion parameter ϵ'. Denote the execution by AVSS-Share$_{i,j}(x_{i,j})[\epsilon']$.

   Participate in all the other invocations of AVSS-Share. (That is, invoke all the other copies of AVSS-Share as a player.)

2. Create a communication set C_i. Add player P_j to C_i if for all $1 \leq l \leq n$ AVSS-Share$_{j,l}$ has output `` `sharing succeeded´ ``.
   Wait until $|C_i| \geq t+1$. Then, set $c_i = C_i^{(t+1)}$ and a-cast `(Attach` A_i `to` P_i`)` .
   (We say that the secrets $\{x_{j,i} | P_j \in c_i\}$ are the secrets attached to player P_i.)

3. Accept a player P_j if the a-cast `(Attach` A_j `to` P_j`)` has been completed, and $c_j \subseteq C_i$. Let G_i be the set players accepted so far.

   Wait until $|G_i| \geq n-t$. Then, let $g_i = G_i^{(n-t)}$ and a-cast `(`P_i `is ready with` G_i`)`.

4. Say that player P_j is supportive, if the `(`P_j `is ready with` G_j`)` a-cast has been received, and each player in g_j is accepted (namely, if $g_j \subseteq G_i$).

   Wait until $n-t$ players are supportive. Then, raise flag `` `reconstruct enabled´ `` . Let Z_i denote the current contents of G_i.
   (Note that a player P_j who was not considered supportive since some $P_k \in g_j$ was not in G_i can become supportive later if P_k is added to G_i.)

5. Wait until the flag `` `reconstruct enabled´ `` is raised. Then, reconstruct the secrets attached to all the accepted players. That is, for each $P_k \in c_j$ such that $P_j \in G_i$ start running AVSS-Rec$_{k,j}[\epsilon']$, and let $r_{k,j}$ be the corresponding output.

6. If a player becomes accepted *after* the flag `` `reconstruct enabled´ `` has been raised, then start running the corresponding copies of AVSS-Rec.

7. Let $u \stackrel{\triangle}{=} \lceil 0.87n \rceil$. For every player $P_j \in G_i$, let v_j, the value associated with P_j, be the sum modulo u of all the secrets attached to P_j. That is, $v_j = (\sum_{k \in c_j} r_{k,j}) \bmod u$.

   Wait until the values associated with all the players in g_i are reconstructed. (Ie, wait until the corresponding invocations of AVSS-Rec complete.) If there exists a player $P_j \in Z_i$ where $v_j = 0$, output 0 and complete the protocol.
   Otherwise, output 1 and complete the protocol.

Figure 10: The Common Coin protocol

and the message (P_i `is ready with` G_i) will be a-casted Similar reasoning implies that every uncorrupted player will eventually be considered supportive by every uncorrupted player in Step 4. Consequently, every uncorrupted player will raise its `reconstruct enabled' flag, and will invoke its AVSS-Recs of Step 5.

It remains to be shown that all the AVSS-Rec protocols invoked by each uncorrupted player will be completed. If an uncorrupted player received an (`Attach` A_j `to` P_j) a-cast, then all the uncorrupted players will receive this a-cast. Thus, if an uncorrupted player invokes AVSS-Rec$_{j,k}$, then all the uncorrupted players will invoke AVSS-Rec$_{j,k}$. Event E now assures us that all the uncorrupted players will complete all their AVSS-Rec protocols. Therefore, all the uncorrupted players will execute Step 6 and complete the protocol. (The invocations of AVSS-Share with corrupted dealers need not output `sharing succeeded'. Once an uncorrupted player completes Step 6, it may abort all non-completed invocations of AVSS-Share.)

Given event E, all invocations of AVSS-Share and AVSS-Rec complete within a constant number of (asynchronous) rounds. constant time. Protocol a-cast completes in constant time. Consequently, protocol Common-Coin completes in constant time as well. □

Lemma 23 *Let* $u \triangleq \lceil 0.87n \rceil$. *Let* P_i *be a player whose a-cast* (`Attach` A_i `to` P_i) *in Step 2 has been completed by some uncorrupted player. Then, there exists a value,* v_i*, such that all the uncorrupted players associate* v_i *with* P_i *in Step 6. Furthermore,*

- v_i *is fixed once the first uncorrupted player has completed the* (`Attach` A_j `to` P_j) *a-cast.*
- v_i *is distributed uniformly over* $[1 \ldots u]$*, and is independent of the values associated with the other players.*

Proof: Let P_i be a player whose (`Attach` A_i `to` P_i) a-cast of Step 2 has been completed by some uncorrupted player. Then, all the uncorrupted players will complete this a-cast with output c_i. Furthermore, The definition of AVSS assures us that for each player $P_j \in c_i$, there exists a fixed value, $r_{j,i}$, such that all the uncorrupted players have $r_{j,i}$ as their output of AVSS-Rec$_{j,i}[\epsilon']$ in Step 5 (that is, $r_{j,i}$ is the value shared by P_j, and attached to P_i.) Consequently, the value that each uncorrupted player associates with P_i in Step 6 is $\sum_{P_j \in c_i} r_{j,i} \bmod u$; let v_i be this value. This value is fixed once c_i is fixed, namely by the time that the first uncorrupted player has completed the a-cast (`Attach` A_j `to` P_j) .

In remains to show that v_i is uniformly distributed over $[1 \ldots u]$, and is independent of the values associated with the other players. Recall that an uncorrupted player starts reconstructing the secrets attached to P_i (namely, invokes the AVSS-Rec$_{j,i}[\epsilon']$ protocols for $P_j \in c_i$) only *after* it completes the (`Attach` A_i `to` P_i) a-cast. Namely, the set c_i is fixed *before* any uncorrupted player invokes an AVSS-Rec$_{k,i}$ for some k. The Secrecy property of AVSS now assures us that, by the time the set c_i is fixed, the adversary view of the invocations of AVSS-Share where the dealers are uncorrupted is distributed independently of the shared values. Thus, the set c_i, as well as the values that were shared by the corrupted players, are independent of the values shared by uncorrupted players. Furthermore, each set c_i contains at least one uncorrupted player, and uncorrupted players share uniformly distributed, mutually independent values. Consequently, the sum v_i is uniformly and independently distributed over $[1 \ldots u]$. □

Lemma 24 *Assume that some uncorrupted player has raised the* `reconstruct enabled' *flag. Then there exists a set,* M*, such that:*

1. *For each player $P_j \in M$, the* `(Attach` A_j `to` P_j`)` *a-cast of Step 2 has been completed by some uncorrupted player. (Note that Lemma 23 applies to each $P_j \in M$.)*

2. *When any uncorrupted player, P_j, raises its* `` `reconstruct enabled' `` *flag, it will hold that $M \subseteq Z_j$.*

3. $|M| \geq \frac{n}{3}$.

Proof: Let P_i be the first uncorrupted player to raise its `` `reconstruct enabled' `` flag. Let M be the set of players, P_k, for whom $P_k \in g_l$ for at least $t+1$ players P_l who are considered supportive by P_i, in Step 4. We show that the set M has the properties required in the Lemma.

Clearly, $M \subseteq Z_i$. Thus, player P_i has received the a-cast `(Attach` A_k `to` P_k`)` of every player $P_k \in M$. This asserts the first property of M. We now assert the second property. Let $P_k \in M$. An uncorrupted player P_j raises its `` `reconstruct enabled' `` flag, when it has found at least $n-t$ players who are supportive in Step 4. However, $P_k \in g_l$ for at least $t+1$ of the `(`P_l `is ready with` G_l`)` a-casts; thus, there must exist a player P_l such that $P_k \in g_l$ and $g_l \subseteq Z_j$. Consequently, $P_k \in Z_j$.

It remains to show that $|M| \geq \frac{n}{3}$. We use a counting argument. Let $m \triangleq |Z_i|$. We have $m \geq n-t$. Consider the $m \times n$ table T (relative to player P_i), where $T_{l,k} = \mathsf{one}$ iff P_i has received the `(`P_l `is ready with` G_l`)` a-cast and $P_k \in g_l$. The set M is the set of players P_k such that the kth column in T has at least $t+1$ one entries. There are $(n-t)$ one entries in each row of T; thus, there are $m \cdot (n-t)$ one entries in T.

Let q denote the minimum number of columns in T that contain at least $t+1$ one entries. We show that $q \geq \frac{n}{3}$. Clearly, the worst distribution of the one entries in this table is letting q columns be all one's (namely, each of the q columns has m one entries), and letting each of the remaining $(n-q)$ columns have t one entries. This distribution requires that the number of one entries be no more than $q \cdot m + (n-q) \cdot t$. However, there are $m \cdot (n-t)$ one entries in T. Thus, we must have:

$$q \cdot m + (n-q) \cdot t \geq m \cdot (n-t)$$

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

$$q \geq \frac{m(n-t)-nt}{m-t} \geq \frac{(n-t)^2-nt}{n-2t} \geq \frac{(n-2t)^2+nt-3t^2}{n-2t} \geq n-2t+\frac{nt-3t^2}{n-2t} \geq n-2t+\frac{t}{n-2t} > \frac{n}{3}.$$

□

Lemma 25 *Let $\epsilon \leq 0.2$, and assume that all the uncorrupted players have completed protocol* Common-Coin[ϵ]. *Then, for every value $\sigma \in \{0,1\}$, with probability at least* 0.25, *all the uncorrupted players output σ.*

Proof: By Lemma 23 we have that for every player, P_j, who is accepted by some uncorrupted player, there exists a value, v_j, distributed uniformly and independently over $[1 \ldots u]$, such that with probability $1-\frac{\epsilon}{n}$ all the uncorrupted players associate v_j with P_j in Step 6. Consequently, with probability $1-\epsilon$, all the uncorrupted players agree on the value associated with each one of the players.)

Consider the case $\sigma = 0$. Let M be the set of players guaranteed by Lemma 24. Clearly, if $v_j = 0$ for some player $P_j \in M$ and all the uncorrupted players associate v_j with P_j, then all the uncorrupted players output 0. The probability that at least one player $P_j \in M$ has $v_j = 0$ is $1-(1-\frac{1}{u})^{|M|}$. Recall that $u = \lceil 0.87n \rceil$, and that $|M| \geq \frac{n}{3}$. Therefore, for all $n > 4$ we have $1-(1-\frac{1}{u})^{|M|} \geq 1 - \mathrm{e}^{-0.38} \geq 0.316$. Thus, Prob(all the uncorrupted players output 0) $\geq 0.316 \cdot (1-\epsilon) \geq 0.25$.

Consider the case $\sigma = 1$. Clearly, if no player P_j has $v_j = 0$ (and all the uncorrupted players associate v_j with every P_j), then all the uncorrupted players output 1. The probability of this event is at least $(1-\frac{1}{u})^n(1-\epsilon) \geq \mathrm{e}^{-1.15} \cdot 0.8 \geq 0.25$. Thus, Prob(all the uncorrupted players output 1) ≥ 0.25. □

9 Byzantine Agreement

Before describing the Byzantine Agreement protocol, let us describe another protocol used in our construction. Roughly speaking, this protocol, denoted Vote, does 'whatever can be done deterministically' to reach agreement.

9.1 The Voting Protocol

In protocol Vote each player tries to find out whether there is a detectable majority for some value among the (binary) inputs of the players. More precisely, each player's output of the protocol can take five different values. For $\sigma \in \{0,1\}$, the output $(\sigma, 2)$ stands for 'overwhelming majority for σ'. Output $(\sigma, 1)$ stands for 'distinct majority for σ'. Output $(\Lambda, 0)$ stands for 'non-distinct majority'. It will be shown that if all the uncorrupted players have the same input, σ, then all uncorrupted players output $(\sigma, 2)$. Furthermore, if some uncorrupted player outputs $(\sigma, 2)$ then every uncorrupted player will output either $(\sigma, 2)$ or $(\sigma, 1)$. If some uncorrupted player outputs $(\sigma, 1)$ then either all outputs are in $\{(\sigma,2),(\sigma,1)\}$, or all outputs are in $\{(\Lambda,0),(\sigma,1)\}$.

The protocol consists of three 'rounds', having similar structure. In the first round, each player a-casts its input value, waits to complete $n-t$ a-casts of other players, and sets its vote to the majority value among these inputs. In the second round, each player a-casts its vote (along with the identities of the $n-t$ players whose a-casted inputs were used to compute the vote), waits to complete $n-t$ a-casts of other votes that are consistent with the a-casted inputs of the first round, and sets its re-vote to the majority value among these votes. In the third round each player a-casts its re-vote, along with the identities of the $n-t$ players whose a-casted votes 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 the second round.

Now, if all the consistent votes received by a player agree on a value, σ, then this player outputs $(\sigma, 2)$. Otherwise, if all the consistent re-votes received by the player agree on a value, σ, then the player outputs $(\sigma, 1)$. Otherwise, the player outputs $(\Lambda, 0)$. Protocol Vote is presented in Figure 11 below.

In Lemmas 26 through 29 we assert the properties of protocol Vote, as describe above. The Lemmas hold for every input and every t-adversary.

Lemma 26 *All the uncorrupted players complete protocol Vote in constant time.*

Proof: The (Input, P_j, x_j) a-cast of every uncorrupted player P_j in Step 1 will be completed. Thus, every uncorrupted player P_i will eventually have $|\mathbf{a}_i| = n-t$, in Step 2, and will a-cast

Protocol Vote(x_i)

Code for player P_i, on input x_i:

1. A-cast `(Input,`P_i`,`x_i`)` .
2. Define a set $\mathbf{a}_i$. Add (P_j, x_j) to $\mathbf{a}_i$ if the `(Input,`P_j`,`x_j`)` a-cast is completed.

   Wait until $|\mathbf{a}_i| \geq n-t$. Set A_i to $\mathbf{a}_i^{(n-t)}$. Then, set v_i to the majority bit among $\{x_j | (P_j, x_j) \in A_i\}$, and a-cast `(Vote,`$P_i$`,`$A_i$`,`$v_i$`)` .
3. Define a set $\mathbf{b}_i$. Add (P_j, A_j, v_j) to $\mathbf{b}_i$ if the `(Vote,`P_j`,`A_j`,`v_j`)` a-cast has been completed, $a_j \subseteq \mathbf{a}_i$, and v_j is the majority bit of A_j.

   Wait until $|\mathbf{b}_i| \geq n-t$. Set B_i to $\mathbf{b}_i^{(n-t)}$. Then, set rv_i to the majority bit among $\{v_j \, |(P_j, \mathrm{vote}_j, A_j) \in B_i\}$, and a-cast `(Re-vote,`$P_i$`,`$B_i$`,`$rv_i$`)` .
4. Define a set C_i. For each player P_j, wait until the `(Re-vote,`P_j`,`B_j`,`rv_j`)` a-cast has been completed, $B_j \subseteq \mathbf{b}_i$, and rv_j is the majority bit of B_j. Then add (P_j, B_j, rv_j) to C_i
5. Wait until $|C_i| \geq n-t$.
   If all the players $P_j \in B_i$ had the same vote $v_j = \sigma$, then output $(\sigma, 2)$ and complete the protocol.
   Otherwise, if all the players $P_j \in C_i$ have the same re-vote, $rv_j = \sigma$, then output $(\sigma, 1)$ and complete the protocol.

   Otherwise, output $(\Lambda, 0)$ and complete the protocol.

Figure 11: The Vote protocol

$(\texttt{Vote}, P_i, A_i, v_i)$. For every uncorrupted player P_j the a-cast of Step 2 will be completed. An uncorrupted player P_i will add (P_j, A_j, v_j) to $\mathbf{b}_i$ for each uncorrupted P_j, in Step 3. Thus, every uncorrupted player P_i will eventually have $|\mathbf{b}_i| = n-t$, in Step 3, and will a-cast $(\texttt{Re-vote}, P_i, B_i, rv_i)$. Similarly, P_i will add every uncorrupted player P_j to C_i. Thus, every uncorrupted player P_i will eventually have $|C_i| = n - t$, in Step 5. Consequently, P_i will complete the protocol. Furthermore, the protocol runs in constant time. □

Lemma 27 *If all the uncorrupted players have input σ, then all the uncorrupted players output $(\sigma, 2)$.*

Proof: Consider an uncorrupted player P_i. If all the uncorrupted players have input σ, then at most t players will a-cast $\bar{\sigma}$ as their input in Step 1. Therefore, each player P_k who was added to $\mathbf{b}_i$ in Step 3 has a majority for the value σ in its A_k set, and $v_k = \sigma$. Thus, P_i outputs $(\sigma, 2)$ in Step 5. □

Lemma 28 *If some uncorrupted player outputs $(\sigma, 2)$, then each uncorrupted player outputs either $(\sigma, 2)$ or $(\sigma, 1)$.*

Proof: Assume that uncorrupted player P_i outputs $(\sigma, 2)$. The size of B_i is $n - t$, hence for each other uncorrupted player P_j it holds that $B_i \cap B_j$ is of size at least $t+1$. Thus, P_j will set its revote rv_j to σ. Therefore, every uncorrupted player outputs either $(\sigma, 2)$ or $(\sigma, 1)$ in Step 5. □

Lemma 29 *If some uncorrupted player outputs $(\sigma, 1)$, and no uncorrupted player outputs $(\sigma, 2)$, then each uncorrupted player outputs either $(\sigma, 1)$, or $(\Lambda, 0)$.*

Proof: Assume some uncorrupted player outputs $(\sigma, 1)$. Then, at most t players P_j a-casted a revote $rv_j = \bar{\sigma}$ in Step 3; therefore, no uncorrupted player P_j has a unanimous $\textsf{re-vote}_j = \bar{\sigma}$ in Step 4, and no uncorrupted player outputs $(\bar{\sigma}, 1)$. Furthermore, at least $t + 1$ players P_k have a-casted $\textsf{vote}_k = \sigma$ in Step 2; therefore, no uncorrupted player had a unanimous $\textsf{vote}$ in Step 3, and no uncorrupted player outputs $(\bar{\sigma}, 2)$. □

9.2 The Byzantine Agreement protocol

The Byzantine Agreement protocol proceeds in iterations. In each iteration, each player has a 'modified input' value; in the first iteration, the modified input of each player is its local input. In each iteration the players invoke two protocols: Vote and Common-Coin. Protocol Common-Coin is invoked only *after* protocol Vote is completed. (The reason for this provision will become clear in the proof of Lemma 32 below.) If a player recognizes a 'distinct majority' for some value, σ, in the output of protocol Vote (namely output $(\sigma, 1)$ or $(\sigma, 2)$), then it sets its modified input for the next iteration to σ. Otherwise, it sets its modified input for the next iteration to be the output of the Common-Coin protocol. (Protocol Common-Coin is invoked by all players in each iteration, regardless of whether their output is used.) Once a player recognizes an 'overwhelming majority' for some value, σ, (namely, output $(\sigma, 2)$ of protocol Vote), it a-casts σ. Once a player has completed $t + 1$ a-casts with the same output value, σ, it outputs σ and halts. The code of the Byzantine Agreement protocol is presented in Figure 12 below.

In Lemmas 30 through 34 we assert the validity of protocol BA. These Lemmas hold for every input and every t-adversary.

Protocol BA$[\epsilon](x_i)$

Code for player P_i, on input x_i, and completion parameter ϵ:

1. Set $r := 0$. Set $v_1 := x_i$.

Repeat until completing:

2. Set $r := r + 1$. Set $(y_r, m_r) :=$Vote(v_r).
3. Wait until Vote(v_r) is completed. Then, set $c_r :=$Common-Coin$[\frac{\epsilon}{4}]$.
4. (a) If $m_r = 2$, set $v_r := y_r$ and a-cast `(Complete with output` v_r`)` .
   Participate in only one more Vote protocol and only one more Common-Coin protocol.[a]
   (b) If $m_r = 1$, set $v_{r+1} := y_r$.
   (c) Otherwise, set $v_{r+1} := c_r$.

- Upon receiving $t + 1$ `(Complete with output` σ`)` a-casts for some value σ, output σ and complete the protocol.

Figure 12: The Byzantine Agreement protocol

[a]The purpose of this restriction is to prevent the players from participating in an unbounded number of iterations before enough `(Complete with output` σ`)` a-casts are completed.

Lemma 30 *If all the uncorrupted players have input σ, then all the uncorrupted players complete and output σ.*

Proof: Assume that all the uncorrupted players have input σ. By Lemma 27, every uncorrupted player P_i has $(y_1, m_1) = (\sigma, 2)$ by the end of Step 1 of the first iteration. Therefore, every uncorrupted player a-casts `(Complete with output` σ`)` in Step 2 of the first iteration. Therefore, every uncorrupted player will receive at least $n - t$ `(Complete with output` σ`)` a-casts, and at most t `(Complete with output` $\bar{\sigma}$`)` a-casts. Consequently, every uncorrupted player will output σ. □

Lemma 31 *If an uncorrupted player completes with output σ, then all uncorrupted players complete with output σ.*

Proof: Let us first establish that if an uncorrupted player a-casts `(Complete with output` σ`)` for some value σ, then all uncorrupted players eventually a-cast `(Complete with output` σ`)` . Let k be the first iteration in which an uncorrupted player initiated a `(Complete with output` σ`)` a-cast for some value σ. By Lemma 28, every uncorrupted player P_i has $y_k = \sigma$ and either $m_k = 2$ or $m_k = 1$. Therefore, no uncorrupted player a-casts `(Complete with output` $\bar{\sigma}$`)` at iteration k. Furthermore, all the uncorrupted players execute the Vote protocol of iteration $k + 1$ with input σ. Lemma 27 now implies that by the end of Step 1 of iteration $k + 1$, every uncorrupted player has $(y_{k+1}, m_{k+1}) = (\sigma, 2)$. Thus, all the uncorrupted players a-cast `(Complete with output` σ`)` , either at iteration k or at iteration $k + 1$.

Now, assume an uncorrupted player completes with output σ. Thus, at least one uncorrupted player a-casted `(Complete with output` σ`)` . Consequently, all the uncorrupted players a-cast (`Complete with output` σ) . Hence, every uncorrupted player will receive at least $n-t$ (`Complete with output` σ) a-casts, and at most t (`Complete with output` $\bar{\sigma}$) a-casts. Therefore, every uncorrupted player will output σ. □

Lemma 32 *Assume all uncorrupted players have initiated and completed some round k. Then, with probability at least $\frac{1}{4}$ all uncorrupted players have the same value for v_{k+1}.*

Proof: We distinguish two cases. If all the uncorrupted players execute Step 4(c) in iteration k, then all the uncorrupted players set their v_{k+1} value to their (local) output of protocol Common-Coin. In this case all the players have the same v_{k+1} value with probability at least $\frac{1}{2}$.

Otherwise, some uncorrupted player has set $v_{k+1} = \sigma$ for some $\sigma \in \{0,1\}$, either in Step 4(a) or Step 4(b) of iteration k. By Lemma 29, no uncorrupted player will use either Step 4(a) or Step 4(b) to set its v_{k+1} variable to $\bar{\sigma}$. Furthermore, with probability at least $\frac{1}{4}$, all the uncorrupted players have output σ of the Common-Coin protocol of Step 3. Therefore, with probability at least $\frac{1}{4}$, all the uncorrupted players have $v_{k+1} = \sigma$ at the end of iteration k. (Note that the players' outputs of protocol Vote are fixed *before* Common-Coin is invoked. Were it not the case, the uncorrupted players could force the output of protocol Vote to prevent agreement.) □

Let C_k denote the event that each uncorrupted player completes all the iterations it initiated, up to (and including) the kth iteration. (That is, for each iteration $1 \leq l \leq k$ and for each player P, if P initiated iteration l then it computes v_{l+1}.) Let C denote the event that C_k occurs for all k.

Lemma 33 *Conditioned on event C, all the uncorrupted players complete protocol BA in constant expected time.*

Proof: We first establish that all the uncorrupted players complete protocol BA within constant time after the first uncorrupted player initiates a (`Complete with output` σ) a-cast in Step 4(a) of the protocol. Assume the first uncorrupted player initiates a (`Complete with output` σ) a-cast in iteration k. Then, all the uncorrupted players participate in the Vote and Common-Coin protocols of all the iterations up to iteration $k+1$. We have seen in the proof of Lemma 31 that in this case, all the uncorrupted players initiate a (`Complete with output` σ) a-cast by the end of iteration $k+1$. All these a-casts complete in constant time. Each uncorrupted player completes the protocol upon completing $t+1$ of these a-casts. Consequently, once the first uncorrupted player initiates a (`Complete with output` σ) a-cast the protocol completes in constant time.

Let the random variable τ count the number of iterations until the first uncorrupted player a-casts (`Complete with output` σ) . (If no uncorrupted player a-casts (`Complete with output` σ) then $\tau = \infty$). Conditioned on event C, all the uncorrupted players complete each iteration in constant time. It is left to show that $\mathrm{E}(\tau|C)$ is constant. We have

$$\mathrm{Prob}(\tau > k|C_k) \leq \mathrm{Prob}(\tau \neq 1|C_k) \cdot \ldots \cdot \mathrm{Prob}(\tau \neq k|C_k \cap \tau \neq 1 \cap \ldots \cap \tau \neq k-1)$$

If follows from Lemma 32 that each one of the k multiplicands of the right hand side of the above equation is at most $\frac{3}{4}$. Thus, $\mathrm{Prob}(\tau > k|C_k) \leq \left(\frac{3}{4}\right)^k$. It follows, via a simple calculation, that $\mathrm{E}(\tau|C) \leq 16$. □

Lemma 34 $\mathrm{Prob}(C) \geq 1-\epsilon$.

Proof: We have

$$\begin{aligned}\mathrm{Prob}(\overline{C}) &\leq \sum_{k\geq 1}\mathrm{Prob}(\tau > k \cap \overline{C_{k+1}}|C_k) &(1)\\ &\leq \sum_{k\geq 1}\mathrm{Prob}(\tau > k|C_k)\cdot \mathrm{Prob}(\overline{C_{k+1}}|C_k \cap \tau > k) &(2)\end{aligned}$$

We have seen in the proof of Lemma 33 that $\mathrm{Prob}(\tau > k|C_k) \leq \left(\frac{3}{4}\right)^{k-1}$. We bound the term $\mathrm{Prob}(\overline{C_{k+1}}|C_k \cap \tau \geq k)$. If all the uncorrupted players execute the kth iteration and complete the kth invocation of Common-Coin, then all the uncorrupted players complete the kth iteration. Protocol Common-Coin is invoked with 'completion parameter' $\frac{\epsilon}{4}$. Thus, with probability $1-\frac{\epsilon}{4}$, all the uncorrupted players complete the kth invocation of Common-Coin. Therefore, for each k, $\mathrm{Prob}(\overline{C_{k+1}}|C_k \cap \tau \geq k) \leq \frac{\epsilon}{4}$. Inequality 1 yields $\mathrm{Prob}(\overline{C}) \leq \sum_{k\geq 1}\frac{\epsilon}{4}\left(\frac{3}{4}\right)^{k-1} = \epsilon$. □

We have thus shown:

Theorem 2 (Byzantine Agreement.) *Let $n \geq 3t+1$. Then, for every $0 < \epsilon \leq 0.2$, protocol* BA$[\epsilon]$ *is a $(1-\epsilon)$-completing, t-resilient, asynchronous Byzantine Agreement protocol for n players. Given that the players complete, they do so in constant expected time. Furthermore, the computational resources required of each player are polynomial in n and* $\log\frac{1}{\epsilon}$.

Acknowledgments

A special thank is due to Michael Rabin for his patience and wise advice. In addition we thank Oded Goldreich and Michael Ben-Or for commenting on our drafts. In addition, we warmly thank Boaz Kelmer. Finally, we thank the anonymous referees for their thoughtful remarks that greatly improved the presentation of the paper.

References

[AFL83] E. Arjomandi, M. Fischer, and N. Lynch. Efficiency of Synchronous Versus Asynchronous Distributed Systems. *Journal of the ACM*, 30(3):449–456, 1983.

[BCG93] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous Secure Computations . In *Proceeding 25th Annual Symposium on the Theory of Computing*. ACM, 1993.

[BE91] M. Ben-Or and R. El-Yaniv. Interactive Consistency in Constant Time. Submitted for publication, 1991.

[Ben83] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols . In *Proceeding of 2nd Annual ACM Symposium on Principles of Distributed Computing*, pages 27–30. ACM, 1983.

[BKR94] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous Secure Computations with Optimal Resilience. In *Proceeding Thirteenth Annual Symposium on Principles of Distributed Computing*, pages 183–192. ACM, 1994.

[Bra84] G. Bracha. An Asynchronous $\lfloor (n-1)/3 \rfloor$-resilient Consensus Protocol . In *Proceeding 3rd Annual Symposium on Principles of Distributed Computing*, pages 154–162. ACM, 1984.

[CC85] B. Chor and B. Coan. A simple and efficient randomized Byzantine agreement algorithm. *IEEE Transactions on Software Engineering*, SE-11(6):531–539, June 1985.

[CCD88] D. Chaum, C. Crepeau, and I. Damgard. Multiplayer Unconditionally Secure Protocols. In *Proceeding 20th Annual Symposium on the Theory of Computing*, pages 11–19. ACM, 1988.

[CD89] B. Chor and C. Dwork. Randomization in Byzantine Agreement. *Advances in Computing Research*, 5:443–497, 1989.

[CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *Proceeding 26th Annual Symposium on the Foundations of Computer Science*, pages 383–395. IEEE, 1985.

[CR93] R. Canetti and T. Rabin. Optimal Asynchronous Byzantine Agreement. In *Proceeding 25th Annual Symposium on the Theory of Computing*, pages 42–51. ACM, 1993.

[Fel89] P. Feldman. Asynchronous Byzantine Agreement in Constant Expected Time. Manuscript, 1989.

[Fis83] M. Fischer. The Consensus Problem in Unreliable Distributed System. Technical report, Department of Computer Science, Yale University, 1983.

[FLP85] M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382, 1985.

[FM88] P. Feldman and S. Micali. An Optimal Algorithm for Synchronous Byzantine Agreement . In *Proceeding 20th Annual Symposium on the Theory of Computing*, pages 148–161. ACM, 1988.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[GM98] J. Garay and Y. Moses. Fully Polynomial Byzantine Agreement in $t+1$ Rounds. *SIAM Journal on Computing*, 27(1), 1998.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *SIAM. J. Computing*, 18(1):186–208, February 1989.

[KY86] A. Karlin and A. Yao. Probabilistic Lower Bounds for Byzantine Agreement. Manuscript, 1986.

[LT89] N. Lynch and M. Tuttle. An introduction to input/output automata. *CWI-Quarterly*, 2(3):219–246, 1989.

[Lyn96] N. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM*, 27(2):228–234, 1980.

[Rab83] M. Rabin. Randomized Byzantine Generals. In *Proceeding 24th Annual Symposium on the Foundations of Computer Science*, pages 403–409. IEEE, 1983.

[Rab94] T. Rabin. Robust Sharing of Secrets When the Dealer is Honest or Faulty. *Journal of the ACM*, 41(6):1089–1109, 1994.

[RB89] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiplayer Protocols with Honest Majority. In *Proceeding 21st Annual Symposium on the Theory of Computing*, pages 73–85. ACM, 1989.

[Sha79] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22:612–613, 1979.

[SL95] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.