Almost-Surely Terminating Asynchronous Byzantine Agreement Revisited

Laasya Bangalore* Ashish Choudhury† Arpita Patra‡

Abstract

The problem of Byzantine Agreement (BA) is of interest to both distributed computing and cryptography community. Following well-known results from the distributed computing literature, BA problem in the *asynchronous* network setting encounters inevitable non-termination issues. The impasse is overcome via randomization that allows construction of BA protocols in two flavours of termination guarantee – with overwhelming probability and with probability one. The latter type termed as *almost-surely terminating* BAs are the focus of this paper. An eluding problem in the domain of almost-surely terminating BAs is achieving a constant expected running time. Our work makes progress in this direction.

In a setting with n parties and an adversary with unbounded computing power controlling at most t parties in Byzantine fashion, we present two almost-surely terminating BA protocols in the asynchronous setting:

- With the *optimal resilience* of $t < \frac{n}{3}$, our first protocol runs for expected $\mathcal{O}(n)$ time. The existing protocols in the same setting either runs for expected $\mathcal{O}(n^2)$ time (Abraham et al, PODC 2008) or requires exponential computing power from the honest parties (Wang, CoRR 2015). In terms of communication complexity, our construction outperforms all the known constructions that offer almost-surely terminating feature.
- With the resilience of $t < \frac{n}{3+\epsilon}$ for any $\epsilon > 0$, our second protocol runs for expected $\mathcal{O}(\frac{1}{\epsilon})$ time. The expected running time of our protocol turns constant when ϵ is a constant fraction. The known constructions with constant expected running time either require ϵ to be at least 1 (Feldman-Micali, STOC 1988), implying $t < n/4$, or calls for exponential computing power from the honest parties (Wang, CoRR 2015).

We follow the traditional route of building BA via common coin protocol that in turn reduces to *asynchronous verifiable secret-sharing* (AVSS). Our constructions are built on a variant of AVSS that is termed as *shunning*. A shunning AVSS fails to offer the properties of AVSS when the corrupt parties strike, but allows the honest parties to *locally* detect and shun a set of corrupt parties for any future communication. Our shunning AVSS with $t < n/3$ and $t < \frac{n}{3+\epsilon}$ guarantee $\Omega(n)$ and respectively $\Omega(\epsilon t^2)$ conflicts to be revealed when failure occurs. Turning this shunning AVSS to a common coin protocol efficiently constitutes yet another contribution of our paper.

*International Institute of Information Technology, Bangalore India. Email: B.Laasya@iiitb.org. Financial support from Infosys foundation acknowledged.

†International Institute of Information Technology, Bangalore India. Email: ashish.choudhury@iiitb.ac.in. Financial support from Infosys foundation acknowledged.

‡Indian Institute of Science, Bangalore India. Email: arpita@iisc.ac.in. Arpita Patra would like to acknowledge the financial support by SERB Women Excellence Award from Science and Engineering Research Board of India and INSPIRE Faculty Fellowship from Department of Science & Technology, India.

1 Introduction

Byzantine Agreement (BA) [20] is a fundamental problem in secure distributed computing. Informally, a BA protocol allows a set of n parties, each holding a private bit, to agree on a common bit, tolerating a *computationally unbounded* adversary who can corrupt any t parties in a *Byzantine* fashion. In the literature, BA has been studied in two prominent network setting – synchronous and asynchronous. In the synchronous setting, it is assumed that the delay of messages over the channels of the network is bounded by a *known constant*. In contrast, in the asynchronous setting, the channels may have arbitrary delays and may deliver messages in any arbitrary order, with the only restriction that every sent message is eventually delivered. To model the worst case, the adversary is allowed to control the scheduling of messages in the network. While the BA problem has been investigated extensively in the *synchronous* setting (see [17, 15, 2] and their references), the progress for the asynchronous setting that models real-world networks like the Internet more appropriately, has been rather slow. The primary difficulty in designing a distributed protocol in the latter setting comes from the fact that it is impossible for an honest party to distinguish between a slow but honest sender (whose messages are delayed) and a corrupt sender (who did not send any message). Hence, at any stage of an asynchronous protocol, a party cannot wait to receive messages from all the n parties (to avoid endless waiting) and the communication from t (potentially slow but honest) parties may have to be ignored.

The condition $t < n/3$ is necessary and sufficient for asynchronous BA (ABA) [20]. An ABA protocol designed with exactly $n = 3t + 1$ parties is thus termed as *optimally-resilient*. From [14], any ABA protocol must have non-terminating runs, where some honest party(ies) may not terminate at all. Use of randomization helps circumvent this impasse [4, 21, 6], leading to two types of constructions. The first kind, known as $(1 - \lambda)$-terminating ABA [9, 8, 19], allow the honest parties to terminate with probability at least $(1 - \lambda)$, for some λ that is non-zero yet negligible. The second kind, referred to as *almost-surely terminating* ABA protocols [1], ensure turning the probability of the occurrence of a non-terminating execution to asymptotically zero, making the honest parties terminate with probability 1. Our focus in this paper is the latter kind of ABA. The historical constructions [4, 6] in this domain require an exponential expected running time (ERT) (and therefore requires exponential expected communication as well as computation complexity). The work of [13] reduces the ERT to a constant at the expense of non-optimal resilience of $t < n/4$. The state-of-the-art constitutes the works of [1] and [22]. The former presents the first construction that offers optimal resilience and polynomial ($\mathcal{O}(n^2)$, to be specific) ERT simultaneously. The latter reduces the ERT to $\mathcal{O}(n)$ at the expense of requiring exponential computation from the honest parties. A side result in the same paper makes the ERT $\mathcal{O}(1/\epsilon)$ while decreasing the resilience to $t < \frac{n}{3+\epsilon}$ for any $\epsilon > 0$. Since then constructing an optimally-resilient, polynomial computation complexity ABA with a constant ERT remains an eluding problem. Our work, elaborated below, makes progress in this direction.

Our Results. We present two almost-surely terminating polynomial-time ABA protocols. With $t < \frac{n}{3}$, our first protocol has $\mathcal{O}(n)$ ERT, improving the ERT of [1] by a factor of n. With $t < \frac{n}{3+\epsilon}$ for any $\epsilon > 0$, our second protocol has ERT $\mathcal{O}(\frac{1}{\epsilon})$. The ERT becomes a constant when ϵ is a constant fraction. Previously, constant ERT was achieved either with $\epsilon \geq 1$ (implying $t < n/4$) [13] or at the cost of exponential computing power from honest parties [22]. In terms of expected communication, both our constructions outperform all the known constructions in their respective settings. Our results put in the context of relevant existing results are presented below. $\mathbb{F}$ denotes a finite field over which computations are done.

Reference	Resilience	ERT (R)	Expected Communication Complexity	Computation Complexity (in n)
[13]	$n > 4t$	$\mathcal{O}(1)$	$\mathcal{O}(n^6 \log n \log \lvert\mathbb{F}\rvert)$	Polynomial
[1]	$n > 3t$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{10} \log \lvert\mathbb{F}\rvert)$	Polynomial
[22]	$n > 3t$	$\mathcal{O}(n)$	$\mathcal{O}(n^7 \log \lvert\mathbb{F}\rvert)$	Exponential
[22]	$n > (3+\epsilon)t, \epsilon > 0$	$\mathcal{O}(1/\epsilon)$	$\mathcal{O}(n^7 \log \lvert\mathbb{F}\rvert)$	Exponential
This paper	$n > 3t$	$\mathcal{O}(n)$	$\mathcal{O}(n^6 \log \lvert\mathbb{F}\rvert)$	Polynomial
This paper	$n > (3+\epsilon)t, \epsilon > 0$	$\mathcal{O}(1/\epsilon)$	$\mathcal{O}(n^6 \log \lvert\mathbb{F}\rvert)$	Polynomial

Our Approach. Most randomized ABA protocols follow the blueprint of [4, 21, 6] by reducing ABA to the design of a *common coin* (CC) protocol, which allows the parties to output common randomness with a certain success probability. The success probability that inversely impacts the running time of the ABA is desired to be a constant fraction, for an ABA protocol to have a constant expected running time. [13] showed how to implement CC with a constant success probability using an *asynchronous verifiable secret sharing* (AVSS) scheme. AVSS is a two phase protocol (Sharing and Reconstruction) carried out among n parties with a designated party called *dealer* D in the presence of an adversary who can corrupt up to any t parties including the dealer. The goal is to let D share a secret s, among the n parties during the sharing phase in a way that would later allow for a unique reconstruction of s in the reconstruction phase, irrespective of whether D is honest or corrupt (correctness), while preserving the secrecy of s until the reconstruction phase, if D is honest (privacy).

[13] shows a perfect (no error) AVSS protocol with $t < n/4$ and turns it into an almost-surely terminating ABA with the same resilience. AVSS protocols with $t < n/3$ and with λ probability of non-termination are proposed in [9, 19] and are used for building $(1-\lambda)$-terminating ABA with a constant expected running time. The work of [1] introduces a weaker form of AVSS called *shunning asynchronous verifiable secret sharing* (SAVSS) with just $t < n/3$ that suffices to construct an almost-surely terminating ABA. An SAVSS fails to offer the correctness property of AVSS when the corrupt parties strike, but allows some honest party to locally detect and shun at least one corrupt party for all future communication. Importantly, the SAVSS scheme *always* terminates. Building on the SAVSS scheme, [1] designs a *shunning common coin* (SCC) protocol. Being a shunning variant of CC, it ensures that either the properties of CC are satisfied or at least one local fault detection occurs along with eventual termination guarantee. Since there are $\mathcal{O}(n^2)$ pairs of honest and corrupt parties, there may be $\mathcal{O}(n^2)$ 'failed' (but terminating) SAVSS instances (where the correctness is violated) before hitting a correct SAVSS instance. Consequently, a correct CC instance may take $\mathcal{O}(n^2)$ 'failed' SCC instances to run, making the ABA protocol of [1] run for $\mathcal{O}(n^2)$ expected running time. [22] boosts the fault detection capability of their SAVSS[1] and SCC by a factor of $\Omega(n)$ to come up with an ABA protocol with $\mathcal{O}(n)$ expected running time, but at the expense of exponential computation complexity. See Appendix A for the analysis of the protocols of [1, 22].

At the heart of our ABA lies a new SAVSS that catches asymptotically the same number of local faults as [22] while requiring only polynomial computation complexity. But we get this at the cost of having a termination guarantee that is weaker than the prior SAVSSs. Namely, there may not be a termination guarantee at all, but in this case, our protocol helps *all* the honest parties to shun $\Omega(t)$ corrupt parties. This leads to a weaker form of SCC that inherits the same termination property. We then turn this weak SCC to one that always terminates by running a constant number of weak SCC instances. The novelty lies in interleaving these weak SCC instances and combining their outputs in a subtle way to achieve a constant fraction success probability. Interestingly, we show that 3 instances are sufficient to achieve an SCC with a success probability of $\frac{1}{4}$. Lastly, we note that our protocol improves over the expected communication complexity of the SAVSS schemes of [1, 22] and is conceptually simpler than the construction of [1] that is built on top of yet another primitive called *moderated weak SVSS* (MW-SVSS). We now elaborate on our SAVSS with $t < n/3$.

High Level Overview of Our SAVSS. Our starting point is the MW-SVSS of [1], based on the idea of sharing the secret by embedding it in the constant term of a t-degree symmetric bivariate polynomial $F(x, y)$, where each party P_i receives $f_i(x) = F(x, i)$. The parties then exchange common values with each other to identify a set of $n - t$ parties referred as $\mathcal{V}$, such that each $P_i \in \mathcal{V}$ is confirmed by a set of $n - t$ parties, $\mathcal{V}_i$, about the receipt of consistent polynomials i.e. $f_i(j) = f_j(i)$ holds for each $P_j \in \mathcal{V}_i$. In the reconstruction phase, the polynomial of every P_i from $\mathcal{V}$ is reconstructed from the values produced by any $n-t-t = n-2t$ parties in $\mathcal{V}_i$. For a guaranteed termination, $n - 2t$ is the maximum number of values that can be waited for in the asynchronous setting. This construction fails to be an SAVSS, satisfying only a weaker notion of

[1] In [22], the SAVSS scheme is called *inferable verifiable secret sharing* (IVSS).

MW-SVSS due to the following: for a *corrupt* D and a *corrupt* P_i from $\mathcal{V}$, an incorrect polynomial of P_i may get reconstructed leading to the violation of correctness, while none of the corrupt parties are shunned. When at least one of D or P_i is *honest*, then either $f_i(x)$ is correctly reconstructed or the honest party between D and P_i can shun the corrupt P_j from $\mathcal{V}_i$ that contributes an incorrect value of $f_i(x)$. We turn this construction to an SAVSS that fails to guarantee correctness at the cost of *locally* catching $\Omega(n)$ faults (irrespective of D) and fails to offer termination at the cost of *globally* shunning $\Omega(n)$ corrupt parties.

While the sharing phase remains to be almost the same, except for the construction of $\mathcal{V}$, we bring the following modifications in the reconstruction phase. First, $n-t-\frac{t}{2}$ values (instead of $n-2t$ values) are taken into account for reconstructing a polynomial. The protocol will not terminate if some $t/2+1$ corrupt parties from one of the $\mathcal{V}_i$ sets do not respond. But *all* the honest parties can keep a tab and shun the non-responsive parties for all future instances. This is the guarantee we promised when termination is not met. Second, the reconstruction of a polynomial is made robust against at most $t/4$ faulty values. This is achieved via Reed-Solomon (RS) error-correction [18] (see the next section for the formal details) that allows error correction of upto $t/4$ values when at least $n-t-t/2$ values of a t-degree polynomial are received and indeed up to $t/4$ values are faulty, provided $n=3t+1$. More than $t/4$ parties in some $\mathcal{V}_i$ must contribute incorrect values for P_i's polynomial to be reconstructed incorrectly, leading to the breach of correctness. But some honest parties will be able to locally identify those parties who contributed incorrectly and shun them. This is the guarantee we promised when correctness fails. We note that this guarantee is achieved, even if *both* D as well as P_i are corrupt. This is due to our modified construction of $\mathcal{V}$; we defer the details to Section 3.

Other Related Work. In this work, we consider pair-wise secure channel model, where the parties are assumed to be connected by pair-wise private and authentic channels and the adversary cannot read the messages exchanged between the honest parties; the adversary can only control message scheduling, without preventing the messages of honest parties from being delivered indefinitely. In [10, 5, 16], BA protocols in the full-information model are presented, where adversary is computationally unbounded and can even listen the communication happening between any pair of honest parties. In [7], computationally-secure ABA protocols are presented, assuming a computationally bounded adversary and a public-key infrastructure (PKI) set-up.

2 Preliminaries

We assume a set of n parties $\mathcal{P}=\{P_1,\ldots,P_n\}$, connected by pair-wise private and authentic channels. A *computationally unbounded* adversary A can corrupt any $t<n/3$ parties during the execution of a protocol and force the parties under its control (called corrupted parties) to behave in any arbitrary manner during the protocol execution. For simplicity, we assume the adversary is *static*, who decides the set of corrupted parties at the beginning of the execution of a protocol. However, our protocols remain secure even against a more powerful *adaptive* adversary, who decides the set of corrupt parties during the run time, depending upon the values seen so far during the execution of a protocol. For simplicity, we assume $n=3t+1$, so that $t=\Theta(n)$. In our protocols, all computation and communication are done over a finite field $\mathbb{F}$, where $|\mathbb{F}|>2n$.

The communication channels are asynchronous allowing arbitrary, but a finite delay (the messages sent by the honest parties reach their destinations eventually). The order of the message delivery is decided by a *scheduler*, controlled by A. The scheduler can only schedule the messages exchanged between the honest parties, without having access to the "contents" of these messages. A protocol execution is considered as a sequence of *atomic steps*, where a single party is *active* in each such step. A party is activated when it receives a message. On receiving a message, it performs an internal computation and then possibly sends messages on its outgoing channels. The order of the atomic steps is controlled by the scheduler. At the beginning of a protocol, each party will be in a special *start* state. A party is said to *terminate/complete* a protocol if it reaches a *halt* state, after which it does not perform any further computation. A protocol execution is said to

be complete if all the honest parties terminate the computation. We measure the running time of a protocol following [8]. Consider a virtual "global clock" measuring time in the network, with no party having access to it. The *delay* of a message transmission denotes the time elapsed from its sending to its reception. The *period* of a finite execution of a protocol is the longest delay of any message transmission during the execution. Let the *duration* of a finite execution denote the total time measured by the global clock divided by the period of this execution. The *expected running time* of a protocol, is the maximum over all inputs and applicable adversaries, of the average of the duration of executions of the protocol over the random inputs of the parties.

In our protocols, each P_i maintains a local "block" set $\mathcal{B}_i$ for all protocol instances and a "wait" set $\mathcal{W}_i$, which are initialized to $\emptyset$. It is to be noted that P_i maintains a *single* local set $\mathcal{B}_i$, where as a *separate* set $\mathcal{W}_i$ is maintained for each SAVSS protocol instance. P_i includes P_j in $\mathcal{B}_i$ if during some protocol instance, x is expected from P_j, but instead $x' \neq x$ is received. An *honest* P_i is said to be in *local conflict* with P_j when $P_j \in \mathcal{B}_i$. P_i includes P_j in $\mathcal{W}_i$ corresponding to some SAVSS protocol instance, if during that instance, P_i is expecting some communication from P_j. While a party taking entry in $\mathcal{B}_i$ remains part of it until the end of the execution of the ABA protocol, any entry in a wait list is temporary and removed as and when the expected communication is received. Until the receipt of the desired communication from a party in $\mathcal{W}_i$, party P_i suspends (saves yet does not use) its future communication.

Definition 2.1 (**Shunning Asynchronous Verifiable Secret Sharing (SAVSS)).** Let $(\mathsf{Sh}, \mathsf{Rec})$ be a pair of protocols for the n parties in $\mathcal{P}$, each maintaining a local $\mathcal{B}_i$ and $\mathcal{W}_i$ set, and for a special party *dealer* $\mathsf{D} \in \mathcal{P}$ that has a private input $s \in \mathbb{F}$ for Sh. $(\mathsf{Sh}, \mathsf{Rec})$ is an SAVSS scheme if the following requirements hold for every possible A.

- **Termination**: **(a)**: If D is *honest* and all honest parties participate in Sh, then each honest party eventually terminates Sh. **(b)**: If some honest party terminates Sh, then every other honest party eventually terminates Sh. **(c)**: If all honest parties participate in Rec, then *one* of the following holds: **(c.i)**: All honest parties eventually terminate Rec; or **(c.ii)**: Some corrupt parties are included in the $\mathcal{W}$ sets of some honest parties.
- **Correctness**: If the honest parties terminate Rec, then there is a unique value $\bar{s}$ where $\bar{s} = s$ for an honest D and $\bar{s} \in \mathbb{F} \cup \{\perp\}$ for a corrupt D, such that *one* of the following holds: **(a)**: All honest parties output $\bar{s}$ at the end of Rec; or **(b)**: Some corrupt parties are included in the $\mathcal{B}$ sets of some honest parties.
- **Privacy**: If D is *honest*, then the view of A during Sh is independent of s.

Definition 2.2 (**Weak Shunning Common Coin (WSCC)).** Let Π be a protocol for the n parties in $\mathcal{P}$, each maintaining a local $\mathcal{B}_i$ and $\mathcal{W}_i$ set, where each party has some local random input and a possible binary output. Then Π is said to be a (p_0, p_1)-weak shunning common coin (WSCC) protocol, if the following hold for every possible A – **Correctness**: If all the honest parties obtain their output, then one of the following holds: **(a)**: For every value $\sigma \in \{0, 1\}$, with probability at least p_σ, all honest parties output σ; or **(b)**: Some corrupt parties are included in the $\mathcal{B}$ sets of some honest parties.

Definition 2.3 (**Shunning Common Coin (SCC)).** A protocol Π is a p-shunning common coin (SCC) if it is a (p, p)-WSCC and additionally satisfies the following termination guarantee for every possible A – **Termination**: If all honest parties participate in Π, then all honest parties eventually terminate Π.

We stress that we do not explicitly define any termination property for WSCC; the parties may continue running the protocol, even after obtaining output. Our WSCC will be used as a blackbox to design our SCC protocol. And the termination of our SCC ensures that all the underlying instances of WSCC also terminate.

Definition 2.4 (**Almost-Surely Terminating Asynchronous Byzantine Agreement (ABA)).** Let Π be a protocol for the n parties in $\mathcal{P}$, where each party has a binary input x_i and a binary output σ_i. Then, Π is said to

be an almost-surely terminating ABA protocol, if the following hold for every possible A – **(a) Termination**: If all honest parties participate, then with probability one, all honest parties eventually terminate Π. **(b) Agreement**: $\sigma_i = \sigma_j$ holds for every honest P_i and P_j. **(c) Validity**: If all honest parties have the same input $x \in \{0,1\}$, then $\sigma_i = x$ holds for every honest P_i.

Existing Tools and Primitives. In our protocols, we use univariate and bivariate polynomials over $\mathbb{F}$. A t-degree univariate polynomial is of the form $f(x) = a_0 + a_1 x + \ldots + a_t x^t$, where each $a_i \in \mathbb{F}$. A t-degree symmetric bivariate polynomial is of the form $F(x,y) = \sum_{i=0}^{i=t} \sum_{j=0}^{j=t} r_{ij} x^i y^j$, where $r_{ij} = r_{ji}$ for $i,j = 1,\ldots,t$ and each $r_{ij} \in \mathbb{F}$. We use the following property of Reed-Solomon (RS) codes. Let $f(x)$ be an arbitrary unknown t-degree polynomial and let $\mathcal{K} = \{(i_1, v_{i_1}), \ldots, (i_N, v_{i_N})\}$ be a set of N points such that at most c of them do not lie on $f(x)$ i.e. $v_{i_k} \neq f(i_k)$ holds for c pairs. RS decoding algorithm $\mathsf{RS\text{-}Dec}(t,c,\mathcal{K})$ allows correct reconstruction of the polynomial $f(x)$ from $\mathcal{K}$ if and only if $N \geq t+1+2c$ [18].

We use the *asynchronous reliable broadcast* protocol of Bracha with $n = 3t+1$ [6], which allows a *sender* $\mathsf{S} \in \mathcal{P}$ to identically send some message m to all the parties in $\mathcal{P}$. It ensures that: (a) if S is *honest*, then every honest party eventually terminates the protocol with output m; (b) if S is *corrupt* and some honest party terminates with output $m^\star$, then every other honest party eventually terminates with output $m^\star$. The protocol has communication complexity of $\mathcal{O}(n^2)$ bits over point-to-point channels to broadcast a single bit message. We use the term P_i *broadcasts* m to mean that P_i acts as S and invokes an instance of Bracha's protocol to broadcast m. Similarly, the term P_j *receives* m *from the broadcast of* P_i means that P_j (as a receiver) completes the execution of P_i's broadcast (namely the instance of broadcast protocol where P_i is S), with m as output. $\mathcal{BC}(x)$ denotes the communication complexity of broadcasting x bits using Bracha's protocol.

3 Our SAVSS Protocol with $n = 3t+1$

D hides its secret s in the constant term of a t-degree symmetric bivariate polynomial $F(x,y)$ and gives the ith t-degree univariate polynomial $f_i(x) = F(x,i)$ to each P_i. For an honest D, every pair of *honest* parties P_i, P_j should be "pair-wise consistent" and $f_i(j) = f_j(i)$ should hold. So each P_i is designated as a "guard" to verify the pair-wise consistency of $f_i(x)$ with $n-t$ parties P_j, who are considered as "sub-guards" for P_i. Each P_i, P_j privately exchanges the common value of $f_i(x)$ and $f_j(x)$ and then publicly announce the status of consistency check. The goal is then to identify if D has given t-degree univariate polynomials to at least $n-t$ guards, say $\mathcal{V}$, such that each guard P_i in $\mathcal{V}$ has further verified the pair-wise consistency of its polynomial $f_i(x)$ with at least $n-t$ sub-guards, denoted by $\mathcal{V}_i$. Moreover, for every guard P_i in $\mathcal{V}$, every sub-guard in $\mathcal{V}_i$ should be from $\mathcal{V}$, implying that sub-guard P_j of *any* guard is itself a guard. To ensure that all parties identify the same $\mathcal{V}$ and sub-guard lists, D is assigned the task of identifying $\mathcal{V}$ and the sub-guard lists and broadcast the same. The parties terminate the sharing phase, if D broadcasts a legitimate $\mathcal{V}$ and sub-guard lists. If D is *honest*, then the set of honest parties always constitute a candidate $\mathcal{V}$ and hence the sharing phase always terminates for an honest D.

During the reconstruction phase, the goal is to reconstruct back t-degree univariate polynomials of *all* the guards in $\mathcal{V}$ and then using them, interpolate the t-degree bivariate polynomial committed by D. Due to asynchrony, the parties cannot afford for *each* guard in $\mathcal{V}$ to reveal its univariate polynomial. Instead, the univariate polynomial of each guard P_j is reconstructed using the points held by the sub-guards P_k in $\mathcal{V}_j$ who are asked to publicly reveal their respective univariate polynomial $f_k(x)$. We wait for $n - t - \frac{t}{2} = \frac{3t}{2} + 1$ sub-guards in $\mathcal{V}_j$ to respond. Then using the $\frac{3t}{2}+1$ revealed $f_j(k)$ values, we try to error-correct $\frac{t}{4}$ errors and interpolate back $f_j(x)$. When at most $\frac{t}{4}$ sub-guards out of the $\frac{3t}{2}+1$ sub-guards produce incorrect polynomials, the reconstructed polynomial $f_j(x)$ is correct. Otherwise, at least $\frac{t}{4}+1$ *corrupt* sub-guards P_k who produced wrong polynomials and are also guards will be identified locally as corrupt by at least one *honest* sub-guard P_l in $\mathcal{V}_k$. The reconstruction phase may not terminate if less than $\frac{3t}{2}+1$ parties from $\mathcal{V}_j$

participate for some guard P_j. But this will lead to communication from at least $\frac{t}{2}+1$ corrupt sub-guards in $\mathcal{V}_j$ being marked as "pending" by *all* honest parties. As a consequence, in any subsequent SAVSS instance, there will be at most $\frac{t}{2}-1$ corrupt participants, resulting in at least $\frac{3t}{2}+1$ *honest* sub-guards in each sub-guard list. So the adversary cannot disrupt the correctness or fail termination of any further SAVSS reconstruction instance.

The sharing and reconstruction protocols are presented in Fig 1. Each instance of SAVSS is associated with a unique id $\mathsf{sid} \in \mathbb{N}$. All messages communicated during the SAVSS instance sid are tagged with this id. However, we skip tagging every message explicitly with id for simplicity. In the sharing phase, once $\mathcal{V}$ is agreed upon, then the parties locally populate their respective $\mathcal{W}$ sets, anticipating the values they expect from the various guards and sub-guards during the reconstruction phase. At the beginning of each instance of the SAVSS, a corresponding *memory management* protocol SAVSS-MM is invoked (Fig 2), according to which each party locally decides whether to process a received message as per the SAVSS, delay it temporarily or block it permanently. The SAVSS-MM protocol for id sid examines the messages produced by the various sub-guards during the reconstruction phase and accordingly the $\mathcal{W}$ and $\mathcal{B}$ lists of the parties are updated. We stress that the parties keep executing SAVSS-MM with id sid, even after terminating the $(\mathsf{Sh}, \mathsf{Rec})$ protocols with id sid. This ensures that if some message is pending from a party in instance sid, then its communication is ignored by the SAVSS-MM protocol in any future instance $\mathsf{sid}' > \mathsf{sid}$. This will be later useful in our WSCC protocol to ensure that at least $\frac{t}{2}+1$ corrupt parties are globally shunned, if any Rec instance fails to terminate in WSCC.

Before we prove the properties of our SAVSS protocol, we prove the properties of the SAVSS memory management protocol, which will be used while proving the properties of SAVSS. The first property is that an honest party is never included in the $\mathcal{B}$ list of any honest party. The second property states that all pending messages from any honest party is eventually removed from the $\mathcal{W}$ list of every honest party.

Lemma 3.1 (Properties of SAVSS Memory Management Protocol). *The following holds for every honest $P_i \in \mathcal{P}$ during the SAVSS memory management protocol with id* sid *for any* $\mathsf{sid} \in \mathbb{N}$*:*

- *If P_j is included in $\mathcal{B}_i$ then P_j is corrupt.*
- *If P_j is honest, then any triplet of the form $(\star, P_j, \star)$ present in $\mathcal{W}_{(i,\mathsf{sid})}$ will eventually be removed.*

Proof. If P_j is honest, then it eventually broadcasts all the messages it is supposed to broadcast during the reconstruction phase of the SAVSS protocol with id sid and by the properties of broadcast, these messages are eventually received by every honest P_i. Hence every triplet of the form $(\star, P_j, \star)$ will eventually be removed from $\mathcal{W}_{(i,\mathsf{sid})}$. Moreover, an honest P_j correctly broadcasts the same values as received during the sharing phase. So the condition for including P_j to $\mathcal{B}_i$ is never satisfied and hence P_j is never included in $\mathcal{B}_i$. □

We next prove the termination property of the SAVSS protocol

Lemma 3.2 (Termination of SAVSS). *For any* $\mathsf{sid} \in \mathbb{N}$*, the following holds during the* $(\mathsf{Sh}, \mathsf{Rec})$ *protocol with id* sid*:*

1. *If* D *is honest and all honest parties participate in* Sh*, then each honest party eventually terminates* Sh*.*
2. *If some honest party terminates* Sh*, then every other honest party eventually terminates* Sh
3. *If all honest parties participate in* Rec*, then one of the following holds:*
   - *All honest parties eventually terminate* Rec *or*
   - *At least $\frac{t}{2}+1$ corrupt parties are included in $\mathcal{W}_{(i,\mathsf{sid})}$ of every honest party P_i*

Proof. We first note that during Sh and Rec, the messages of every honest party is cleared by the SAVSS-MM protocol and eventually delivered to the honest recipients. When D is honest, every honest P_i eventually broadcasts $(\texttt{ok}, P_j)$ for every honest P_j, with P_j eventually broadcasting `sent`. Thus, D eventually includes every honest P_j in the set $\mathcal{V}_i$ of every honest party P_i. D can follow the following steps to find the sets $(\mathcal{V}, \{\mathcal{V}_i\}_{P_i \in \mathcal{V}})$:

Figure 1: SAVSS with $n = 3t + 1$ and id sid.

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

Distribution by D – Dealer D selects a random, t-degree symmetric bivariate polynomial $F(x, y)$ with $F(0, 0) = s$ and sends $f_i(x) = F(x, i)$ to each $P_i \in \mathcal{P}$.

Pair-wise consistency check – Each party $P_i \in \mathcal{P}$ (including D), executes the following code:

- Wait until a t-degree polynomial $\widehat{f}_i(x)$ is received from D.
- For $j = 1, \ldots, n$, send $\widehat{f}_i(j)$ to party P_j. Then broadcast the message `sent`.
- If a value $\widehat{f}_{ji}$ is received from P_j and the message `sent` is received from the broadcast of P_j, such that $\widehat{f}_{ji} = \widehat{f}_i(j)$ holds, then broadcast the message $(\mathtt{ok}, P_j)$.

Construction of $\mathcal{V}$ – The following code is executed only by D:

- Find sets $\mathcal{V} \subseteq \mathcal{P}$ and $\mathcal{V}_i \subseteq \mathcal{P}$, for every $P_i \in \mathcal{V}$, such that: $|\mathcal{V}| \geq n-t$, $|\mathcal{V} \cap \mathcal{V}_i| \geq n-t$ and for every $P_i \in \mathcal{V}$ and $P_j \in \mathcal{V}_i$, $(\mathtt{ok}, P_j)$ and `sent` is received from the broadcast of P_i and P_j respectively.
- Redefine $\mathcal{V}$ as $\mathcal{V} = \mathcal{V} \cap (\bigcup_{P_j \in \mathcal{V}} \mathcal{V}_j)$ and $\mathcal{V}_i = \mathcal{V} \cap \mathcal{V}_i$. Broadcast the redefined $\mathcal{V}$ and $\mathcal{V}_i$ sets.

Verifying $\mathcal{V}$ and populating $\mathcal{W}$ sets – Each $P_i \in \mathcal{P}$ (including D) executes the following code:

- Wait to receive $\mathcal{V}$ and $\mathcal{V}_j$ corresponding to each $P_j \in \mathcal{V}$ from the broadcast of D. On receiving check if $\mathcal{V} = \bigcup_{P_j \in \mathcal{V}} \mathcal{V}_j$ and $|\mathcal{V} \cap \mathcal{V}_j| \geq n - t$ holds for every $P_j \in \mathcal{V}$. *Accept* $\mathcal{V}$ if for each $P_j \in \mathcal{V}$ and each $P_k \in \mathcal{V}_j$, the message $(\mathtt{ok}, P_k)$ and `sent` is received from the broadcast of P_j and P_k respectively.
- Wait until $\mathcal{V}$ is accepted. Then populate $\mathcal{W}_{(i,\mathsf{sid})}$, initialized to $\emptyset$ as follows and terminate:
  - If $P_i = \mathsf{D}$, then add $(P_j, P_k, f_j(k))$ to $\mathcal{W}_{(i,\mathsf{sid})}$ for each $P_j \in \mathcal{V}$ and each $P_k \in \mathcal{V}_j$. This is interpreted as D expects P_k to reveal $f_j(k)$ as a sub-guard for the guard P_j during Rec.
  - If $P_i \in \mathcal{V}$, then add the following triplets to $\mathcal{W}_{(i,\mathsf{sid})}$:
    - $(P_i, P_j, \widehat{f}_i(j))$ for each $P_j \in \mathcal{V}_i$. This is interpreted as P_i (as a guard), expects P_j to reveal $\widehat{f}_i(j)$ (as a sub-guard) during Rec.
    - $(P_i, P_j, \widehat{f}_i(j))$ for all P_j such that $P_j \notin \mathcal{V}_i$, $P_j \in \mathcal{V}$ and $P_i \in \mathcal{V}_j$. This is interpreted as P_i (as a sub-guard), expects P_j to reveal $\widehat{f}_i(j)$ (as a sub-guard of some guard) during Rec.
  - Else add the triplet $(P_j, P_k, \star)$ to $\mathcal{W}_{(i,\mathsf{sid})}$ for each $P_j \in \mathcal{V}$ and each $P_k \in \mathcal{V}_j$. This is interpreted as P_i expects P_k to reveal some value from $\mathbb{F}$ as a sub-guard for the guard P_j during Rec.

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

Making the polynomials public — Each $P_i \in \mathcal{V}$ broadcasts the t-degree polynomial $\widehat{f}_i(x)$.

Reconstructing the polynomials of guards: The following code is executed by every party $P_i \in \mathcal{P}$:

- For each $P_j \in \mathcal{V}$, initialize $\mathcal{K}_j$ to $\emptyset$. Add $(k, \widehat{f}_k(j))$ to $\mathcal{K}_j$, if a t-degree polynomial $\widehat{f}_k(x)$ is received from the broadcast of $P_k \in \mathcal{V}_j$.
- Wait until $|\mathcal{K}_j| \geq n-t-\frac{t}{2} = \frac{3}{2}t+1$ for each $P_j \in \mathcal{V}$ and then execute $\mathsf{RS\text{-}Dec}(t, \frac{t}{4}, \mathcal{K}_j)$. If for some $P_j \in \mathcal{V}$, $\mathsf{RS\text{-}Dec}(t, \frac{t}{4}, \mathcal{K}_j)$ does not output a t-degree polynomial, then output $\perp$ and terminate.
- Else let $\bar{f}_j(x)$ denote the t-degree polynomial obtained by $\mathsf{RS\text{-}Dec}(t, \frac{t}{4}, \mathcal{K}_j)$, for each $P_j \in \mathcal{V}$. If there exists a t-degree symmetric bivariate polynomial $\overline{F}(x, y)$, such that $\overline{F}(x, j) = \bar{f}_j(x)$ holds for each $P_j \in \mathcal{V}$, then output $\bar{s} \stackrel{def}{=} \overline{F}(0, 0)$ and terminate. Else output $\perp$ and terminate.

Figure 2: Protocol SAVSS-MM(D): SAVSS memory management protocol with id sid.

The following code is executed by each $P_i \in \mathcal{P}$:
Initialization: Initialize $\mathcal{W}_{(i,\mathsf{sid})}$ and $\mathcal{B}_i$ to $\emptyset$. The set $\mathcal{B}_i$ is a set that is initialized by the party P_i only once (when sid = 0) and dynamically updated during the instances of SAVSS-MM. The set $\mathcal{W}_{(i,\mathsf{sid})}$ is initialized in and maintained for SAVSS instance with id sid only.
Blocking messages: If $P_j \in \mathcal{B}_i$, discard any message received from P_j during (Sh, Rec) with id sid.
Filtering messages: If $P_j \notin \mathcal{B}_i$, then any message received from P_j during the SAVSS instance with id sid is processed as follows:

1. If a t-degree polynomial $\widehat{f}_k(x)$ is received from the broadcast of $P_k \in \mathcal{V}$ during Rec with id sid such that $(P_j, P_k, val) \in \mathcal{W}_{(i,\mathsf{sid})}$. Then do the following:
   - If $val = \star$, remove (P_j, P_k, x) from $\mathcal{W}_{(i,\mathsf{sid})}$ and forward $\widehat{f}_k(x)$ to the SAVSS instance sid;
   - If $val \neq \star$ and $\widehat{f}_k(j) = val$, then do the same as above;
   - If $val \neq \star$ and $\widehat{f}_k(j) \neq val$, add P_k to $\mathcal{B}_i$.
2. Else forward the received message to the SAVSS instance sid.

– **Initialisation:** Initialise a dynamic set $\mathcal{T}$ to $\emptyset$. For every party P_i initialise a dynamic set $\mathcal{V}_i$ to $\emptyset$.
– **Update of $\mathcal{V}$ Sets:** Include every P_j in $\mathcal{V}_i$ for which `sent` is received from the broadcast of P_j and $(\mathtt{ok}, P_j)$ is received from the broadcast of P_i.
– **Update of $\mathcal{T}$:** Include a party P_i in $\mathcal{T}$ if $|\mathcal{V}_i| \geq n - t$ holds.
– **Finding Candidate Solution:** If there exists a subset $\mathcal{V}$ of $\mathcal{T}$ such that $|\mathcal{V} \cap \mathcal{V}_i| \geq n - t$ is true for every $P_i \in \mathcal{V}$, then return $(\mathcal{V}, \{\mathcal{V}_i\}_{P_i \in \mathcal{V}})$ and stop. Else wait and verify if the above condition is true after every update of $\mathcal{T}$ or $\mathcal{V}_i$s.

As there are at most t corrupt parties, $|\mathcal{V}_i| \geq n - t$ eventually holds for every honest P_i and so every honest P_i is eventually included in the set $\mathcal{T}$. Hence, D eventually finds a $\mathcal{V} \subseteq \mathcal{T}$, such that $|\mathcal{V} \cap \mathcal{V}_i| \geq n - t$ holds for every $P_i \in \mathcal{V}$. As a result, D eventually finds a $\mathcal{V}$ and then re-defines this set by restricting it to only parties in $\Big(\bigcup_{P_j \in \mathcal{V}} \mathcal{V}_j\Big)$. Further, D re-defines the existing $\mathcal{V}_i$ sets by restricting them within the re-defined $\mathcal{V}$. We next claim that the re-defined $\mathcal{V}_i$ sets will still have an overlap of size at least $n - t$ with the re-defined set $\mathcal{V}$. We define $\mathcal{V}^{\mathsf{Old}}$ and $\mathcal{V}^{\mathsf{New}}$ to represent the contents of $\mathcal{V}$ before and after re-defining the $\mathcal{V}$ set respectively. Similarly, let $\mathcal{V}_i^{\mathsf{Old}}$ and $\mathcal{V}_i^{\mathsf{New}}$ represent the contents of $\mathcal{V}_i$ before and after re-defining the set $\mathcal{V}_i$ respectively. We now have that $\mathcal{V}^{\mathsf{New}} = \mathcal{V}^{\mathsf{Old}} \cap \Big(\bigcup_{P_j \in \mathcal{V}^{\mathsf{Old}}} \mathcal{V}_j^{\mathsf{Old}}\Big)$ and $\mathcal{V}_i^{\mathsf{New}} = \mathcal{V}^{\mathsf{New}} \cap \mathcal{V}_i^{\mathsf{Old}}$. By substituting for $\mathcal{V}^{\mathsf{New}}$ and $\mathcal{V}_i^{\mathsf{New}}$, we get that $\big(\mathcal{V}^{\mathsf{New}} \cap \mathcal{V}_i^{\mathsf{New}}\big) = \mathcal{V}^{\mathsf{New}} \cap \big(\mathcal{V}^{\mathsf{New}} \cap \mathcal{V}_i^{\mathsf{Old}}\big) = \mathcal{V}^{\mathsf{Old}} \cap \Big(\bigcup_{P_j \in \mathcal{V}^{\mathsf{Old}}} \mathcal{V}_j^{\mathsf{Old}}\Big) \cap \mathcal{V}_i^{\mathsf{Old}} = \mathcal{V}^{\mathsf{Old}} \cap \mathcal{V}_i^{\mathsf{Old}}$; this is because $P_i \in \mathcal{V}^{\mathsf{Old}}$ and hence $\mathcal{V}_i^{\mathsf{Old}} \subseteq \Big(\bigcup_{P_j \in \mathcal{V}^{\mathsf{Old}}} \mathcal{V}_j^{\mathsf{Old}}\Big)$. This proves the claim that $|\mathcal{V}^{\mathsf{New}} \cap \mathcal{V}_i^{\mathsf{New}}| \geq n - t$ as it follows from the fact that $|\mathcal{V}^{\mathsf{Old}} \cap \mathcal{V}_i^{\mathsf{Old}}| \geq n - t$ holds. From now on, we simply refer to these re-defined sets $\mathcal{V}^{\mathsf{New}}$ and $\mathcal{V}_i^{\mathsf{New}}$ as $\mathcal{V}$ and $\mathcal{V}_i$ respectively as only these sets are needed for the rest of the proof. So, D eventually broadcasts $\mathcal{V}$ and $\mathcal{V}_i$ sets corresponding to every $P_i \in \mathcal{V}$. By the properties of broadcast, every honest party eventually receives these sets from the broadcast of D, who is honest, and accepts the same after verifying them. So, every honest party eventually terminates Sh. This proves the first part.

Let P_h be the first honest party who terminates Sh. This implies that P_h receives and accepts the set $\mathcal{V}$ of size at least $n - t$ and the $\mathcal{V}_i$ sets corresponding to every $P_i \in \mathcal{V}$ from the broadcast of D, such that $|\mathcal{V} \cap \mathcal{V}_i| \geq n - t$ holds for every party $P_i \in \mathcal{V}$. By the properties of broadcast, every other honest party

eventually receives these sets. It follows easily that every other honest party also eventually accepts the set $\mathcal{V}$ and terminates Sh, proving the second requirement.

For the third part, we first note that honest parties participate in Rec, only after terminating Sh and accepting $\mathcal{V}$. Moreover, apart from accepting $\mathcal{V}$, each honest party P_i also populates $\mathcal{W}_{(i,\mathsf{sid})}$. That is, for every $P_j \in \mathcal{V}$ and every P_k in $\mathcal{V}_j$, there is a corresponding triplet of the form $(\star, P_k, \star)$ present in $\mathcal{W}_{(i,\mathsf{sid})}$. If a corrupt $P_k \in \mathcal{V}_j$ does not participate in the reconstruction protocol (by not broadcasting the values it is supposed to broadcast), then as per the protocol SAVSS-MM, the communication from P_k is marked as pending and not removed from $\mathcal{W}_{(i,\mathsf{sid})}$ forever. Now, if Rec does not terminate for all the honest parties, then it implies that there is some $P_j \in \mathcal{V}$, such that t-degree polynomial $\widehat{f}_k(x)$ is received from the broadcast of at most $n - t - \frac{t}{2} - 1 = \frac{3t}{2}$ parties $P_k \in \mathcal{V}_j$. This further implies that there are at least $\frac{t}{2} + 1$ corrupt parties $P_k \in \mathcal{V}_j$, who do not participate in the Rec protocol. It now follows easily that corresponding to such P_k, a triplet of the form $(\star, P_k, \star)$ will be present in $\mathcal{W}_{(i,\mathsf{sid})}$ permanently for every honest P_i . □

As a corollary of Lemma 3.2, we can state that if there are at most $\frac{t}{2} - 1$ corrupt parties participating during Rec, then protocol Rec eventually terminates for each honest party. This is because there can be at most $\frac{t}{2} - 1$ corrupt sub-guards in each $\mathcal{V}_j$, who do not participate during Rec. However, the values from the remaining $\frac{3t}{2} + 1$ honest sub-guards from $\mathcal{V}_j$ are eventually received and hence RS-Dec is eventually executed for each P_j in $\mathcal{V}$.

Corollary 3.3. *For any* sid $\in \mathbb{N}$*, if all the honest parties participate during the* Rec *protocol with id* sid *and if there are at most* $\frac{t}{2} - 1$ *corrupt parties participating during the* Rec *protocol, then the* Rec *protocol eventually terminates for each honest party.*

We next prove the correctness of SAVSS.

Lemma 3.4 (Correctness of SAVSS). *For any* sid $\in \mathbb{N}$*, the following holds during the* (Sh, Rec) *protocol with id* sid*: If honest parties terminate* Rec*, then there exists a unique value* $\bar{s}$ *where* $\bar{s} = s$ *for an honest* D *and* $\bar{s} \in \mathbb{F} \cup \{\perp\}$ *for a corrupt* D*, such that one of the following holds: all honest parties output* $\bar{s}$ *at the end of* Rec *or at least* $\frac{t}{4} + 1$ *local conflicts occur.*

Proof. For the first part of the proof, we consider an *honest* dealer. Note that if D is *honest*, then for *every* party $P_j \in \mathcal{V}$, the polynomial $\widehat{f}_j(x)$ received by P_j, is the same as t-degree polynomial $f_j(x)$, where $f_j(x) = F(x, j)$ and $F(x, y)$ is the t-degree bivariate polynomial selected by D. To prove this lemma, we need to show that each *honest* party P_i either reconstructs $\bar{f}_j(x) = f_j(x)$ during Rec for *each* $P_j \in \mathcal{V}$ or $\frac{t}{4} + 1$ local conflicts occur. If, for each $P_j \in \mathcal{V}$, the honest P_i reconstructs $\bar{f}_j(x) = f_j(x)$ during Rec, then clearly $\overline{F}(x, y) = F(x, y)$ holds and hence $\bar{s} = s$ holds. On the other hand, let there exists some $P_j \in \mathcal{V}$, such that P_i reconstructs $\bar{f}_j(x) \neq f_j(x)$ during Rec. Note that $\bar{f}_j(x)$ is the output of executing RS-Dec $\big($with $N = \frac{3t}{2} + 1$ and $c = \frac{t}{4}$ $\big)$ on the set $\mathcal{K}_j$ and if there are at most $\frac{t}{4}$ incorrect values in the set $\mathcal{K}_j$ constructed by P_i, then $\bar{f}_j(x) = f_j(x)$ holds. So, for $\bar{f}_j(x) \neq f_j(x)$ to be true, there should be at least $\frac{t}{4} + 1$ incorrect values $(k, \widehat{f}_k(j))$ in $\mathcal{K}_j$, such that $\widehat{f}_k(j) \neq f_j(k)$ holds. This implies that at least $\frac{t}{4} + 1$ *corrupt* parties $P_k \in \mathcal{V}_j$ broadcast t-degree $\widehat{f}_k(x)$ during Rec, such that $\widehat{f}_k(x) \neq f_k(x)$. We note that if $P_k \in \mathcal{V}_j$ then it implies that $P_k \in \mathcal{V}$ as well because the sub-guards in each sub-guard list is restricted within $\mathcal{V}$. This implies that there exists a set $\mathcal{V}_k$ as well. We claim that if a t-degree polynomial $\widehat{f}_k(x)$ is received from the broadcast of P_k during Rec, such that $\widehat{f}_k(x) \neq f_k(x)$, then at least one honest party $P_l \in \mathcal{V}_k$ will be in local conflict with P_k. It easily follows from this claim that at least $\frac{t}{4} + 1$ local conflicts occur since there are at least $\frac{t}{4} + 1$ corrupt parties P_k, which reveal $\widehat{f}_k(x) \neq f_k(x)$.

To prove our claim, we note that for *every* party $P_k \in \mathcal{V}$, the set $\mathcal{V}_k$ has at least $n - 2t = t + 1$ honest parties P_l from $\mathcal{V}$ because $|\mathcal{V} \cap \mathcal{V}_k| \geq n - t$ is ensured. Let $\mathcal{H}_k$ be the set of such honest parties P_l. For each party $P_l \in \mathcal{H}_k$, the condition $f_k(l) = f_l(k)$ holds. This is because P_l in included in $\mathcal{V}_k$ only after

P_k broadcasts the message $(\texttt{ok}, l)$, on receiving $f_{lk} = f_l(k)$ from P_l and locally verifying that $f_k(l) = f_l(k)$ holds. Moreover, the values f_{lk} corresponding to the parties $P_l \in \mathcal{H}_k$ uniquely define the t-degree polynomial $f_k(x)$ since $|\mathcal{H}_k| \geq t+1$. So, if a polynomial $\widehat{f}_k(x)$ is received from the broadcast of P_k during Rec, such that $\widehat{f}_k(x) \neq f_k(x)$, then there are at most t parties P_l from $\mathcal{H}_k$ for which $\widehat{f}_k(l) = f_{lk}$ holds. This is because two *different* t-degree polynomials $\widehat{f}_k(x), f_k(x)$ can have at most t common values. Hence there exists at least one party $P_l \in \mathcal{H}_k$ for which $\widehat{f}_k(l) \neq f_{lk}$ holds. So during SAVSS-MM, party P_l will be in local conflict with P_k and includes P_k to the set $\mathcal{B}_l$. This proves the claim and hence the lemma for the case of an honest dealer. We next consider the case of a *corrupt* dealer to complete the rest of the proof.

We start by defining $\bar{s}$. Let $P_h \in \mathcal{P}$ be the first honest party which terminates Sh. This implies that P_h accepts a $\mathcal{V}$, broadcasted by D, where $|\mathcal{V}| \geq n-t$. This further implies that there are at least $n-2t = t+1$ honest parties in $\mathcal{V}$. Let $\mathcal{H}$ denote the set of honest parties in $\mathcal{V}$ and let $\widehat{f}_j(x)$ denote the t-degree polynomial received by each $P_j \in \mathcal{H}$ from D. If there exists a t-degree symmetric bivariate polynomial, say $\overline{F}(x,y)$, such that $\overline{F}(x,j) = \widehat{f}_j(x)$ holds for each $P_j \in \mathcal{H}$, then we set $\bar{s} = \overline{F}(0,0)$, otherwise we set $\bar{s} = \perp$. Now there are two possible cases.

If $s \neq \perp$, then the proof of the lemma is exactly the same as in the case of an honest dealer. This is because in this case, the t-degree univariate polynomials of *all* the parties in $\mathcal{V}$ (including the ones who are outside $\mathcal{H}$) lie[2] on $\overline{F}(x,y)$, as for each P_j in $\mathcal{V} \setminus \mathcal{H}$, there are at least $t+1$ parties P_k from $\mathcal{H}$ in $\mathcal{V}_j$, with whom the polynomial of P_j is pair-wise consistent. That is, $\widehat{f}_j(k) = \widehat{f}_k(j)$ holds. And the $\widehat{f}_k(j)$ values of these honest parties uniquely define a t-degree polynomial $\overline{F}(x,j)$. Now, consider the second case when $\bar{s} = \perp$. If any honest party, upon terminating Rec, outputs $\bar{\bar{s}} \neq \bar{s}$, then it implies that there exists at least one party from $\mathcal{H}$, say P_j, such that $\widehat{f}_j(x)$ is not reconstructed correctly during Rec; i.e. $\bar{f}_j(x) \neq \widehat{f}_j(x)$ holds. Using a similar reasoning as in the case of an honest dealer, we can conclude that in this case, at least $\frac{t}{4}+1$ local conflicts occur. □

We next prove the privacy of SAVSS.

Lemma 3.5 (Privacy of SAVSS). *For any* sid $\in \mathbb{N}$, *if* D *is honest, then the view of the adversary is independent of the input s of* D *during the* Sh *protocol with id* sid.

Proof. Let $\mathcal{C}$ be the set of parties under the control of A, where $|\mathcal{C}| \leq t$ and D $\notin \mathcal{C}$. So A will know the polynomials $f_i(x)$, where $P_i \in \mathcal{C}$. We first claim that throughout the protocol Sh, the adversary obtains no additional information other than these polynomials. During the instance Sh, adversary A obtains at most t shares of the t-degree polynomial $f_j(x)$ for an honest party P_j. These t shares are already known to A, as these can be computed from the row polynomials of the t parties in $\mathcal{C}$. Hence no new information about $f_j(x)$ is revealed to A. We now show that given only the polynomials of the corrupted parties in $\mathcal{C}$, no information about the secret $s = F(0,0)$ is revealed to A. The proof follows from the properties of t-degree symmetric bivariate polynomials, as given in [11]; for the sake of completeness, we recall the proof in the sequel.

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

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

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

The polynomial $h(x)$ has degree at most t, where $h(0) = 1$ and $h(i) = 0$, for every $P_i \in \mathcal{C}$. Now, define the bivariate polynomial $Z(x,y)$ as follows:

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

[2] We say a t-degree univariate polynomial $f_j(x)$ lie on a t-degree bivariate polynomial $F(x,y)$, if $F(x,j) = f_j(x)$ holds.

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

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

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

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

and

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

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

Next we prove the communication complexity of the SAVSS.

Lemma 3.6 (Communication Complexity of SAVSS). *For any* sid $\in \mathbb{N}$*, the following holds during the* (Sh, Rec) *protocol with id* sid*: the* Sh *protocol has communication complexity of* $\mathcal{O}(n^4 \log |\mathbb{F}|)$ *bits. The* Rec *protocol has communication complexity of* $\mathcal{O}(n^4 \log |\mathbb{F}|)$ *bits.*

Proof. During Sh, dealer D distributes $\mathcal{O}(n^2)$ values from $\mathbb{F}$ and parties also exchange $\mathcal{O}(n^2)$ values from $\mathbb{F}$, which amounts to $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits of communication over the point to point channels. In addition, D broadcasts a $\mathcal{V}$ containing the identity of $\mathcal{O}(n)$ parties and $\mathcal{O}(n)$ $\mathcal{V}_i$ sets, each containing the identity of $\mathcal{O}(n)$ parties, where the identity of each part can be represented by $\log n$ bits. This amounts to broadcast of $\mathcal{O}(n^2 \log n)$ bits, which translates to a communication of $\mathcal{O}(n^4 \log n)$ bits (since broadcast of one bit requires $\mathcal{O}(n^2)$ bits of communication). During Rec, each party in $\mathcal{V}$ broadcasts its t-degree polynomial, which requires broadcasting $\mathcal{O}(n)$ values from $\mathbb{F}$. Hence Rec protocol has a broadcast of $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits, which requires a communication of $\mathcal{O}(n^4 \log |\mathbb{F}|)$ bits. □

The following theorem follows from Lemma 3.2-3.6.

Theorem 3.7. *For every* sid $\in \mathbb{N}$*, the pair of protocols* (Sh, Rec) *with id* sid *is an SAVSS scheme. The* Sh *and* Rec *protocols have a communication complexity of* $\mathcal{O}(n^4 \log |\mathbb{F}|)$ *bits each.*

4 Weak Shunning Common Coin (WSCC) Protocol with $n = 3t + 1$

Similar to the SCC of [1], our WSCC protocol WSCC has two main stages. In the first stage, each party attaches itself with a uniform random value that is unknown to all including itself and simultaneously ensures that a "sufficiently large" number of parties do the same. In the second stage, the secret values are revealed and a bit is output taking these opened values into account. The first stage is implemented by making each party act as a dealer to run n instances of Sh and share n random values from $\mathbb{F}$, one for each party. Each party P_i *attaches* itself, with the secrets meant for it, to a set of at least $t + 1$ dealers C_i whose Sh instances it terminates. Then each P_i decides whether there exists a sufficiently large set of parties H_i, such that the set C_j has been identified for each P_j in H_i. Once this is done, the next stage is to reconstruct the values attached to the parties in H_i, based on which each party in H_i is *associated* with a uniformly random value. The transition from stage one to stage two is locally marked by setting a local Boolean variable $\texttt{Flag}_i$ by each P_i. Party P_i then decides its outcome based on the values associated to the parties in H_i.

The SCC protocol of [1] ensures that either all honest parties output the same coin with probability at least $\frac{1}{4}$ or one of the Rec instances fail and *at least one* local conflict occurs. However, the SCC protocol *always* terminates, as their underlying Rec instances *always* terminate. We depart from the protocol of [1] in several ways. First, when Rec instance fails and the correctness of common coin is disturbed, at least $\frac{t}{4}+1$ local conflicts, instead of just one, are guaranteed to occur, allowing a faster convergence to a correct instance. This directly carries over from the correctness of the underlying SAVSS protocol. Second, the parties may not be able to compute their outcome always, owing to the fact that some underlying Rec instance may not terminate, in contrast to guaranteed termination of the SCC of [1]. In this case, we ensure that at least $\frac{t}{2}+1$ corrupt parties are shunned by *all* honest parties. This guarantee ensures that in any subsequent invocation of WSCC, there can be at most $\frac{t}{2}-1$ corrupt parties participating, who can never prevent the honest parties from computing their outcome. Ensuring the above, however, needs an additional step in the existing SCC protocol and a clever memory management protocol WSCCMM. Specifically, we do it in two steps. First, by using a "watch-list mechanism" and applying a stricter condition for attaching a party's shared value, we ensure that at least $t+1$ honest parties shun at least $\frac{t}{2}+1$ corrupt parties when some Rec instance contributing its secret to some honest party's output does not terminate. Next, our WSCCMM ensures that all honest parties shun those $\frac{t}{2}+1$ corrupt parties from participating in any subsequent instance of WSCC. The details follow.

Let Sh_{jk} denote the Sh instance with P_j as a dealer to share a secret for P_k and let Rec_{jk} denote the corresponding Rec instance. Each party P_i maintains a local dynamic watch-list $\mathcal{T}_i$, to keep track of instances Sh_{jk} that terminate for P_i. The goal to is to keep their corresponding Rec instances under surveillance so that P_i shuns at least $\frac{t}{2}+1$ corrupt parties in case Rec instance of some Sh instance from $\mathcal{T}_i$ does not terminate. Now to ensure that not just P_i alone but at least a set of $t+1$ honest parties shun the same set of corrupt parties, the condition for inclusion in C_i and attaching a value is made stricter. Namely, P_i adds P_j to C_i if at least $n-t$ parties confirm that Sh_{jk} for all $k=1,\ldots,n$ terminates for them and is present in their respective $\mathcal{T}$ sets. Now to ensure that *every* honest party in $\mathcal{P}$ (and not just $t+1$ honest parties) shuns these corrupt parties, we do the following. As part of the memory management protocol WSCCMM, a party P_i "approves" a party P_j *locally* by broadcasting `OK` signal only when P_j clears all pending messages as a part of all the Rec instances corresponding to $\mathcal{T}_i$ which is *freezed* at the time of setting $\mathtt{Flag}_i$. Finally, P_i approves P_j *globally* and allows it to participate in any subsequent WSCC instance, if at least $n-t$ parties broadcast `OK` message for P_j during WSCCMM. As a consequence of these steps, any instance Rec_{jk} that contributes to the attached value of some honest party is watched by at least $t+1$ honest parties. These parties will shun and never broadcast `OK` signal for the shunned $\frac{t}{2}+1$ corrupt parties when the Rec_{jk} instance does not terminate. So these corrupt parties are never approved globally by any of the honest parties. Consequently, these corrupt parties are shunned for all subsequent WSCC invocations.

Note that due to asynchrony, the sets H_i and H_j might be different for P_i and P_j. To ensure that *all* honest parties participate in the Rec instances, required to reconstruct the attached values of the parties in H_i and H_j, every honest party keeps participating in WSCC, even after obtaining its output. This is crucial as there is no guarantee of termination of a Rec instance, if only a strict subset of honest parties participate in the Rec instance. Hence, there is no termination condition in WSCC. We next present WSCC, WSCCMM and proof of the properties of these protocols. We note that our protocol has diffenent probabilities of outputting 0 and 1. Yet, our SCC that uses WSCC as a subprotocol outputs both possible bits with the same probability of $\frac{1}{4}$, a property achieved traditionally by the existing common coins.

Protocol WSCC and WSCCMM are formally presented in Fig 3 and Fig 4 respectively. Each instance of WSCC is identified by a unique id of the form $(\mathsf{sid},\mathsf{r})$, where $\mathsf{sid}\in\mathbb{N}$ and $\mathsf{r}\in\{1,2,3\}$. Looking ahead, in our SCC protocol SCC with id sid, three parallel instances of WSCC are executed and we will associate the ids $(\mathsf{sid},1)$, $(\mathsf{sid},2)$ and $(\mathsf{sid},3)$ to these WSCC instances. Inside the WSCC protocol, each $P_i\in\mathcal{P}$ acts as a D and invokes an instance of Sh on the behalf of every party $P_j\in\mathcal{P}$. These Sh (and the corresponding Rec and SAVSS-MM) instances will be associated with id of the form[3] $(\mathsf{sid},\mathsf{r},P_i,P_j)$; accordingly, the corresponding

[3]Recall that during the description of Sh, Rec and SAVSS-MM, we used ids of the form sid; the additional components r, P_i and

$\mathcal{W}$ sets for each P_k will be of the form $\mathcal{W}_{(k,\mathsf{sid},\mathsf{r},P_i,P_j)}$. Also, we add additional technicalities in WSCC to ensure proper working of the watch-list mechanism discussed earlier. Specifically, once P_i sets $\texttt{Flag}_i$ to one, it stops including new Sh instances to $\mathcal{T}_i$, even if those Sh instances terminate for P_i; we stress that P_i keeps participating in all the Sh instances which has not yet terminated for P_i, even if P_i sets $\texttt{Flag}_i$ to one. Looking ahead, this will be very crucial for the termination of our SCC, where WSCC is used as a sub-protocol. An instance of WSCCMM accompany each instance of WSCC, according to which each party locally decides whether to process a received message as per the WSCC protocol, delay it temporarily or block it permanently.

Figure 3: WSCC protocol with $n = 3t + 1$ and id $(\mathsf{sid}, \mathsf{r})$.

WSCC$(\mathsf{sid}, \mathsf{r})$

Each $P_i \in \mathcal{P}$ executes the following code:

1. For $1 \leq j \leq n$, choose a random secret $s_{ij} \in_R \mathbb{F}$ on the behalf of P_j and as a D, invoke an instance $\mathsf{Sh}(P_i, \mathcal{P}, s_{ij})$ of Sh with id $(\mathsf{sid}, \mathsf{r}, P_i, P_j)$. Denote this invocation by Sh_{ij}. Participate in the invocations Sh_{jk} for every $P_j, P_k \in \mathcal{P}$.
2. Broadcast $(\texttt{Completed}, (\mathsf{sid}, \mathsf{r}, P_j, P_k))$ and add $(\mathsf{sid}, \mathsf{r}, P_j, P_k)$ to $\mathcal{T}_{(i,\mathsf{sid},\mathsf{r})}$, initialized to $\emptyset$, on terminating Sh_{jk}.
3. Initialize a set $\mathcal{C}_i$ to $\emptyset$. Add party P_j to $\mathcal{C}_i$, if the following two conditions are satisfied for all $1 \leq k \leq n$:
   - Sh_{jk} has terminated and
   - The message $(\texttt{Completed}, (\mathsf{sid}, \mathsf{r}, P_j, P_k))$ is received from the broadcast of $n - t$ parties.

   Wait until $|\mathcal{C}_i| \geq t + 1$. Then, assign $C_i = \mathcal{C}_i$ and broadcast the message $(\texttt{Attach}, C_i, P_i)$. We say that that the values $\{s_{ji} | P_j \in C_i\}$ represent the *secrets attached to party* P_i.
4. *Accept* P_j and include it in the set $\mathcal{G}_i$, if the message $(\texttt{Attach}, C_j, P_j)$ is received from the broadcast of P_j and $C_j \subseteq \mathcal{C}_i$ holds. Wait until $|\mathcal{G}_i| \geq n - t$. Then, let $G_i = \mathcal{G}_i$ and broadcast the message $(\texttt{Ready}, P_i, G_i)$.
5. Consider P_j to be *supportive* and include it in the set $\mathcal{S}_i$, if P_i receives the message $(\texttt{Ready}, P_j, G_j)$ from the broadcast of P_j and each party in G_j is accepted (i.e. $G_j \subseteq \mathcal{G}_i$). Wait until $|\mathcal{S}_i| \geq n - t$. Then, set $\texttt{Flag}_i = 1$ (initialized to 0). Let S_i and H_i denote the contents of $\mathcal{S}_i$ and $\mathcal{G}_i$ respectively, when $\texttt{Flag}_i$ becomes 1.
6. Wait until $\texttt{Flag}_i = 1$. Then reconstruct the secrets attached to all the accepted parties. That is, for each $P_j \in C_k$ such that $P_k \in \mathcal{G}_i$, start participating in the instances $\mathsf{Rec}(P_j, \mathcal{P}, s_{jk})$. Denote this instance of Rec as Rec_{jk} and let r_{jk} be the corresponding output. (Some parties may be included in $\mathcal{G}_i$ after $\texttt{Flag}_i$ is set to 1. In that case, immediately start participating in the corresponding Rec instances.) In addition, stop executing step 2. That is, only keep participating in any Sh instance which has not yet terminated, after setting $\texttt{Flag}_i$ to one. But upon terminating these Sh instances, do not include these Sh instances to $\mathcal{T}_i$ and do not broadcast any Completed message
7. Let $u \stackrel{def}{=} \lceil 2.22n \rceil$. For every $P_k \in \mathcal{G}_i$, define v_k, the *value associated with* P_k, to be the sum modulo u of all the secrets attached to P_k. That is, $v_k \stackrel{def}{=} \left(\sum_{P_j \in C_k} r_{jk} \right) mod\ u$.
8. Wait until the values associated will all the parties in H_i are computed. If there exists a party $P_k \in H_i$ where $v_k = 0$, then output 0. Else, output 1. [a]

[a] Party P_i keeps executing the protocol even if it obtains its output. Note that the step for computing the output will be executed at most once, as the set H_i gets fixed, once $\texttt{Flag}_i$ is set to 1.

P_j are brought out here to distinguish the various instances of Sh, Rec and SAVSS-MM invoked inside WSCC.

Figure 4: Memory management protocol for WSCC with id $(\mathsf{sid}, \mathsf{r})$.

WSCCMM(sid, r)

Each $P_i \in \mathcal{P}$ executes the following code:

Initialization: Initialize the set $\mathcal{T}_{(i,\mathsf{sid},\mathsf{r})}$ to $\emptyset$ and the set $\mathcal{A}_{(i,\mathsf{sid},\mathsf{r})}$ to $\emptyset$ at the beginning of WSCC instance with id $(\mathsf{sid}, \mathsf{r})$.

Permanently blocking: If $P_j \in \mathcal{B}_i$, then all the messages received from P_j during the WSCC instance with id $(\mathsf{sid}, \mathsf{r})$ are discarded.

Filtering messages:

- If $\mathsf{r} = 1$, then forward every message received from P_j during the WSCC instance with id $(\mathsf{sid}, \mathsf{r})$ to the underlying WSCC protocol.
- If $\mathsf{r} > 1$, then any message received from P_j during the WSCC instance with id $(\mathsf{sid}, \mathsf{r})$ is processed as follows:
  - If $P_j \in \mathcal{A}_{(i,\mathsf{sid},\mathsf{r}')}$ for all $\mathsf{r}' < \mathsf{r}$, then forward the message to the WSCC instance with id $(\mathsf{sid}, \mathsf{r})$.
  - Else, delay it.

Identifying pending messages from the current WSCC instance:

- If $\mathtt{Flag}_i = 1$, then broadcast the message $(\mathtt{OK}, P_j)$ if the following conditions hold
  - $P_j \notin \mathcal{B}_i$ and
  - For every SAVSS instance $(\mathsf{sid}, \mathsf{r}, P_k, P_l) \in \mathcal{T}_{(i,\mathsf{sid},\mathsf{r})}$, there exists no triplet of the form $(\star, P_j, \star)$ in the set $\mathcal{W}_{(i,\mathsf{sid},\mathsf{r},P_k,P_l)}$.
- If $n - t$ parties broadcast $(\mathtt{OK}, P_j)$, then add P_j to $\mathcal{A}_{(i,\mathsf{sid},\mathsf{r})}$.

Before we prove the properties of WSCC, we prove two important properties of WSCCMM (Lemma 4.2), which will be used while proving the properties of WSCC. The first property states that each *honest* party is eventually included in the $\mathcal{A}$ set of every *honest* party. The second property states that if some honest party fails to terminate any Rec instance while reconstructing the secrets attached to an accepted party, then at least $\frac{t}{2} + 1$ corrupt parties are disapproved by *all* the honest parties. To prove these properties of the memory management protocol, we first prove a lemma that will be useful; the lemma states that for every accepted party P_k in $\mathcal{G}_i$, at least $t + 1$ honest parties terminate the instance Sh_{jk}, corresponding to each $P_j \in C_k$.

Lemma 4.1. *For any* $(\mathsf{sid}, \mathsf{r})$ *with* $\mathsf{sid} \in \mathbb{N}$ *and* $\mathsf{r} \in \{1, 2, 3\}$ *, the following holds for any honest party* P_i *during the* WSCC *protocol with id* $(\mathsf{sid}, \mathsf{r})$*: if* $P_k \in \mathcal{G}_i$*, then for every* $P_j \in C_k$*, there are at least* $t + 1$ *honest parties* P_l *which include the* Sh *instance with id* $(\mathsf{sid}, \mathsf{r}, P_j, P_k)$ *to* $\mathcal{T}_{(l,\mathsf{sid},\mathsf{r})}$.

Proof. Let P_i be an arbitrary honest party. As $P_k \in \mathcal{G}_i$, it implies that $C_k \subseteq \mathcal{C}_i$ holds. This further implies that every P_j in C_k also belongs to $\mathcal{C}_i$. Since P_i adds party P_j to $\mathcal{C}_i$, it follows that P_i receives the message $(\mathtt{Completed}, (\mathsf{sid}, \mathsf{r}, P_j, P_k))$ from the broadcast of $n - t$ different parties. Out of these $n - t$ parties, at least $n - 2t = t + 1$ parties P_l are honest. These $t + 1$ honest parties broadcast the `Completed` message only after terminating the Sh instance with id $(\mathsf{sid}, \mathsf{r}, P_j, P_k)$ and hence the id $(\mathsf{sid}, \mathsf{r}, P_j, P_k)$ is included to the set $\mathcal{T}_{(l,\mathsf{sid},\mathsf{r})}$. □

Lemma 4.2 (Properties of WSCC Memory Management Protocol). *For every honest* P_i*, the following hold during the* WSCCMM *protocol with id* $(\mathsf{sid}, \mathsf{r})$ *for any* $(\mathsf{sid}, \mathsf{r})$*, where* $\mathsf{sid} \in \mathbb{N}$ *and* $\mathsf{r} \in \{1, 2, 3\}$*:*

- *Every honest party P_j is eventually included in the $\mathcal{A}_{(i,\mathsf{sid},\mathsf{r})}$ set of P_i.*
- *For every $P_k \in \mathcal{G}_i$ and every $P_j \in C_k$, if Rec_{jk} does not terminate for P_i, then at least $\frac{t}{2}+1$ corrupt parties are not included in the set $\mathcal{A}_{(l,\mathsf{sid},\mathsf{r})}$ of every honest party P_l.*

Proof. If P_j is an honest party, then it follows from lemma 3.1 that there will be no pending communication from P_j in any Rec instance. Hence every triplet of the form $(\star, P_j, \star)$ present in the set $\mathcal{W}_{(i,\mathsf{sid},\mathsf{r},\star,\star)}$, corresponding to SAVSS instance $(\mathsf{sid},\mathsf{r},\star,\star)$, is eventually removed from the set $\mathcal{W}_{(i,\mathsf{sid},\mathsf{r},\star,\star)}$ of every honest party P_i; this holds irrespective of whether $(\mathsf{sid},\mathsf{r},\star,\star) \in \mathcal{T}_{(i,\mathsf{sid},\mathsf{r})}$ of every honest party P_i. Moreover, $P_j \notin \mathcal{B}_i$ holds. So, the conditions for broadcasting the message $(\mathtt{OK}, P_j)$ eventually become true for each honest P_i and hence each honest P_i eventually broadcasts the message $(\mathtt{OK}, P_j)$. As there are at least $n-t$ honest parties P_i, it follows that at least $n-t$ parties eventually broadcast the $(\mathtt{OK}, P_j)$ message, which are eventually received by every honest party. Hence, P_j is eventually included in $\mathcal{A}_{(i,\mathsf{sid},\mathsf{r})}$ set of every honest party P_i. This proves the first part.

For the second part, we first note that if $P_k \in \mathcal{G}_i$ and $P_j \in C_k$, then it implies that the Sh instance $(\mathsf{sid},\mathsf{r},P_j,P_k)$ is included in the $\mathcal{T}$ set of at least $t+1$ honest parties (see Lemma 4.1). Let $\mathcal{H}$ denote the set of these $t+1$ honest parties. Now consider the Rec instance Rec_{jk}. It follows from Lemma 3.2, that if the instance Rec_{jk} does not terminate, then communication from at least $\frac{t}{2}+1$ corrupt parties are marked as pending by the parties in $\mathcal{H}$. Let $\mathcal{F}$ denote the set of these $\frac{t}{2}+1$ corrupt parties. This implies that for each $P_c \in \mathcal{F}$, there exists triplets of the $(\star, P_c, \star)$ in the $\mathcal{W}$ set $\mathcal{W}_{(h,\mathsf{sid},\mathsf{r},P_j,P_k)}$ of every party $P_h \in \mathcal{H}$, which are never removed. This implies that no party in $\mathcal{H}$ ever broadcasts the $(\mathtt{OK}, P_c)$ message during the WSCCMM protocol, which further implies that there are less than $n-t$ parties who broadcast $(\mathtt{OK}, P_c)$ message. Hence the parties in $\mathcal{F}$ are not included in the $\mathcal{A}_{(i,\mathsf{sid},\mathsf{r})}$ set of any honest party P_i. □

We next prove that in the WSCC protocol, every honest party eventually sets its `Flag` variable to 1.

Lemma 4.3. *For any $(\mathsf{sid},\mathsf{r})$ with $\mathsf{sid} \in \mathbb{N}$ and $\mathsf{r} \in \{1,2,3\}$, if all honest parties participate in the* WSCC *protocol with id $(\mathsf{sid},\mathsf{r})$, then every honest party P_i eventually sets $\mathtt{Flag}_i = 1$.*

Proof. We first note that during the WSCC protocol with id $(\mathsf{sid},\mathsf{r})$, the message of every honest party is eventually delivered to every other honest party by the WSCCMM protocol. While this is trivially true if $\mathsf{r}=1$, for $\mathsf{r}>1$, this follows from the fact that every honest party is eventually included in the $\mathcal{A}$ set of every other honest party in all the past WSCCMM instances with id $(\mathsf{sid},\mathsf{r}')$, where $\mathsf{r}'<\mathsf{r}$ (see Lemma 4.2, part one). This implies that every honest party participates in the SAVSS instance Sh_{ij}, corresponding to every pair of honest parties (P_i,P_j). Hence it follows from the termination property of SAVSS (Lemma 3.2, part one) that every honest party eventually terminate such Sh_{ij} instances. As there are at least $n-t$ honest parties, it follows that C_i eventually becomes $t+1$ for every honest P_i and P_i eventually broadcasts the message $(\mathtt{Attach}, C_i, P_i)$. Hence every honest party P_i eventually receives the message $(\mathtt{Attach}, C_j, P_j)$ from the broadcast of every honest P_j. Moreover, $C_j \subseteq \mathcal{C}_i$ eventually holds. This is because if $P_k \in C_j$, then P_j terminates the instance Sh_{kl} for $l=1,...,n$. And from the termination property of SAVSS (Lemma 3.2, part two), every honest P_i eventually terminates Sh_{kl} for $l=1,...,n$ as well. This implies that every honest party P_j is eventually accepted by every honest party P_i and included in the set $\mathcal{G}_i$. Thus the size of $\mathcal{G}_i$ eventually becomes $n-t$ and hence every honest P_i eventually broadcasts the message $(\mathtt{Ready}, P_i, G_i)$. Following the same argument, the support set $\mathcal{S}_i$ of every honest party P_i eventually becomes of size $n-t$ and hence every honest party P_i sets $\mathtt{Flag}_i = 1$. □

We next prove that in WSCC, either every honest party obtains its output or at least $\frac{t}{2}+1$ corrupt parties are shunned by all the honest parties.

Lemma 4.4 (Output Computation of WSCC). *For any $(\mathsf{sid},\mathsf{r})$ with $\mathsf{sid} \in \mathbb{N}$ and $\mathsf{r} \in \{1,2,3\}$, if all honest parties participate in* WSCC *with id $(\mathsf{sid},\mathsf{r})$, then the following holds:*

1. *All honest parties eventually compute their output; or*
2. *At least* $\frac{t}{2}+1$ *corrupt parties are shunned by all the honest parties and not included in the* $\mathcal{A}$ *set* $\mathcal{A}_{(i,\mathsf{sid},\mathsf{r})}$ *of any honest party* P_i.

Proof. From Lemma 4.3, it follows that each honest party P_i eventually sets $\texttt{Flag}_i = 1$. So if P_i is able to terminate all Rec_{jk} instances, corresponding to each $P_k \in \mathcal{G}_i$ and each $P_j \in C_k$, then P_i is able to compute its output, proving the first part. For the second part, let Rec_{jk} does not terminate for P_i, corresponding to some $P_k \in \mathcal{G}_i$ and some $P_j \in C_k$. As Rec_{jk} does not terminate, it follows from the property of WSCCMM (Lemma 4.2, part two) that at least $\frac{t}{2}+1$ corrupt parties are shunned by all the honest parties and not included in the $\mathcal{A}$ set $\mathcal{A}_{(i,\mathsf{sid},\mathsf{r})}$ of every honest party P_i. □

We next prove that if some honest party P_j obtains its output during WSCC, then for every other honest party P_i, it holds that $H_j \subseteq \mathcal{G}_i$ and $S_j \subseteq \mathcal{S}_i$. Looking ahead, this property will be crucial for the termination of our SCC, where WSCC is used as a sub-protocol.

Lemma 4.5. *If all honest parties participate in* WSCC *with id* $(\mathsf{sid},\mathsf{r})$*, then the following holds for every* $\mathsf{sid} \in \mathbb{N}$ *and* $\mathsf{r} \in \{1,2,3\}$*: If* P_j *obtains its output during the* WSCC *protocol with id* $(\mathsf{sid},\mathsf{r})$*, then* $H_j \subseteq \mathcal{G}_i$ *and* $S_j \subseteq \mathcal{S}_i$ *eventually holds for every honest party* P_i.

Proof. The proof follows from the fact that any Sh instance which terminates for P_j, eventually terminates for P_i (Lemma 3.2, part two). Moreover, any message which is received by P_j as part of some broadcast is eventually and identically received by P_i. Furthermore, every honest party keeps on participating in WSCC, even after obtaining its output. □

We next prove a few lemmas that are useful for proving the correctness of WSCC; these lemmas are slight modifications of the lemmas proved in [9] to prove the correctness of their CC protocol.

Lemma 4.6. *For any* $(\mathsf{sid},\mathsf{r})$ *with* $\mathsf{sid} \in \mathbb{N}$ *and* $\mathsf{r} \in \{1,2,3\}$*, the following holds during* $\mathsf{WSCC}(\mathsf{sid},\mathsf{r})$*: Let* $u \stackrel{def}{=} \lceil 2.22n \rceil$*. Then, once some honest party* P_i *receives the message* $(\texttt{Attach}, C_k, P_k)$ *from the broadcast of any party* P_k*, then a unique value* v_k *is fixed such that the following holds:*

- *If the honest parties are able to compute the associated value of* P_k*, then one of the following holds:*
  - *All honest parties associate* v_k *with* P_k*; or*
  - *At least* $\frac{t}{4}+1$ *local conflicts occur during the protocol.*
- *The value* v_k *is distributed uniformly over* $[0,\ldots,u-1]$ *and is independent of the values associated with the other parties.*

Proof. Let $\{\bar{s}_{jk}\}_{P_j \in C_k}$ denote the set of values committed by $P_j \in C_k$ (as a D), on the behalf of P_k, during the instance Sh_{jk}. If P_j is *honest* then $\bar{s}_{jk} = s_{jk}$ holds and hence $\bar{s}_{jk}$ belongs to $\mathbb{F}$. On the other hand, if P_j is *corrupt*, then $\bar{s}_{jk} \in \mathbb{F} \cup \{\perp\}$. We follow the convention that if $\bar{s}_{jk} = \perp$, then we replace it by a publicly known default value from $\mathbb{F}$. Let $\{r_{jk}\}_{P_j \in C_k}$ denote the values, which are reconstructed during the instance Rec_{jk}. We define $v_k \stackrel{def}{=} \left(\sum_{P_j \in C_k} \bar{s}_{jk}\right) \mod u$. If the set $\{r_{jk}\}_{P_j \in C_k} = \{\bar{s}_{jk}\}_{P_j \in C_k}$, then clearly all honest parties associate v_k with P_k. Otherwise, there exists some pair of instance $(\mathsf{Sh}_{jk}, \mathsf{Rec}_{jk})$, such that the value r_{jk} reconstructed during Rec_{jk} is different from the value $\bar{s}_{jk}$ shared during Sh_{jk}. It follows from the correctness property of SAVSS (Lemma 3.4) that $\frac{t}{4}+1$ local conflicts occur during Rec_{jk}.

For the second property, we note that the parties start executing the instances $\{\mathsf{Rec}_{jk}\}_{P_j \in C_k}$ only after receiving the broadcasted message $(\texttt{Attach}, C_k, P_k)$. This implies that the set C_k is fixed, before any instance

in $\{\mathsf{Rec}_{jk}\}_{P_j \in C_k}$ is invoked. The set C_k consists of at least one *honest* party P_j. The privacy property of Sh (Lemma 3.5) ensures that the view of the adversary during the instance Sh_{jk} is independent of the secret s_{jk} shared by P_j. Now since the secrets shared by honest parties during WSCC are mutually independent and uniformly selected from $\mathbb{F}$, it follows that v_k is uniformly and independently distributed over $[0, \ldots, u-1]$. $\square$

Lemma 4.7. *For any* $(\mathsf{sid}, \mathsf{r})$ *with* $\mathsf{sid} \in \mathbb{N}$ *and* $\mathsf{r} \in \{1,2,3\}$*, the following holds during* $\mathsf{WSCC}(\mathsf{sid}, \mathsf{r})$*: Once some honest party sets its* $\mathtt{Flag}$ *to* 1*, then there exists a set, say* $\mathcal{M}$*, such that:*

1. *For each* $P_j \in \mathcal{M}$*, some honest party receives the message* $(\mathtt{Attach}, C_j, P_j)$ *from the broadcast of* P_j*.*
2. *Whenever any honest party* P_i *sets its* $\mathtt{Flag}_i = 1$*, it holds that* $\mathcal{M} \subseteq H_i$*.*
3. $|\mathcal{M}| \geq \frac{n}{3}$*.*

Proof. Let P_k be the first *honest* party to set its $\mathtt{Flag}$ variable $\mathtt{Flag}_k$ to one. We set $\mathcal{M}$ to be the set of parties P_m, who belong to G_l set of at least $t+1$ parties P_l in the set S_k. We next show that this set $\mathcal{M}$ has all the properties as stated in the lemma.

It is easy to see that $\mathcal{M} \subseteq H_k$. Hence party P_k receives the message $(\mathtt{Attach}, C_j, P_j)$ from the broadcast of every $P_j \in \mathcal{M}$. Since P_k is assumed to be an honest party, the first part of the lemma is proved.

An *honest* P_i sets $\mathtt{Flag}_i$ to one only when $\mathcal{S}_i$ contains $2t+1$ parties. Now note that $P_m \in \mathcal{M}$ implies that P_m belongs to G_l set of at least $t+1$ parties P_l in S_k. This ensures that there is at least one such P_l who belongs to S_i, as well as S_k. Now $P_l \in S_i$ implies that P_i had ensured that $G_l \subseteq \mathcal{G}_i$. This implies that $P_m \in \mathcal{M}$ belongs to $\mathcal{G}_i$, before party P_i sets $\mathtt{Flag}_i$ to one. Since H_i is the instance of $\mathcal{G}_i$ at the time when P_i sets $\mathtt{Flag}_i$ to one, it is obvious that P_m belongs to H_i as well. This proves the second part of the lemma.

We now proceed to prove the third part of the lemma, for which we use a counting argument. Let $h = |H_k|$. We have $h \geq n - t$. Now consider an $h \times n$ table Λ (relative to party P_k), such that the $\Lambda_{l,i}$ entry of Λ is one if and only if the following holds: (a) P_k has received the message $(\mathtt{Ready}, P_l, G_l)$ from the broadcast of P_l and included P_l in the set $\mathcal{S}_k$ before setting $\mathtt{Flag}_k = 1$ and (b)$P_i \in G_l$. Then, as per the definition of $\mathcal{M}$, the set $\mathcal{M}$ is the set of parties P_m such that the mth column in Λ contains one at least at $t+1$ positions. Notice that each row of Λ contains one at $n-t$ positions. Thus Λ contains one at $h(n-t)$ positions.

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

$$qh + (n-q)t \geq h(n-t).$$

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

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

This shows that $|\mathcal{M}| = q \geq \frac{n}{3}$. $\square$

Lemma 4.8 (Correctness of WSCC). *Let* $u \stackrel{def}{=} \lceil 2.22n \rceil$. *If the honest parties are able to compute their output during the* WSCC *protocol with id* $(\mathsf{sid}, \mathsf{r})$, *then one of the following holds for any* $\mathsf{sid} \in \mathbb{N}$ *and* $\mathsf{r} \in \{1, 2, 3\}$:

1. *All honest parties output* $\sigma = 0$ *with probability at least* $p_0 = 0.139$. *And all honest parties output* $\sigma = 1$ *with probability at least* $p_1 = 0.63$; *otherwise*
2. *At least* $\frac{t}{4} + 1$ *local conflicts occur during the protocol.*

Proof. Let P_i be an arbitrary *honest* party, which is able to compute its output of WSCC. Recall that in WSCC, party P_i sets its output bit based on the values associated with the parties in H_i. Moreover, for every $P_k \in H_i$, party P_i receives the message $(\texttt{Attach}, C_k, P_k)$ from the broadcast of P_k. This further guarantees that a uniformly and independently distributed value $v_k \in [0, \ldots, u-1]$ is fixed, corresponding to P_k (Lemma 4.6). Now there are two possible cases:

1. **Case I : For every $P_k \in H_i$, party P_i correctly associates the corresponding v_k.** Here we show that party P_i outputs zero with probability at least 0.139 and one with probability at least 0.63. There are two possible sub-cases:
   - Let $\mathcal{M}$ be the set of parties, as discussed in Lemma 4.7. From the same lemma, it holds that $\mathcal{M} \subseteq H_i$. If $v_k = 0$ holds for some $P_k \in \mathcal{M}$, then party P_i outputs 0 in WSCC. The probability that for at least one party $P_k \in \mathcal{M}$, the associated value $v_k = 0$ is $1 - (1 - \frac{1}{u})^{|\mathcal{M}|}$. Since $u = \lceil 2.22n \rceil$ and $|\mathcal{M}| \geq \frac{n}{3}$ (Lemma 4.7, part three), we have $1-(1-\frac{1}{u})^{|\mathcal{M}|} \geq 1-e^{-0.15} \geq 0.139$. So, the probability that P_i outputs $\sigma = 0$ is at least 0.139.
   - If $v_k \neq 0$ for every party $P_k \in H_i$, then P_i outputs $\sigma = 1$. The probability of this event is at least $(1 - \frac{1}{u})^n \geq e^{-0.45} \geq 0.63$.
2. **Case II : For some $P_k \in H_i$, party P_i associates $v'_k \neq v_k$.** In this case, from the first part of the Lemma 4.6, it holds that at least $\frac{t}{4} + 1$ local conflicts occur, proving the second part of this lemma.

□

The following theorem follows from Lemma 4.4-4.8. The communication complexity follows from the fact that n^2 instances of $(\mathsf{Sh}, \mathsf{Rec})$ are executed inside WSCC.

Theorem 4.9. *For any id* $(\mathsf{sid}, \mathsf{r})$ *with* $\mathsf{sid} \in \mathbb{N}$ *and* $\mathsf{r} \in \{1, 2, 3\}$, *protocol* WSCC *with id* $(\mathsf{sid}, \mathsf{r})$ *is a* $(0.139, 0.63)$*-WSCC scheme with communication complexity of* $\mathcal{O}(n^6 \log |\mathbb{F}|)$ *bits.*

5 The Shunning Common Coin (SCC) Protocol with $n = 3t + 1$

Here we present our SCC protocol SCC. In SCC, the parties execute three parallel instances of WSCC and it is guaranteed that each party obtains its output in at least two WSCC instances, based on which it decides its outcome for SCC and terminates. Note that different parties may end up using different pairs of WSCC instances to decide their outcome for SCC. Despite this, we ensure that the output of SCC for all honest parties is the same with a probability of at least 0.25. It is precisely to achieve this property for which the parties need to consider the outcome of two WSCC instances. The intuition behind the guaranteed output delivery in two out of the three WSCC instances is as follows: if there is an WSCC instance for which no honest party obtains its output, then the corresponding WSCCMM protocol ensures that communication from at least $\frac{t}{2}+1$ corrupt parties are ignored by all the honest parties in the remaining WSCC instances. As a consequence, there can be

at most $\frac{t}{2} - 1$ corrupt parties participating in the remaining WSCC instances. And these many corrupt parties are not sufficient to prevent output delivery in the remaining WSCC instances.

There might be a problem if a P_j terminates SCC (and the underlying WSCC instances) *immediately* after obtaining its output in two WSCC instances. This is because a different party, say P_i, may be still in the process of obtaining its outcome in two WSCC instances. And the participation of *all* honest parties is required for the output delivery in every WSCC instance. This is because output delivery in WSCC instances depend upon the termination of the underlying Rec instances, which is not guaranteed if only a strict subset of honest parties participate in the Rec instances. To deal with this problem, as soon as an honest P_j obtains its output from two WSCC instances, it broadcasts the identity of these WSCC instances, along with the corresponding H_j and S_j sets and then terminates SCC. Every party P_i who has not yet obtained its output from two WSCC instances check if the broadcasted H_j and S_j sets are subset of its own $\mathcal{G}_i$ and $\mathcal{S}_i$ sets respectively, which eventually happens (see Lemma 4.5). As a result, P_i can now decide its outcome for the WSCC instances based on the received H_j sets, instead of basing them on H_i sets (a similar idea is also used in the SCC protocol of [22] to ensure guaranteed termination). Protocol SCC is presented in Fig 5. Each instance of SCC is identified by a unique id $\mathsf{sid} \in \mathbb{N}$. Inside SCC, there are three invocations of the WSCC protocol with ids of the form $(\mathsf{sid}, \mathsf{r})$, where $\mathsf{r} \in \{1, 2, 3\}$.

Figure 5: Protocol $\mathsf{SCC}(\mathsf{sid})$: the shunning common coin protocol with $n = 3t + 1$ and id sid.

Each $P_i \in \mathcal{P}$ executes the following code:

1. Invoke $\mathsf{WSCC}(\mathsf{sid}, \mathsf{r})$ for $\mathsf{r} = 1, 2, 3$. Denote the instance $\mathsf{WSCC}(\mathsf{sid}, \mathsf{r})$ as WSCC_r and let $\mathcal{C}_{\mathsf{sid},\mathsf{r},i}, C_{\mathsf{sid},\mathsf{r},i}, \mathcal{G}_{\mathsf{sid},\mathsf{r},i}, G_{\mathsf{sid},\mathsf{r},i}, \mathcal{S}_{\mathsf{sid},\mathsf{r},i}, S_{\mathsf{sid},\mathsf{r},i}$ and $H_{\mathsf{sid},\mathsf{r},i}$ denote the $\mathcal{C}_i, C_i, \mathcal{G}_i, G_i, \mathcal{S}_i, S_i$ and H_i sets respectively, built during WSCC_r. // See protocol WSCC in Section 4 for the notations.
2. If output is obtained during WSCC_r, include $(\mathsf{sid}, \mathsf{r})$ to the decision set $\mathsf{DS}_{(i,\mathsf{sid})}$, initialized to $\emptyset$.
3. If $|\mathsf{DS}_{(i,\mathsf{sid})}| \geq 2$ then do the following, followed by terminating $\mathsf{WSCC}_1, \mathsf{WSCC}_2, \mathsf{WSCC}_3$, along with the underlying WSCCMM, SAVSS-MM instances, followed by terminating SCC:
   - Broadcast $(\texttt{Terminate}, P_i, \mathsf{DS}_{(i,\mathsf{sid})}, \{(S_{\mathsf{sid},\mathsf{r},i}, H_{\mathsf{sid},\mathsf{r},i})\}_{(\mathsf{sid},\mathsf{r}) \in \mathsf{DS}_{(i,\mathsf{sid})}})$.
   - Output 0 if 0 is computed as the output during WSCC_r for any $(\mathsf{sid}, \mathsf{r}) \in \mathsf{DS}_{(i,\mathsf{sid})}$, else output 1.
4. If $(\texttt{Terminate}, P_j, \mathsf{DS}_{(j,\mathsf{sid})}, \{(S_{\mathsf{sid},\mathsf{r},j}, H_{\mathsf{sid},\mathsf{r},j})\}_{(\mathsf{sid},\mathsf{r}) \in \mathsf{DS}_{(j,\mathsf{sid})}})$ is received from the broadcast of some party P_j, then do the following:

   a Check if all the following holds:
   - $|\mathsf{DS}_{(j,\mathsf{sid})}| \geq 2$;
   - For every $(\mathsf{sid}, \mathsf{r}) \in \mathsf{DS}_{(j,\mathsf{sid})}$, the condition $S_{\mathsf{sid},\mathsf{r},j} \subseteq \mathcal{S}_{\mathsf{sid},\mathsf{r},i}$ and $H_{\mathsf{sid},\mathsf{r},j} \subseteq \mathcal{G}_{\mathsf{sid},\mathsf{r},i}$ holds;
   - For every $(\mathsf{sid}, \mathsf{r}) \in \mathsf{DS}_{(j,\mathsf{sid})}$, the associated value v_k of every $P_k \in H_{\mathsf{sid},\mathsf{r},j}$ is computed during WSCC_r. // See WSCC in Section 4 for the meaning of associated values.

   b If all the above holds then do the following, followed by terminating $\mathsf{WSCC}_1, \mathsf{WSCC}_2, \mathsf{WSCC}_3$, along with the underlying WSCCMM, SAVSS-MM instances, followed by terminating SCC.
   - For every $(\mathsf{sid}, \mathsf{r}) \in \mathsf{DS}_{(j,\mathsf{sid})}$ where output is not yet obtained in WSCC_r, compute the output of WSCC_r as follows: If there exists a party $P_k \in H_{\mathsf{sid},\mathsf{r},j}$ whose associated value $v_k = 0$, then set 0 as the output of WSCC_r. Else, set 1 as the output of WSCC_r.
   - Output 0 if 0 is the output during WSCC_r for any $(\mathsf{sid}, \mathsf{r}) \in \mathsf{DS}_{(j,\mathsf{sid})}$, else output 1.

Now, we prove the properties of SCC and begin with the termination property. To prove the termination, we prove a crucial lemma, which states that there can be at most one instance WSCC_r among $\mathsf{WSCC}_1, \mathsf{WSCC}_2$ and WSCC_3, such that *no* honest party obtains its output during WSCC_r.

Lemma 5.1. *For any* $\mathsf{sid} \in \mathbb{N}$*, the following holds during the* SCC *protocol with id* sid*: if all the honest*

parties participate in SCC*, then there can be at most one instance out of* $\mathsf{WSCC}_1, \mathsf{WSCC}_2$ *and* WSCC_3 *in which no honest party obtains its output.*

Proof. Let $\mathsf{WSCC}_{\mathsf{min}}$ denote the first WSCC instance among $\mathsf{WSCC}_1, \mathsf{WSCC}_2$ and WSCC_3, such that no honest party obtains its output during $\mathsf{WSCC}_{\mathsf{min}}$. If $\mathsf{min} = 3$, then there is nothing to prove. So consider the case when $\mathsf{min} < 3$. In this case, it follows from Lemma 4.4 that at least $\frac{t}{2} + 1$ corrupt parties are shunned and not included in the $\mathcal{A}$ set $\mathcal{A}_{(i,\mathsf{sid},\mathsf{min})}$ of all the honest parties P_i. As a result, the messages of these $\frac{t}{2} + 1$ corrupt parties are not considered by any honest party in the subsequent invocations of WSCC, invoked as part of SCC. That is, for every WSCC_r, where $\mathsf{r} > \mathsf{min}$, the WSCCMM protocol with id $(\mathsf{sid}, \mathsf{r})$ does not allow the messages of the $\frac{t}{2} + 1$ shunned corrupt parties for processing by WSCC_r. This implies that there are at most $\frac{t}{2} - 1$ corrupt parties, whose messages are considered during WSCC_r. It now follows from the termination property of Rec (Corollary 3.3) that all Rec instances invoked collectively by the honest parties during WSCC_r eventually terminates. Moreover, as proved in Lemma 4.3, all honest party eventually sets its `Flag` variable to 1 during WSCC_r. Hence each honest party eventually obtains its output during WSCC_r. □

We next prove another important property, which is used to prove the termination of SCC. The property states that if some honest party terminates SCC by obtaining its output for two instances of WSCC during SCC, then every other honest party eventually terminates SCC.

Lemma 5.2. *For any* sid $\in \mathbb{N}$*, if all honest parties participate in the* SCC *protocol with id* sid*, then the following holds: if* $|\mathsf{DS}_{(j,\mathsf{sid})}| \geq 2$ *holds for some honest party* P_j*, then every honest party eventually terminates* SCC*.*

Proof. Let P_j be an honest party such that $|\mathsf{DS}_{(j,\mathsf{sid})}| \geq 2$ holds. This implies that P_j terminates SCC. In addition, party P_j broadcasts $(\texttt{Terminate}, P_j, \mathsf{DS}_{(j,\mathsf{sid})}, \{(S_{\mathsf{sid},\mathsf{r},i}, H_{\mathsf{sid},\mathsf{r},j})\}_{(\mathsf{sid},\mathsf{r}) \in \mathsf{DS}_{(j,\mathsf{sid})}})$. Let P_i be another honest party, which has not yet terminated SCC. The property of broadcast ensures that P_i eventually receives the above broadcast message. Moreover, from Lemma 4.5, it holds that for every WSCC instance $(\mathsf{sid}, \mathsf{r}) \in \mathsf{DS}_{(j,\mathsf{sid})}$, the conditions $S_{\mathsf{sid},\mathsf{r},j} \subseteq \mathcal{S}_{\mathsf{sid},\mathsf{r},i}$ and $H_{\mathsf{sid},\mathsf{r},j} \subseteq \mathcal{G}_{\mathsf{sid},\mathsf{r},i}$ are eventually true. Furthermore, for every $(\mathsf{sid}, \mathsf{r}) \in \mathsf{DS}_{(j,\mathsf{sid})}$, party P_i eventually computes the associated value of every $P_k \in H_{\mathsf{sid},\mathsf{r},j}$ during WSCC_r. This is because since P_j is able to compute these associated values by terminating the required Rec instances, it holds that P_i also eventually terminates these Rec instances (follows from Lemma 3.2, part three). Hence, as part of SCC, party P_i eventually computes its output for every WSCC instance $(\mathsf{sid}, \mathsf{r}) \in \mathsf{DS}_{(j,\mathsf{sid})}$ and hence terminates SCC. □

Lemma 5.3 (Termination of SCC). *For any* sid*, with* sid $\in \mathbb{N}$*, if each honest party participates in the protocol* SCC *with id* sid*, then each honest party eventually terminates* SCC*.*

Proof. From Lemma 5.1, there can be at most one instance WSCC_r among $\mathsf{WSCC}_1, \mathsf{WSCC}_2, \mathsf{WSCC}_3$, for which no honest party obtains its output. This implies that for the remaining two WSCC instances $\mathsf{WSCC}_{\mathsf{r}'} \in \{\mathsf{WSCC}_1, \mathsf{WSCC}_2, \mathsf{WSCC}_3\} \setminus \mathsf{WSCC}_\mathsf{r}$, there exists at least one honest party which eventually obtains its output for $\mathsf{WSCC}_{\mathsf{r}'}$. The proof now follows from Lemma 5.2. □

We next proceed to prove the correctness of SCC. Note that in SCC, a party P_i decides its outcome for any $\mathsf{WSCC}_\mathsf{r} \in \{\mathsf{WSCC}_1, \mathsf{WSCC}_2, \mathsf{WSCC}_3\}$, either based on its own H_i set or based on the H_j set of some other party P_j, who has obtained its output for WSCC_r. We first prove that irrespective of the way a party decides its outcome for WSCC_r, the correctness property for the instance WSCC_r holds. That is, with probability 0.139 and 0.63, all honest parties output 0 and 1 respectively during WSCC_r; otherwise at least $\frac{t}{4} + 1$ local conflicts occur.

Lemma 5.4. *For any* sid*, with* sid $\in \mathbb{N}$*, the following holds in the protocol* SCC *with id* sid*: if the honest parties are able to compute their output during any* $\mathsf{WSCC}_\mathsf{r} \in \{\mathsf{WSCC}_1, \mathsf{WSCC}_2, \mathsf{WSCC}_3\}$*, then one of the following holds:*

1. *All honest parties output $\sigma = 0$ during $\mathsf{WSCC_r}$ with probability at least $p_0 = 0.139$. And all honest parties output $\sigma = 1$ during $\mathsf{WSCC_r}$ with probability at least $p_1 = 0.63$; otherwise*

2. *At least $\frac{t}{4} + 1$ local conflicts occur during the protocol.*

Proof. Let P_i be an arbitrary honest party, such that P_i is able to compute its output during $\mathsf{WSCC_r} \in \{\mathsf{WSCC_1}, \mathsf{WSCC_2}, \mathsf{WSCC_3}\}$. Then there are two possible cases.

1. **Party P_i decides its output for $\mathsf{WSCC_r}$ based on its own set**[4] $H_{\mathsf{sid},\mathsf{r},i}$: In this case, the proof is exactly the same as in Lemma 4.8 and follows from the correctness property of WSCC.

2. **Party P_i decides its output for $\mathsf{WSCC_r}$ based on some other set**[5] $H_{\mathsf{sid},\mathsf{r},j}$: Here there are two further sub-cases. For any $P_k \in H_{\mathsf{sid},\mathsf{r},j}$, let $v_k \in [0, \ldots, u-1]$ denote the uniformly random value, which is guaranteed to exist, as per Lemma 4.6. If P_i is able to correctly associate v_k for every $P_k \in H_{\mathsf{sid},\mathsf{r},j}$, then again the proof is exactly the same as in Lemma 4.8. On the other hand, if for some $P_k \in H_{\mathsf{sid},\mathsf{r},j}$, party P_i associates $v'_k \neq v_k$, then as per Lemma 4.6, at least $\frac{t}{4} + 1$ local conflicts occur.

□

We next prove that if it is ensured that for every WSCC instance $\mathsf{WSCC_r} \in \{\mathsf{WSCC_1}, \mathsf{WSCC_2}, \mathsf{WSCC_3}\}$ for which all honest parties are able to compute their output, 0 and 1 can be the potential common outcome with probability 0.139 and 0.63 respectively, then for every value $\sigma \in \{0, 1\}$, with probability at least 0.25, all honest parties output σ in SCC.

Lemma 5.5. *For any* sid*, with* $\mathsf{sid} \in \mathbb{N}$*, the following holds in the protocol* SCC *with id* sid*: if it is ensured that for every* $\mathsf{WSCC_r} \in \{\mathsf{WSCC_1}, \mathsf{WSCC_2}, \mathsf{WSCC_3}\}$ *for which all honest parties are able to compute their output, the parties output* 0 *and* 1 *with probability* 0.139 *and* 0.63 *respectively, then for every value* $\sigma \in \{0, 1\}$*, with probability at least* 0.25*, all honest parties output* σ *on terminating* SCC

Proof. Let for every $\mathsf{WSCC_r} \in \{\mathsf{WSCC_1}, \mathsf{WSCC_2}, \mathsf{WSCC_3}\}$ where all the honest parties are able to compute their output, 0 and 1 occurs as the common output with probability 0.139 and 0.63 respectively. In SCC, each party sets its output based on the outcome of two WSCC instances for which the party is able to compute its local output. Due to asynchrony, different honest parties may consider different pairs of WSCC instances from $\{\mathsf{WSCC_1}, \mathsf{WSCC_2}, \mathsf{WSCC_3}\}$ to decide their output for SCC. Now in order that all the honest parties output 0 in SCC, the honest parties should output 0 in at least *two* WSCC instances [6] out of $\mathsf{WSCC_1}, \mathsf{WSCC_2}$ and $\mathsf{WSCC_3}$. The probability that at least two WSCC instances output 0 for all the honest parties is at least $1-(1-0.139)^2 \geq 0.25$. Hence, the probability that all honest parties output $\sigma = 0$ is at least 0.25. Similarly, in order that all the honest parties output 1 in SCC, the parties should output 1 in all the three WSCC instances $\mathsf{WSCC_1}, \mathsf{WSCC_2}$ and $\mathsf{WSCC_3}$. The probability of this is at least $(0.63)^3 \geq 0.25$. □

Finally, we state the correctness property for SCC, which simply follows from Lemma 5.4 and Lemma 5.5 respectively.

Lemma 5.6 (Correctness of SCC). *For any* sid *with* $\mathsf{sid} \in \mathbb{N}$*, if each honest party participates in the protocol* SCC *with id* sid*, then one of the following holds:*

[4] In this case, P_i executes step 3 of SCC to decide its output for $\mathsf{WSCC_r}$.

[5] In this case, P_i executes step $4(b)$ of SCC to decide its output for $\mathsf{WSCC_r}$

[6] Otherwise, if the common outputs for the honest parties in $\mathsf{WSCC_1}, \mathsf{WSCC_2}$ and $\mathsf{WSCC_3}$ are $0, 1$ and 1 respectively, then due to asynchrony some honest parties may end up considering $0, 1$ to determine their output for SCC, while others may end up considering $1, 1$ to determine their output for SCC, leading to different outputs for the parties.

1. *For every* $\sigma \in \{0,1\}$*, with probability at least* 0.25*, all honest parties output* σ *on terminating* SCC*; or*
2. *At least* $\frac{t}{4}+1$ *local conflicts occur during the protocol.*

The following theorem follows from Lemma 5.3-5.6. The communication complexity follows from the fact that three instances of WSCC are invoked inside SCC.

Theorem 5.7. *For every* sid $\in \mathbb{N}$*, protocol* SCC *is a* $\frac{1}{4}$*-SCC scheme, with communication complexity of* $\mathcal{O}(n^6 \log |\mathbb{F}|)$ *bits.*

6 Almost-Surely Terminating ABA

The design of our almost-surely terminating ABA protocol ABA from SCC is exactly the same as that of [1, 22]. We first recall a voting protocol called Vote from [8], which will be used in our ABA protocol. The current description of Vote and its properties is reproduced from [19].

6.1 Existing Voting Protocol

Informally, the voting protocol does "whatever can be done deterministically" to reach agreement. In a voting protocol, every party has a single bit as input. The protocol tries to find out whether there is a detectable majority for some value among the inputs of the parties. In the protocol, each party's output can have *five* different forms:

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

The voting protocol ensures the following properties:

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

The voting protocol consists of three "stages", each having a similar structure. The protocol called Vote is presented in Fig 6. In the protocol, each party P_i has the input bit x_i. Each instance of Vote has a unique id sid associated with it, where sid $\in \mathbb{N}$. For simplicity, we do not associate sid explicitly in the protocol steps of Vote.

The properties of the protocol Vote are stated in the following lemmas, whose proofs are available in [8]. For the sake of completeness, we recall the proofs here.

Lemma 6.1 ([8])**.** *For any* sid $\in \mathbb{N}$*, each honest party terminates the protocol* Vote *with id* sid *in a constant time.*

Proof. Every honest party P_i broadcasts its input x_i. As there are at least $n-t$ honest parties, from the properties of broadcast, for every honest P_i, eventually $|\mathcal{X}_i| = |\mathcal{Y}_i| = |Z_i| = n-t$ holds. Consequently, every honest P_i terminates the protocol in a constant time. □

Figure 6: Vote protocol with id sid.

Protocol Vote(sid)

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

1. On input x_i, broadcast $(\texttt{input}, P_i, x_i)$.
2. Create a dynamic set $\mathcal{X}_i$, which is initialized to $\emptyset$. Add (P_j, x_j) to $\mathcal{X}_i$ if $(\texttt{input}, P_j, x_j)$ is received from the broadcast of P_j.
3. Wait until $|\mathcal{X}_i| = n - t$. Assign $X_i = \mathcal{X}_i$. Set a_i to the majority bit among $\{x_j \mid (P_j, x_j) \in X_i\}$ and broadcast $(\texttt{vote}, P_i, X_i, a_i)$.
4. Create a dynamic set $\mathcal{Y}_i$, which is initialized to $\emptyset$. Add (P_j, X_j, a_j) to $\mathcal{Y}_i$ if $(\texttt{vote}, P_j, X_j, a_j)$ is received from the broadcast of P_j, $X_j \subseteq \mathcal{X}_i$, and a_j is the majority bit of X_j.
5. Wait until $|\mathcal{Y}_i| = n - t$. Assign $Y_i = \mathcal{Y}_i$. Set b_i to the majority bit among $\{a_j \mid (P_j, X_j, a_j) \in Y_i\}$ and broadcast $(\texttt{re-vote}, P_i, Y_i, b_i)$.
6. Create a set Z_i, which is initialized to $\emptyset$. Add (P_j, Y_j, b_j) to Z_i if $(\texttt{re-vote}, P_j, Y_j, b_j)$ is received from the broadcast of P_j, $Y_j \subseteq \mathcal{Y}_i$, and b_j is the majority bit of Y_j.
7. Wait until $|Z_i| = n - t$. If all the parties $P_j \in Y_i$ have the same $\texttt{vote}\ a_j = \sigma$, then output $(\sigma, 2)$ and terminate.
   Otherwise, if all the parties $P_j \in Z_i$ have the same $\texttt{re-vote}\ b_j = \sigma$, then output $(\sigma, 1)$ and terminate.
   Otherwise, output $(\Lambda, 0)$ and terminate.

Lemma 6.2 ([8]). *For any* sid $\in \mathbb{N}$*, if every honest party has the same input* σ *for the* Vote *protocol with id* sid*, then each honest party outputs* $(\sigma, 2)$ *in the* Vote *protocol with id* sid.

Proof. Consider an arbitrary honest party P_i. If all the honest parties have the same input σ, then at most t (corrupt) parties may broadcast $\overline{\sigma}$ as their input. Therefore, it is easy to see that *every* $P_k \in \mathcal{Y}_i$ must have broadcasted its $\texttt{vote}\ b_k = \sigma$. Hence the honest P_i outputs $(\sigma, 2)$. □

Lemma 6.3 ([8]). *For any id* sid $\in \mathbb{N}$*, if some honest party outputs* $(\sigma, 2)$ *during the* Vote *protocol with id* sid*, then every other honest party outputs either* $(\sigma, 2)$ *or* $(\sigma, 1)$ *during the* Vote *protocol with id* sid.

Proof. Let an *honest* P_i outputs $(\sigma, 2)$. This implies that every $P_j \in Y_i$ broadcasts $\texttt{vote}\ a_j = \sigma$. As $|Y_i| = 2t + 1$, it implies that for every other *honest* party P_k, it holds that $|Y_i \cap Y_k| \geq t + 1$ and so P_k is bound to broadcast $\texttt{re-vote}\ b_k = \sigma$ and hence outputs either $(\sigma, 2)$ or $(\sigma, 1)$. □

Lemma 6.4 ([8]). *For any id* sid $\in \mathbb{N}$*, if some honest party outputs* $(\sigma, 1)$ *and no honest party outputs* $(\sigma, 2)$ *during the* Vote *protocol with id* sid*, then every other honest party outputs either* $(\sigma, 1)$ *or* $(\Lambda, 0)$ *during the* Vote *protocol with id* sid.

Proof. Assume that an *honest* party P_i outputs $(\sigma, 1)$. This implies that all the parties $P_j \in Z_i$ broadcasts the same $\texttt{re-vote}\ b_j = \sigma$. Since $|Z_i| = n - t$, in the worst case there can be most t parties (outside Z_i) who may broadcast $\texttt{re-vote}\ \overline{\sigma}$. Thus it is clear that no honest party outputs $(\overline{\sigma}, 1)$. Now since the honest parties in Z_i broadcast their $\texttt{re-vote}$ as σ, there must be at least $t + 1$ parties who broadcasts their $\texttt{vote}$ as

σ. Thus no honest party outputs $(\overline{\sigma}, 2)$ for which at least $n - t = 2t + 1$ parties are required to broadcast their `vote` as $\overline{\sigma}$. Hence no honest party outputs from $\{(\overline{\sigma}, 2), (\overline{\sigma}, 1)\}$. Therefore the honest parties output either $(\sigma, 1)$ or $(\Lambda, 0)$. □

The communication complexity of the protocol Vote is stated in the following lemma.

Lemma 6.5. *Protocol* Vote *has a communication complexity of* $\mathcal{O}(n^4 \log n)$ *bits.*

Proof. In the protocol, each party broadcasts X, Y and Z sets, each containing the identity of $\mathcal{O}(n)$ parties. Since the identity of each party can be represented by $\log n$ bits, the protocol requires a broadcast of $\mathcal{O}(n^2 \log n)$ bits and hence a total communication of $\mathcal{O}(n^4 \log n)$ bits. □

6.2 The ABA Protocol

Given protocol SCC and Vote, we design our ABA protocol called ABA for a single bit (Fig 7), following [1, 22]. The protocol is based on the idea of [8], which uses CC and Vote for several rounds to decide the outcome. In our case, we replace the CC protocol by our SCC protocol. In the protocol, during each iteration, every party computes a "modified input" value. In the first iteration, the modified input of a party P_i is its private input bit (for the ABA protocol) x_i. In each iteration, the parties execute an instance of the protocol Vote and SCC *sequentially*, that is, a party participates in the instance of SCC, only after terminating the instance of Vote (the reason for this provision will be clear while proving the properties of the ABA protocol). If a party outputs $(\sigma, 1)$ in the instance of the Vote protocol, implying that it finds a "distinct majority" for the value σ, then the party sets its modified input for the next iteration to σ, irrespective of the value which is going to be output in the instance of SCC; otherwise, the party sets its modified input for the next iteration to be the output of the SCC protocol, which is invoked by all the parties in each iteration, irrespective of whether the output of the SCC protocol is used or not by the parties for setting the modified inputs for the next iteration. Once a party outputs $(\sigma, 2)$ in an instance of the Vote protocol, implying that it finds an "overwhelming majority" for the value σ, then it broadcasts σ. Finally, once a party receives σ from the broadcast of $t + 1$ parties, it outputs σ and terminates.

We now proceed to prove the properties of ABA, which are similar to [22], which in turn are slight variants of the properties proved in [9]. We first prove the validity property.

Lemma 6.6. *If all the honest parties have the same input* σ*, then each honest party outputs* σ *during* ABA.

Proof. If every honest party has the same input σ during the ABA protocol, then by Lemma 6.2, all honest party outputs $(y_1, m_1) = (\sigma, 2)$ at the end of the Vote protocol during the first iteration. Hence every honest party broadcasts the message (`Terminate with` σ) during the first iteration. As there are at least $n - t$ honest parties, each honest party eventually receives $n - t$ (`Terminate with` σ) broadcast message and at most t (`Terminate with` $\overline{\sigma}$) broadcast message. Consequently, each honest party outputs σ and terminates the protocol at the end of second iteration. □

We next prove the agreement property.

Lemma 6.7. *In protocol* ABA*, if some honest party terminates with output* σ*, then every other honest party eventually terminates* ABA *with output* σ*.*

Proof. We first show that if an honest party broadcasts (`Terminate with` σ) for some value σ, then every other honest party eventually broadcasts (`Terminate with` σ). Let k be the first iteration, when some honest party, say P_i, broadcasts (`Terminate with` σ). Then, from the property of Vote protocol (Lemma 6.3), every

Figure 7: Almost-surely terminating ABA protocol for a single bit with $n = 3t + 1$.

ABA

Each $P_i \in \mathcal{P}$ executes the following code:

1. Set $\mathsf{sid} = 0$ and $v_i = x_i$, where $x_i \in \{0, 1\}$ is the input bit for party P_i.
2. Repeat until terminating: (each iteration is considered as a round)
   a. Set $\mathsf{sid} = \mathsf{sid} + 1$. Participate in $\mathsf{Vote}(\mathsf{sid})$ with input v_{sid} and wait for its termination. Let $(y_{\mathsf{sid}}, m_{\mathsf{sid}})$ denote the output of $\mathsf{Vote}(\mathsf{sid})$.
   b. Invoke $\mathsf{SCC}(\mathsf{sid})$ and wait until its termination. Let c_{sid} denote the output of $\mathsf{SCC}(\mathsf{sid})$.
   c. Consider the following cases:
      i. If $m_{\mathsf{sid}} = 2$, then set $v_{\mathsf{sid}+1} = y_{\mathsf{sid}}$ and broadcast the message $(\texttt{Terminate with } v_{\mathsf{sid}})$. Participate in only one more instance of Vote protocol and only one more instance of SCC.
      ii. If $m_{\mathsf{sid}} = 1$, then set $v_{\mathsf{sid}+1} = y_{\mathsf{sid}}$.
      iii. Else, set $v_{\mathsf{sid}+1} = c_{\mathsf{sid}}$
   d. Upon receiving the message $(\texttt{Terminate with } \sigma)$ for some value $\sigma \in \{0, 1\}$ from the broadcast of $t + 1$ different parties, output σ and terminate the protocol.

honest party outputs $y_k = \sigma$ and either $m_k = 2$ or $m_k = 1$ at the end of $\mathsf{Vote}(k)$. Hence no honest party broadcasts $(\texttt{Terminate with } \overline{\sigma})$ during iteration k. Furthermore, all honest parties (including P_i) participate in $\mathsf{Vote}(k + 1)$ with input σ. So from the property of Vote (Lemma 6.2), at the end of $\mathsf{Vote}(k + 1)$, each honest party has $(y_{k+1}, m_{k+1}) = (\sigma, 2)$. Hence, each honest party broadcasts $(\texttt{Terminate with } \sigma)$ either during iteration k or $k + 1$.

Now let some honest party, say P_h, terminate ABA with output σ. This implies that P_h receives $(\texttt{Terminate with } \sigma)$ broadcast message from $t + 1$ different parties. This implies that at least one honest party broadcasts $(\texttt{Terminate with } \sigma)$. Consequently, each honest party eventually broadcasts $(\texttt{Terminate with } \sigma)$. Hence, each honest party eventually receives $n - t$ $(\texttt{Terminate with } \sigma)$ broadcast message and at most t $(\texttt{Terminate with } \overline{\sigma})$ broadcast message. Hence, each honest party outputs σ. □

We next prove a crucial lemma, which states that at the end of each iteration, the updated value of all honest parties will be the same with probability at least $\frac{1}{4}$, or $\frac{t}{4} + 1$ *new* local conflicts occur.

Lemma 6.8. *In protocol* ABA*, if all honest parties initiate and complete iteration k, then one of the following holds:*

- *With probability at least $\frac{1}{4}$, all honest parties have the same value for v_{k+1}; or*
- *A new set of $\frac{t}{4} + 1$ local conflicts occur.*

Proof. We first note that P_i is *honest* and if at all any local conflict (P_i, P_j) occurs during iteration k of ABA, then the conflict is different from any local conflict of the form $(P_i, \star)$, which could have occurred during any iteration k' of ABA, where $k' < k$. On contrary, let the local conflict (P_i, P_j) occurs both during iteration k' as well as k. Since the conflict occurs during the iteration k', it follows that during the execution of $\mathsf{SCC}(k')$, one of the underlying memory management protocols includes P_j to the set $\mathcal{B}_i$. As a result, any communication from party P_j is completely ignored by P_i in any instance $\mathsf{SCC}(k)$, with $k > k'$. Consequently, the local conflict (P_i, P_j) does not re-occur during iteration k, which is a contradiction.

Now to prove the lemma statement, we have to consider two cases. If all honest parties set v_{k+1} to the output of $\mathsf{SCC}(k)$ (namely by executing step 2.c(iii) of ABA), then the lemma is true as per the correctness property of SCC (Lemma 5.6). Otherwise, let some honest party set v_{k+1} to the output of $\mathsf{Vote}(k)$. That is, some honest party sets $v_{k+1} = \sigma$ for some $\sigma \in \{0, 1\}$, either during step 2.c(i) or step 2.c(ii) of iteration k. The property of Vote (Lemma 6.4) ensures that the other honest parties who set v_{k+1} either during step 2.c(i) or step 2.c(ii) of iteration k, never set v_{k+1} to $\bar{\sigma}$. The correctness property of SCC (Lemma 5.6) ensures that with probability at least $\frac{1}{4}$, the outcome of $\mathsf{SCC}(k)$ is the same as σ for all the honest parties, or otherwise a set of $\frac{t}{4} + 1$ local conflicts are revealed. Hence, even for the honest parties who set v_{k+1} as the output of $\mathsf{Vote}(k)$ instead of $\mathsf{SCC}(k)$, the value v_{k+1} is either set to σ or a set of $\frac{t}{4}+1$ local conflicts are revealed. □

As a corollary of Lemma 6.8, we can state that there can be at most $8t + 4$ iterations where the honest parties have the same updated value at the end with probability *strictly less* than $\frac{1}{4}$. This is because there are at most $(n - t)t$ different local conflicts which can occur throughout ABA. From Lemma 6.8, if the honest parties do not have the same updated value at the end of an iteration with probability $\frac{1}{4}$ or more, a new set of $\frac{t}{4} + 1$ local conflicts occur. Hence there can be at most $\frac{(n-t)t}{\frac{t}{4}+1} \leq 8t + 4$ such iterations, as $n = 3t + 1$. We formally state this property, which will be crucial while proving the expected running time of ABA.

Corollary 6.9. *Let $\mathcal{I}$ denote the set of iterations k in* ABA*, such that all the honest parties terminate iteration k but have the same value for v_{k+1} with probability less than $\frac{1}{4}$. Then $|\mathcal{I}| \leq 8t + 4$.*

Lemma 6.10. *The honest parties terminate* ABA *within constant time after the first honest party broadcasts the message* (`Terminate with` σ) *for some $\sigma \in \{0, 1\}$.*

Proof. Assume that in ABA, the first honest party broadcasts the message (`Terminate with` σ) for some $\sigma \in \{0, 1\}$ during iteration k. Then all honest parties participate in $\mathsf{Vote}(k + 1)$ as well as $\mathsf{SCC}(k + 1)$. As seen in the proof of Lemma 6.7, every honest party broadcasts the message (`Terminate with` σ) by iteration $k + 1$. All these instances of broadcast complete in constant time. And each honest party terminates ABA after completing $t + 1$ of these broadcasts. Consequently, once the first honest party broadcasts the message (`Terminate with` σ), each honest party terminates the protocol in a constant time. □

We now proceed to derive the expected running time of ABA. We begin with a simple case, where we assume that at the end of *each* iteration of ABA, all honest parties have the same updated modified input (which occurs except with probability at most $\frac{3}{4}$); we show in this case, protocol ABA requires 16 iterations in expectation.

Lemma 6.11. *In protocol* ABA*, if for every iteration k, all the honest parties have the same updated modified input for the $(k + 1)$th iteration with probability at least $\frac{1}{4}$, then the protocol requires expected 16 rounds to terminate.*

Proof. We first note that in ABA, every iteration k in which all honest parties participate, eventually terminates for every honest party. This is because the instance of Vote and SCC in such iterations terminate for all honest parties, which follows from the termination properties of Vote (Lemma 6.1) and SCC (Lemma 5.3). From Lemma 6.10, the honest parties terminate ABA within constant time, once an honest party broadcasts the message (`Terminate with` σ) for some $\sigma \in \{0, 1\}$. Now as per the lemma condition, for every iteration k, all the honest parties have the same updated modified input for the $(k+1)$th iteration, except with probability at most $\frac{3}{4}$. To complete the proof, we need to show that in this case, there can be 16 expected iterations in ABA until some honest party broadcasts the message (`Terminate with` σ). Let τ be a random variable which counts the number of iterations until some honest party broadcasts the message (`Terminate with` σ). The probability that $\tau = k$ i.e. $\Pr(\tau = k)$, is given as:

$$\Pr(\tau \neq 1) \cdot \Pr(\tau \neq 2 | \tau \neq 1) \cdot \ldots \cdot \Pr(\tau \neq (k-1) | \tau \neq 1 \cap \ldots \cap \tau \neq (k-2)) \cdot \Pr(\tau = k | \tau \neq 1 \cap \ldots \cap \tau \neq (k-1))$$

Now, as per the lemma condition, each multiplicand on the right hand side, except the last one, is upper bounded by $\frac{3}{4}$. And the last multiplicand is upper bounded by $\frac{1}{4}$ as this is equivalent to the probability of terminating in a particular iteration. Hence we get $\Pr(\tau = k) \leq (\frac{3}{4})^{k-1}(\frac{1}{4})$. Now the expected value $E(\tau)$ of τ is computed as follows:

$$\begin{aligned} E(\tau) &= \sum_{k=0}^{\infty} k \Pr(\tau = k) \\ &\leq \sum_{k=0}^{\infty} k(\frac{3}{4})^{k-1}(\frac{1}{4}) \\ &\leq \sum_{k=0}^{\infty} k(\frac{3}{4})^{k} = \frac{1}{1-\frac{3}{4}} + \frac{(1)(\frac{3}{4})}{(1-\frac{3}{4})^2} = 16 \end{aligned}$$

The expression for $E(\tau)$ is a sum of AGP upto infinite terms, which is given by $\frac{a}{1-r} + \frac{dr}{(1-r)^2}$, where $a = 1$, $r = \frac{3}{4}$ and $d = 1$. Hence, we have that $E(\tau) \leq 16$. □

We next derive the expected number of rounds required in the protocol ABA. This automatically derives the expected running time of ABA, as each round in ABA requires a constant time.

Lemma 6.12. *Protocol* ABA *terminates for the honest parties in $\mathcal{O}(n)$ expected running time.*

Proof. From Lemma 6.10, the honest parties terminate ABA within constant time, once an honest party broadcasts the message (`Terminate with` σ) for some $\sigma \in \{0, 1\}$. To complete the proof, we claim that there can be $8t + 20 = \mathcal{O}(n)$ expected number of iterations in ABA until some honest party broadcasts the message (`Terminate with` σ). This follows from Corollary 6.9 and Lemma 6.11. Specifically, there can be at most $8t + 4$ iterations in ABA, where at the end of the iteration, the modified input values of the honest parties are different with probability more than $\frac{3}{4}$. After this, in each iteration of ABA, the honest parties will have the same modified input value, except with probability at most $\frac{3}{4}$ and as a result, ABA will require expected 16 rounds. As each round requires constant time, it follows that ABA terminates for the honest parties in $\mathcal{O}(n)$ expected running time. □

The following theorem follows from Lemma 6.6, Lemma 6.7 and Lemma 6.12. The expected communication complexity follows from the fact that there are $\mathcal{O}(n)$ expected number of iterations executed in ABA and each iteration involves one instance of Vote and one instance of SCC.

Theorem 6.13. *Let $n = 3t+1$. Then protocol* ABA *is an almost-surely terminating ABA protocol for a single bit with expected communication complexity of $\mathcal{O}(n^7 \log |\mathbb{F}|))$ bits and expected running time of $\mathsf{R} = \mathcal{O}(n)$.*

7 Variants of Our ABA Protocol

We first discuss how to modify the protocol ABA to get an almost-surely terminating ABA protocol MABA, which allows to reach agreement on $t + 1$ bits simultaneously, without affecting the expected communication complexity.

7.1 Agreement on $t + 1$ Bits Simultaneously

Let the parties want to reach agreement on $t + 1 = \Theta(n)$ bits simultaneously in an almost-surely terminating fashion. A trivial way to do this is to invoke $t + 1$ independent instances of ABA, which will have an

expected communication complexity of $\mathcal{O}(n^8 \log |\mathbb{F}|))$ bits, with $\mathsf{R} = \mathcal{O}(n)$ being the expected running time of the protocol. Instead, we show how to achieve agreement on $t+1$ bits simultaneously, with expected communication complexity of $\mathcal{O}(n^7 \log |\mathbb{F}|)$ bits. So in the amortized sense, agreement on a single bit involves an expected communication complexity of $\mathcal{O}(n^6 \log |\mathbb{F}|)$ bits. We first discuss how we can modify WSCC to obtain a multi-valued WSCC protocol MWSCC, which allows the parties to obtain $t+1$ independent common coins, instead of a single common coin. Before doing that, we discuss a well known randomness-extraction technique, used in MWSCC.

Information-Theoretic Randomness Extraction [12, 3, 19]. Let $a_1, \ldots, a_N \in \mathbb{F}$, such that *at least* K out of these N values are selected uniformly at random from $\mathbb{F}$; however, the exact identities of those K values are unknown. The goal is to compute K values from $a_1, \ldots, a_N$, say $b_1, \ldots, b_K$, each of which is uniformly distributed over $\mathbb{F}$. This is achieved as follows: let $f(x)$ be the polynomial of degree at most $N-1$, such that $f(i) = a_{i+1}$, for $i = 0, \ldots, N-1$. Then set $b_1 = f(N), \ldots, b_K = f(N+K-1)$ (note that, we require $|\mathbb{F}| \geq N+K$ to make the technique work; in our protocols, N, K and $|\mathbb{F}|$ will be such that this relationship holds). The elements $b_1, \ldots, b_K$ are uniformly distributed over $\mathbb{F}$, as there exists a one-to-one mapping between $b_1, \ldots, b_K$ and the K random elements in the vector $(a_1, \ldots, a_N)$. We call this algorithm as Extrand and invoke it as $(b_1, \ldots, b_K) = \mathsf{Extrand}(a_1, \ldots, a_N)$. We are now ready to describe the protocol MWSCC.

From WSCC to MWSCC: The idea behind extending WSCC to MWSCC is based on a technique of [19]. Each party P_i now ensures that its C_i set is $2t+1$ instead of $t+1$. We note that this eventually happens for each *honest* P_i, as there are at least $2t+1$ Sh instances, invoked by the honest dealers, which eventually terminates for P_i. This modification ensures that there are at least $t+1$ honest parties in the set C_i, who share truly random and unknown values on the behalf of P_i. This implies that at least $t+1$ truly random values from $\mathbb{F}$ are *attached* to P_i. Now let $\{s_{ji}\}_{P_j \in C_i}$ denote the secrets attached to P_i. And let $\{r_{ji}\}_{P_j \in C_i}$ denote the corresponding values, which are later reconstructed. If $\{r_{ji}\}_{P_j \in C_i} \neq \{s_{ji}\}_{P_j \in C_i}$, then from the proof of the correctness property of WSCC (Lemma 4.8), at least $\frac{t}{4}+1$ local faults occur. On the other hand, consider the case when $\{r_{ji}\}_{P_j \in C_i} = \{s_{ji}\}_{P_j \in C_i}$. As at least $t+1$ values in the set $\{r_{ji}\}_{P_j \in C_i}$ are truly random, it follows from the properties of Extrand that if we set $(V_{i1}, \ldots, V_{i(t+1)}) = \mathsf{Extrand}(\{r_{ji}\}_{P_j \in C_i})$, with N and K being $2t+1$ and $t+1$ respectively, then the values $V_{i1}, \ldots, V_{i(t+1)}$ will be truly random values from $\mathbb{F}$. This further implies that if we set $v_{ik} \stackrel{def}{=} V_{ik} \mod u$ for $k = 1, \ldots, t+1$, then the $t+1$ values $v_{i1}, \ldots, v_{i(t+1)}$ will be uniformly distributed over $[0, \ldots u-1]$. These $t+1$ values are now considered as the $t+1$ values *associated* with P_i. This way, instead of associating one random value in the range $[0, \ldots, u-1]$ with P_i, the parties now associate $t+1$ random values.

Now the decision rule to decide the final outcome of MWSCC is extended for the case of $t+1$ bits, by applying $t+1$ times the decision rule used in WSCC. Specifically, to decide the lth output bit σ_l for MWSCC, party P_i checks if for some party in H_i, the lth associated value is 0. If so, then σ_l is set to 0, else it is set to 1. More formally, for $l \in \{1, \ldots, t+1\}$, party P_i sets $\sigma_l = 0$, if for some $P_k \in H_i$, the lth associated value v_{kl} is 0, else P_i sets σ_l to 1. Finally, party P_i outputs $t+1$ values $\sigma_1, \ldots, \sigma_{t+1}$. We note that communication complexity of MWSCC remains the same as WSCC. We summarize the above discussion in the following lemma, whose proof is similar to that of WSCC and so we do not provide it again to avoid repetition.

Lemma 7.1. *For any id* $(\mathsf{sid}, \mathsf{r})$ *where* $\mathsf{sid} \in \mathbb{N}$ *and* $\mathsf{r} \in \{1, 2, 3\}$*, the following holds during* $\mathsf{MWSCC}(\mathsf{sid}, \mathsf{r})$*:*

1. **Correctness***: If all honest parties are able to compute their output, then one of the following holds:*
   - *(a) For every* $l \in \{1, \ldots, t+1\}$*, all honest parties output* $\sigma_l = 0$ *and* $\sigma_l = 1$ *with probability at least* 0.139 *and* 0.63 *respectively; otherwise*
   - *(b) At least* $\frac{t}{4}+1$ *local conflicts occur during the protocol.*

2. **Communication Complexity**: *The protocol has communication complexity of* $\mathcal{O}(n^6 \log |\mathbb{F}|)$ *bits.*

We next discuss how we can extend SCC to get a multi-valued SCC protocol MSCC, which allows the parties to obtain $t+1$ independent common coins, instead of a single common coin, with guaranteed termination.

From SCC **to** MSCC: Protocol MSCC is a straight-forward extension of SCC to deal with $t+1$ output bits. More specifically, instead of executing three parallel instances of WSCC, the parties now execute three parallel instances of MWSCC and wait for receiving $t+1$ output bits in two out of these three instances. The decision rule used to decide the output bit in SCC, is now applied $t+1$ times. That is, each party P_i decides its $t+1$ outputs bits in MSCC as follows: to decide the lth output bit, party P_i checks the two MWSCC instances, for which P_i is able to obtain its output (as in SCC, it can be proved that each honest P_i eventually obtains its output in at least two out of the three MWSCC instances). Now if the lth output bit in any of these two MWSCC instances is 0, then P_i sets the lth output bit for MSCC to 0, else it is set to 1. The properties of MSCC (stated in Lemma 7.2) follows in a straight-forward fashion from the properties of SCC and so we do not provide the formal proof to avoid repetition.

Lemma 7.2. *For any id* sid *where* $\mathsf{sid} \in \mathbb{N}$*, the following holds during* MSCC(sid)*:*

1. **Termination**: *If all honest parties participate in* MSCC*, then each honest party eventually terminates.*
2. **Correctness**: *One of the following holds:*
   (a) *For every* $l \in \{1, \ldots, t+1\}$*, for every possible* $\sigma_l \in \{0,1\}$*, with probability at least* 0.25*, all honest parties output* σ_l *as the* l*th output bit on terminating* SCC*; otherwise*
   (b) *At least* $\frac{t}{4}+1$ *local conflicts occur during the protocol.*
3. **Communication Complexity**: *The protocol has communication complexity of* $\mathcal{O}(n^6 \log |\mathbb{F}|)$ *bits.*

Finally we discuss how to extend ABA to MABA.

From ABA **to** MABA: Protocol MABA is similar to ABA, except that the parties use MSCC instead of SCC; moreover, $t+1$ instances of Vote protocol are invoked in each iteration. Each party participates in MABA with an input vector of $t+1$ bits. Then the protocol consists of several iterations of $t+1$ instances of Vote followed by one instance of MSCC and at the end of each iteration, the parties update their vector of $t+1$ bits. More specifically, during the first iteration, the value for each party will consists of its private $t+1$ input bits. Then in each iteration, the parties execute $t+1$ parallel instances of the Vote protocol (one instance on behalf of each bit), followed by a single instance of the MSCC. Each party then applies the same logic as in ABA $t+1$ times, taking into account the outcome of the Vote and MSCC, to update its $t+1$ values for the next iteration.

More formally, after completing the iteration k, each party updates its lth bit for the $(k+1)$th iteration as follows, for each $l \in \{1, \ldots, t+1\}$: if $(\sigma_l, 1)$ is obtained as the output of the lth instance of Vote during the kth iteration, then the lth bit is updated to σ_l; otherwise the lth bit is updated to the lth common coin, obtained at the end of MSCC protocol during the kth iteration. This process is repeated till $(\sigma_l, 2)$ is obtained as the output of the lth instance of Vote during some iteration, in which case, a party stops all computations related to the lth bit and broadcasts the `Terminate` message for σ_l. Protocol MABA finally terminates when $t+1$ `Terminate` message is broadcasted for each of the $t+1$ output bits. For formal details, see Fig 8.
The properties of MABA (stated in Theorem 7.3) follow using similar arguments as used for ABA and from the properties of MWSCC and MSCC. We do not give the complete proof to avoid repetition.

Theorem 7.3. MABA *is an almost-surely terminating ABA protocol for* $t+1 = \Theta(n)$ *bits with* $\mathsf{R} = \mathcal{O}(n)$ *expected running time and expected communication complexity of* $\mathcal{O}(n^7 \log |\mathbb{F}|)$ *bits. In an amortized sense, to agree on a single bit, the protocol requires an expected communication complexity of* $\mathcal{O}(n^6 \log |\mathbb{F}|)$ *bits.*

Figure 8: Almost-surely terminating ABA protocol for $t+1$ bits with $n = 3t+1$.

MABA

Each $P_i \in \mathcal{P}$ executes the following code:

1. Set $\mathsf{sid} = 0$. On having the input $(x_{i1}, \ldots, x_{i(t+1)})$, set $v_{il} = x_{il}$, for $l = 1, \ldots, t+1$ and set $\mathsf{flag}_l = 0$. [a]
2. Repeat until terminating: (each iteration is considered as a round)
   a. Set $\mathsf{sid} = \mathsf{sid} + 1$. Participate in $t+1$ instances of Vote, where for $l \in \{1, \ldots, t+1\}$, the input for the lth instance is v_{il}. Let $(y_{(\mathsf{sid},l)}, m_{(\mathsf{sid},l)})$ denote the output of the lth instance of Vote.
   b. Wait to terminate all the $t+1$ instances of the Vote protocol, executed in the previous step. Then invoke MSCC(sid) and wait until its termination. Let $(c_{(\mathsf{sid},1)}, \ldots, c_{(\mathsf{sid},t+1)})$ denote the output of MSCC(sid).
   c. For every $l \in \{1, \ldots, t+1\}$, such that $\mathsf{flag}_l = 0$, do the following:
      i. If $m_{(\mathsf{sid},l)} = 2$, then set $v_{(\mathsf{sid}+1,l)} = y_{(\mathsf{sid},l)}$, broadcast the message $(\texttt{Terminate with } v_{(\mathsf{sid}+1,l)}, l)$ and participate in one more instance of Vote protocol corresponding to the l^{th} bit with $v_{(\mathsf{sid}+1,l)}$ as the input.
      Participate in one more instance of MSCC if $(\texttt{Terminate with } v_{(\mathsf{sid}+1,l)}, l)$ is broadcasted for all $l = 1, \ldots, t+1$.
      i.. If $m_{(\mathsf{sid},l)} = 1$, set $v_{(\mathsf{sid}+1,l)} = y_{(\mathsf{sid},l)}$.
      iii. Else, set $v_{(\mathsf{sid}+1,l)} = c_{(\mathsf{sid},l)}$.
   d. Upon receiving $(\texttt{Terminate with } \delta_l, l)$ from the broadcast of at least $t+1$ parties, for some value $\delta_l \in \{0,1\}$, output δ_l as the lth bit, terminate all the computation regarding lth bit and set $\mathsf{flag}_l = 1$.
   e. If $\mathsf{flag}_l = 1$ for every $l \in \{1, \ldots, t+1\}$ then terminate the protocol with output $(\delta_1, \ldots, \delta_{t+1})$.

[a] Here $\mathsf{flag}_1, \ldots, \mathsf{flag}_{t+1}$ are the (local) Boolean flags to indicate whether the agreement on the lth bit has been achieved.

7.2 Almost Surely Terminating ABA with a Constant Expected Running Time

We have already seen that the expected running time of our ABA protocol for $n = 3t+1$ is $\mathcal{O}(n)$. We now show how the expected running time of our ABA protocol can be reduced from $\mathcal{O}(n)$ to $\mathcal{O}(\frac{1}{\epsilon})$ for $n \geq (3+\epsilon)t$ where $\epsilon > 0$. This improvement in the expected running time is ultimately the result of the modifications made to our SAVSS protocol. So, we first discuss our modified SAVSS protocol.

The Modified SAVSS Scheme. The modified SAVSS comprises of a pair of protocols (CSh, CRec). While CSh is the same as Sh, the following modification is made in Rec to get CRec:

– All RS decoding procedures are invoked as RS-Dec$(t, \frac{2n-5t-2}{4}, \mathcal{K}_j)$; hence the parameter c is changed from $\frac{t}{4}$ to $\frac{2n-5t-2}{4}$. Notice that $N = |\mathcal{K}_j| = n - t - \frac{t}{2}$ is automatically changed from $\frac{3t}{2}+1$ to $\frac{3t}{2} + \epsilon t$. This is because now $n \geq (3+\epsilon)t$, instead of $n = 3t+1$.

In Rec, at least $\frac{t}{4}+1$ local conflicts occur whenever correctness property is violated. We next show that at least $\frac{\epsilon t^2}{4}(1+2\epsilon)$ local conflicts occur in CRec if correctness is violated. The properties of (CSh, CRec) are formally stated in lemma 7.4.

Lemma 7.4. *For any* $\mathsf{sid} \in \mathbb{N}$, *the following holds during the* (CSh, CRec) *protocol with id* sid*:*

1. **Termination**: *The following holds:*

   (a) *If* D *is honest and all honest parties participate in* CSh*, then each honest party eventually terminates* CSh.

   (b) *If some honest party terminates* CSh*, then every other honest party eventually terminates* CSh.

   (c) *If all honest parties participate in* CRec*, then one of the following holds:*

      i. *All honest parties eventually terminate* CRec*; or*

      ii. *At least* $\frac{t}{2}+1$ *corrupt parties are included in the* $\mathcal{W}$ *set* $\mathcal{W}_{(i,\mathsf{sid})}$ *of every honest party* P_i

2. **Correctness**: *If the honest parties terminate* CRec*, then there exists a unique value* $\bar{s}$ *where* $\bar{s}=s$ *for an honest* D *and* $\bar{s}\in\mathbb{F}\cup\{\perp\}$ *for a corrupt* D*, such that one of the following holds:*

   (a) *Each honest party outputs* $\bar{s}$ *at the end of* CRec*; or*

   (b) *At least* $\frac{\epsilon t^2}{4}(1+2\epsilon)$ *local conflicts occur.*

Proof. The proof for the termination property is exactly the same as in Lemma 3.2 and so we avoid repeating it. We next prove the correctness property for the case when D is *honest*. Note that if D is *honest*, then for every party $P_j\in\mathcal{V}$, the polynomial $\widehat{f}_j(x)$ received by P_j, is the same as t-degree polynomial $f_j(x)$, where $f_j(x)=F(x,j)$ and $F(x,y)$ is the t-degree bivariate polynomial selected by D. To prove the correctness, we need to show that each honest party P_i either reconstructs $\bar{f}_j(x)=f_j(x)$ during CRec for each $P_j\in\mathcal{V}$ or at least $\frac{\epsilon t^2}{4}(1+2\epsilon)$ local conflicts occur. If for each $P_j\in\mathcal{V}$, the honest P_i reconstructs $\bar{f}_j(x)=f_j(x)$ during CRec, then clearly $\overline{F}(x,y)=F(x,y)$ holds and hence $\bar{s}=s$ holds. On the other hand, let there exists some $P_j\in\mathcal{V}$, such that P_i reconstructs $\bar{f}_j(x)\neq f_j(x)$ during CRec. Note that $\bar{f}_j(x)$ is the output of executing RS-Dec $\big($with $N=n-\frac{3t}{2}=\frac{3t}{2}+\epsilon t$ and $c=\frac{2n-5t-2}{4}=\frac{t+2\epsilon t-2}{4}\big)$ on the set $\mathcal{K}_j$. If there are at most $\frac{t+2\epsilon t-2}{4}$ incorrect values in the set $\mathcal{K}_j$ constructed by P_i, then $\bar{f}_j(x)=f_j(x)$ holds. So, for $\bar{f}_j(x)\neq f_j(x)$ to be true, there should be at least $\frac{t+2\epsilon t}{4}$ incorrect values $(k,\widehat{f}_k(j))$ in $\mathcal{K}_j$, such that $\widehat{f}_k(j)\neq f_k(j)$ holds. This implies that at least $\frac{t+2\epsilon t}{4}$ corrupt parties $P_k\in\mathcal{V}_j$ broadcast $\widehat{f}_k(x)$ during CRec, such that $\widehat{f}_k(x)\neq f_k(x)$. We note that if $P_k\in\mathcal{V}_j$ then it implies that $P_k\in\mathcal{V}$ as well because the sub-guards in each sub-guard list is restricted within $\mathcal{V}$. This implies that there exists a set $\mathcal{V}_k$ as well. We claim that if a polynomial $\widehat{f}_k(x)$ is received from the broadcast of P_k during CRec, such that $\widehat{f}_k(x)\neq f_k(x)$, then at least ϵt honest parties $P_l\in\mathcal{V}_k$ will be in local conflict with P_k. It easily follows from this claim that at least $\frac{\epsilon t^2}{4}(1+2\epsilon)$ local conflicts occur since there are at least $\frac{t+2\epsilon t}{4}$ corrupt parties P_k, which reveal $\widehat{f}_k(x)\neq f_k(x)$.

To prove our claim, we note that for every party $P_k\in\mathcal{V}$, the set $\mathcal{V}_k$ has at least $n-2t=t+\epsilon t$ honest parties P_l from $\mathcal{V}$ because $|\mathcal{V}\cap\mathcal{V}_k|\geq n-t$ is ensured. Let $\mathcal{H}_k$ be the set of such honest parties P_l. For each party $P_l\in\mathcal{H}_k$, the condition $f_k(l)=f_l(k)$ holds. This is because P_l is included in $\mathcal{V}_k$ only after P_k broadcasts the message $(\mathtt{ok},l)$, on receiving $f_{lk}=f_l(k)$ from P_l and locally verifying that $f_k(l)=f_l(k)$ holds. Moreover, the values f_{lk} corresponding to the parties $P_l\in\mathcal{H}_k$ uniquely define the t-degree polynomial $f_k(x)$ since $|\mathcal{H}_k|\geq t+\epsilon t$. So, if a polynomial $\widehat{f}_k(x)$ is received from the broadcast of P_k during CRec, such that $\widehat{f}_k(x)\neq f_k(x)$, then there are at most t parties P_l from $\mathcal{H}_k$ for which $\widehat{f}_k(l)=f_{lk}$ holds. This is because two different t-degree polynomials $\widehat{f}_k(x), f_k(x)$ can have at most t common values. Hence there exists at least ϵt parties $P_l\in\mathcal{H}_k$ for which $\widehat{f}_k(l)\neq f_{lk}$ holds. So during SAVSS-MM, such parties P_l will be in local conflict with P_k and include P_k to the set $\mathcal{B}_l$.

Lastly, we need to show the correctness property holds for a *corrupt* D as well. This is, for the most part, similar to the proof of lemma 3.4 for a corrupt D. The proof mainly diverges from lemma 3.4 when incorrect polynomials are reconstructed i.e. $\bar{f}_j(x)\neq\widehat{f}_j(x)$ for some $P_j\in\mathcal{V}$. Whenever this happens, we can show that at least $\frac{\epsilon t^2}{4}(1+2\epsilon)$ local conflicts occur by using similar reasoning as in the case of an honest D. □

We next extend our MWSCC protocol to get the protocol ConstMWSCC for the case $n\geq(3+\epsilon)t$ case.

The ConstMWSCC protocol is exactly the same as MWSCC, except that it invokes (CSh, CRec), instead of (Sh, Rec). It easily follows from the properties of (CSh, CRec) that at least $\frac{\epsilon t^2}{4}(1+2\epsilon)$ local conflicts occur, whenever the common randomness property for the honest parties is violated with probability more than $\frac{3}{4}$. The properties of ConstMWSCC are stated formally in lemma 7.5; we do not give the proof to avoid repetition, as they follow from the proof of the properties of MWSCC.

Lemma 7.5. *Let $n \geq (3+\epsilon)t$. Then for any id $(\mathsf{sid}, \mathsf{r})$ where $\mathsf{sid} \in \mathbb{N}$ and $\mathsf{r} \in \{1,2,3\}$, the following holds during* ConstMWSCC$(\mathsf{sid}, \mathsf{r})$*:*

1. **Correctness***: If all honest parties are able to compute their output, then the following holds:*
   (a) *For every $l \in \{1, \ldots, t+1\}$, all honest parties output $\sigma_l = 0$ and $\sigma_l = 1$ with probability at least 0.139 and 0.63 respectively; otherwise*
   (b) *At least $\frac{\epsilon t^2}{4}(1+2\epsilon)$ local conflicts occur during the protocol.*

We next use the ConstMWSCC protocol to get the extended version of MSCC protocol ConstMSCC. In other words, in ConstMSCC, three instances of ConstMWSCC are invoked. From the properties of ConstMWSCC, we can infer that, in the ConstMSCC protocol, all honest parties have the same common randomness with probability at least 0.25 or at least $\frac{\epsilon t^2}{4}(1+2\epsilon)$ local conflicts occur. A more formal statement of the properties of ConstMSCC is stated in lemma 7.6.

Lemma 7.6. *Let $n \geq (3+\epsilon)t$, where $\epsilon > 0$. Then for any id* sid *where* $\mathsf{sid} \in \mathbb{N}$*, the following holds during* ConstMSCC(sid)*:*

1. **Termination***: If all honest parties participate in* ConstMSCC*, then each honest party eventually terminates.*
2. **Correctness***: One of the following holds:*
   (a) *For every $l \in \{1, \ldots, t+1\}$, for every possible $\sigma_l \in \{0,1\}$, with probability at least 0.25, all honest parties output σ_l as the lth output bit on terminating* ConstMSCC *or*
   (b) *At least $\frac{\epsilon t^2}{4}(1+2\epsilon)$ local conflicts occur during the protocol.*

Finally, we extend MABA to ConstMABA, which is similar to MABA, except that instead of MSCC, the parties invoke ConstMSCC in each iteration and use the output of ConstMSCC to update their values in the same way as mentioned in the MABA protocol. It easily follows from the properties of ConstMSCC that at the end of each iteration, the updated values of all honest parties will be the same with probability at least $\frac{1}{4}$, or other wise $\frac{\epsilon t^2}{4}(1+2\epsilon)$ *new* local conflicts occur. Note that there can be at most $(n-t)t$ different local conflicts that can occur throughout ConstMABA. Hence, there can be at most $\frac{(n-t)t}{\frac{\epsilon t^2}{4}(1+2\epsilon)} = \frac{(2t+\epsilon t)t}{\frac{\epsilon t^2}{4}(1+2\epsilon)} = \frac{4(2+\epsilon)}{\epsilon(1+2\epsilon)} \leq \frac{8}{\epsilon}$ iterations where the updated values of all honest parties will be different with probability more than $\frac{3}{4}$. From this, we can infer that ConstMABA protocol will run for at least $\frac{8}{\epsilon}$ iterations in the worst case. By using similar arguments as used in Lemma 6.11 and Lemma 6.12, we can conclude that the expected running time of ConstMABA is $\mathcal{O}(\frac{1}{\epsilon})$. The properties of ConstMABA (stated in Theorem 7.7) follow using similar arguments as used for MABA and from the properties of ConstMWSCC and ConstMSCC. We do not give the complete proof to avoid repetition.

Theorem 7.7. *Let $n \geq (3+\epsilon)t$, where $\epsilon > 0$. Then protocol* ConstMABA *is an almost-surely terminating ABA protocol for $t+1$ bits with an expected communication complexity of $\mathcal{O}(n^7 \log|\mathbb{F}|)$ bits and expected running time of $\mathsf{R} = \mathcal{O}(\frac{1}{\epsilon})$. In an amortized sense, to agree on a single bit, the protocol requires an expected communication complexity of $\mathcal{O}(n^6 \log|\mathbb{F}|)$ bits.*

Acknowledgement: We would like to thank the anonymous referee of PODC 2018 for their helpful comments.

References

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

[2] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*, volume 19. John Wiley & Sons, 2004.

[3] Z. Beerliová-Trubíniová and M. Hirt. Perfectly-Secure MPC with Linear Communication Complexity. In *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230. Springer Verlag, 2008.

[4] M. Ben-Or. Another Advantage of Free Choice (Extended Abstract): Completely Asynchronous Agreement Protocols. In *PODC*, pages 27–30. ACM, 1983.

[5] M. Ben-Or, E. Pavlov, and V. Vaikuntanathan. Byzantine Agreement in the Full-Information Model in $O(\log n)$ Rounds. In *STOC*, pages 179–186. ACM, 2006.

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

[7] C. Cachin, K. Kursawe, and V. Shoup. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. *J. Cryptology*, 18(3):219–246, 2005.

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

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

[10] B. Chor and B. A. Coan. A Simple and Efficient Randomized Byzantine Agreement Algorithm. *IEEE Trans. Software Eng.*, 11(6):531–539, 1985.

[11] R. Cramer and I. Damgård. *Multiparty Computation, an Introduction. Contemporary Cryptography*. Birkhåuser Basel, 2005.

[12] I. Damgård and J. B. Nielsen. Scalable and Unconditionally Secure Multiparty Computation. In *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer Verlag, 2007.

[13] Paul Feldman and Silvio Micali. Optimal Algorithms for Byzantine Agreement. In *STOC*, pages 148–161. ACM, 1988.

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

[15] Matthias Fitzi. *Generalized Communication and Security Models in Byzantine Agreement*. PhD thesis, 2002.

[16] V. King and J. Saia. Byzantine Agreement in Expected Polynomial Time. *J. ACM*, 63(2):13:1–13:21, 2016.

[17] Nancy A Lynch. *Distributed algorithms*. Morgan Kaufmann, 1996.

[18] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes*. North-Holland Publishing Company, 1978.

[19] A. Patra, A. Choudhury, and C. Pandu Rangan. Asynchronous Byzantine Agreement with Optimal Resilience. *Distributed Computing*, 27(2):111–146, 2014.

[20] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[21] Michael O. Rabin. Randomized Byzantine Generals. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 403–409, 1983.

[22] C. Wang. Asynchronous Byzantine Agreement with Optimal Resilience and Linear Complexity. *CoRR*, abs/1507.06165, 2015.

A Analysis of the ABA Protocols of [1] and [22]

We provide an analysis of the expected communication complexity, expected running time and local computation of the ABA protocols of [1] and [22] in this section. For the protocols in both [1] and [22], all communication and computation are done over a finite field $\mathbb{F}$, where $|\mathbb{F}| > n$.

A.1 Analysis of [1]

In [1], the general framework described in [8] is followed, where an SAVSS is used, instead of AVSS and SCC is used in place of a CC protocol. Further, a moderated weak SVSS (MW-SVSS), which is a weaker version of the SAVSS, has been constructed and used as a building block for the SAVSS protocol. The MW-SVSS protocol comprises of two sub-protocols: Share and Reconstruct. In the Share protocol of MW-SVSS, the dealer selects $\mathcal{O}(n)$ t-degree polynomials and sends a polynomial to each party, which involves a communication of $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits. Having received these polynomials from the dealer, the parties carry out a pairwise consistency check of their received polynomials, which also involves a communication of $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits. Based on these pairwise checking, each party broadcasts the identities of $n - t$ parties, whose polynomials are pair-wise consistent with it. This involves a communication of $\mathcal{BC}(\mathcal{O}(n^2 \log n))$ bits, as the identity of each party can be represented by $\log n$ bits. Since broadcast of a single bit requires a communication of $\mathcal{O}(n^2)$ bits over the point-to-point channels, $\mathcal{BC}(\mathcal{O}(n^2 \log n))$ bits translates to a point-to-point communication of $\mathcal{O}(n^4 \log n)$ bits. During the reconstruction phase, each party ends up broadcasting its entire t-degree polynomial, incurring a total communication of $\mathcal{BC}(\mathcal{O}(n^2 \log |\mathbb{F}|))$ bits or a point-to-point communication of $\mathcal{O}(n^4 \log |\mathbb{F}|)$ bits. Overall, the sharing protocol has a communication complexity of $\mathcal{O}(n^4 \log |\mathbb{F}|)$ bits (since $|\mathbb{F}| > n$, we have $\mathcal{O}(\log n) = \mathcal{O}(\log |\mathbb{F}|)$). And the reconstruction protocol has a communication complexity of $\mathcal{O}(n^4 \log |\mathbb{F}|)$ bits.

Now, let us analyze the SAVSS protocol. Similar to the Sharing protocol of MW-SVSS, the dealer sends a t-degree polynomial to each of the n parties, incurring a total communication of $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits. Then, each party acts as a dealer and invokes $n-1$ instances of MW-SVSS Share protocol, one with respect to every other party acting as a moderator. This implies that $\mathcal{O}(n^2)$ MW-SVSS Share instances are invoked. So, the SAVSS protocol sharing protocol has communication complexity of $\mathcal{O}(n^6 \log |\mathbb{F}|)$ bits. The reconstruction phase of SAVSS involves $\mathcal{O}(n^2)$ instances of MW-SVSS reconstruction protocol and hence has communication complexity of $\mathcal{O}(n^6 \log |\mathbb{F}|)$ bits.

In SCC, each party acts as a dealer and shares n secrets using the SAVSS share protocol and later these values are reconstructed. Hence $\mathcal{O}(n^2)$ sharing and reconstruction instances are involved. As a result, one instance of SCC has communication complexity of $\mathcal{O}(n^8 \log |\mathbb{F}|)$ bits. The ABA protocol invokes the SCC

protocol in each iteration, in a similar manner as [8]. It is to be noted that whenever the correctness property of SCC is violated, then at least *one* local conflict occurs. Since there are at most t corrupt parties, there can be at most $(n-t)t = \mathcal{O}(n^2)$ iterations in which correctness is violated. In all other iterations, the correctness property of SCC holds and only expected constant number of such iterations are needed for the ABA to terminate. Hence, the ABA protocol requires an expected running time of $\mathsf{R} = \mathcal{O}(n^2)$. This implies that the expected number of SCC instances invoked inside the ABA protocol is $\mathcal{O}(n^2)$. Hence the ABA protocol has an expected communication complexity of $\mathcal{O}(n^{10} \log |\mathbb{F}|)$ bits. The local computation required by each party in each iteration is polynomial, as the underlying SAVSS and SCC instances require polynomial amount of local computation.

A.2 Analysis of [22]

The ABA protocol presented in [22] also follows the general framework of obtaining an ABA from an AVSS described in [8]. Instead of using an AVSS, a variant of AVSS known as *Inferable Verifiable Secret Sharing* (IVSS) is used. Property-wise, IVSS is similar to SAVSS except that instead of *local* conflicts, the parties *globally* identify *faulty pairs* of the form (P_i, P_j), if correctness is violated. It is guaranteed that at least one party in every globally identified faulty pair (P_i, P_j) is corrupt. Just like an AVSS, the IVSS comprises of a sharing protocol and a reconstruction protocol. In the sharing protocol, the dealer picks a t-degree symmetric bivariate polynomial and sends a t-degree univariate polynomial, that is related to the bivariate polynomial to each party. This requires a total communication of $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits. Then, the parties perform pair-wise consistency check of the received univariate polynomials, which requires a communication of $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits. For each party P_j, whose polynomial is pair-wise consistent with the polynomial of P_i, the party P_i broadcasts the message "Equal(i, j)", which of size $\mathcal{O}(\log |\mathbb{F}|)$ bits. Since this is done for all pairs of parties, this step has communication complexity of $\mathcal{BC}(\mathcal{O}(n^2 \log n))$ bits, where the identity of each party can be represented by $\log n$ bits. Based on these messages, dealer creates a *consistency graph* with $P_1, \ldots, P_n$ being the vertices and where the edge between P_i and P_j implies that P_i and P_j have broadcasted Equal(i, j) and Equal(j, i) respectively. Based on the consistency graph, dealer tries to find a set $\mathcal{M}$ of $n-t$ parties, such the univariate polynomials of *all* the parties in $\mathcal{M}$ are mutually pair-wise consistent. This is equivalent to finding a *clique* of size $n-t$ in the consistency graph and this requires *exponential* amount of computation from the dealer. Once dealer finds $\mathcal{M}$, it broadcasts it, which has communication complexity of $\mathcal{BC}(\mathcal{O}(n \log n))$ bits. On receiving $\mathcal{M}$, the parties themselves locally construct the consistency graph and verify if $\mathcal{M}$ is a clique of size $n-t$ in the consistency graph. Overall, the sharing protocol requires $\mathcal{BC}(\mathcal{O}(n^2 \log |\mathbb{F}|))$ bits (since $|\mathbb{F}| > n$, we have that $\mathcal{O}(\log |\mathbb{F}|) = \mathcal{O}(\log n)$). During the reconstruction protocol, each party in $\mathcal{M}$ broadcasts its univariate polynomial and this requires a communication complexity of $\mathcal{BC}(\mathcal{O}(n^2 \log |\mathbb{F}|))$ bits. By replacing the communication required for each $\mathcal{BC}$ instance as $\mathcal{O}(n^2)$, we get the overall communication complexity of the sharing protocol to be $\mathcal{O}(n^4 \log |\mathbb{F}|)$ bits and that of the reconstruction protocol is $\mathcal{O}(n^4 \log |\mathbb{F}|)$ bits as well.

A variant of the CC protocol called *Inferable Common Coin* (ICC) was presented in [22]. The construction of ICC from an IVSS is similar to that of CC from AVSS [8]. The ICC invokes $\mathcal{O}(n^2)$ instances of IVSS sharing and reconstruction and hence has communication complexity of $\mathcal{O}(n^6 \log |\mathbb{F}|)$ bits, with the parties performing exponential amount of local computation. The ICC ensures that if the common randomness property (correctness) gets violated, then $\Omega(n)$ faulty pairs are globally inferred. The ABA protocol invokes an instance of ICC in each iteration. Since there can be $\mathcal{O}(n^2)$ faulty pairs, the number of iterations in which the correctness property is violated is $\mathcal{O}(n)$. For the ABA protocol to terminate, an expected constant number of iterations in which the correctness of ICC is not violated are needed. This implies that the overall expected running time of the ABA protocol is $\mathsf{R} = \mathcal{O}(n)$. From this, we gather that the expected communication complexity of the ABA protocol is $\mathcal{O}(n^7 \log |\mathbb{F}|)$ bits, with the parties performing exponential amount of computation.