An Almost-Surely Terminating Polynomial Protocol for Asynchronous Byzantine Agreement with Optimal Resilience

Ittai Abraham
Hebrew University
ittaia@cs.huji.ac.il

Danny Dolev*
Hebrew University
dolev@cs.huji.ac.il

Joseph Y. Halpern†
Cornell University
halpern@cs.cornell.edu

ABSTRACT

Consider an asynchronous system with private channels and n processes, up to t of which may be faulty. We settle a long-standing open question by providing a Byzantine agreement protocol that simultaneously achieves three properties:

1. *(optimal) resilience*: it works as long as $n > 3t$;
2. *(almost-sure) termination*: with probability one, all nonfaulty processes terminate;
3. *(polynomial) efficiency*: the expected computation time, memory consumption, message size, and number of messages sent are all polynomial in n.

Earlier protocols have achieved only two of these three properties. In particular, the protocol of Bracha is not polynomially efficient, the protocol of Feldman and Micali is not optimally resilient, and the protocol of Canetti and Rabin does not have almost-sure termination. Our protocol utilizes a new primitive called *shunning (asynchronous) verifiable secret sharing* (SVSS), which ensures, roughly speaking, that either a secret is successfully shared or a new faulty process is ignored from this point onwards by some nonfaulty process.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed SystemsF.0 [Theory of Computation]: General.

General Terms

Security, Theory

*Part of the work was done while the author visited Cornell university. The work was funded in part by ISF, NSF, CCR, and AFOSR.

†Supported in part by NSF under grants ITR-0325453 and IIS-0534064, and by AFOSR under grant FA9550-05-1-0055.

PODC'08, August 18–21, 2008, Toronto, Ontario, Canada.

Keywords

Distributed computing, secret sharing, Byzantine agreement.

1. INTRODUCTION

The *Byzantine agreement problem*, introduced in 1980 by Pease, Shostak, and Lamport [13], has emerged as one of the most fundamental problems in Distributed Computing. The problem is easy to describe: each process has an input value; the goal is for all processes to agree on a consensus value that is an input value of one of the processes. The challenge lies in reaching agreement despite the presence of faulty processes. Many variants of the problem have been studied. After three decades of extensive research, tight bounds have been obtained for almost all of the variants, with one significant exception: *asynchronous Byzantine agreement*, where communication channels between processes have unbounded delay (although messages are guaranteed to arrive eventually), and the faulty processes are malicious in an arbitrary way (though nonfaulty processes have secure private channels).

An execution of a Byzantine agreement protocol is *nonterminating* if some nonfaulty process does not output a value. The celebrated result of Fischer, Lynch, and Paterson [11] shows that any protocol that never reaches disagreement (i.e., has no executions where two nonfaulty processes output different values) must have some nonterminating executions. For a protocol that never reaches disagreement, the best we can hope for is that the set of nonterminating executions has probability 0. We say such protocols are *almost-surely terminating*. Ben-Or [1] showed that almost-surely terminating asynchronous Byzantine agreement can be achieved as long as $n > 5t$, where n is the number of processes in the system and t is a bound on the number of faulty processes. However, his protocol required an expected number of rounds that is exponential in n. This started a lengthy sequence of research on asynchronous Byzantine agreement; see, for example, [1, 3, 4, 9, 12, 14]. It is well known that Byzantine agreement for n processes cannot be reached if $n \leq 3t$ [13]. Therefore the best resilience one can hope for is $n > 3t$. We will call such protocols *optimally resilient*. Bracha [3] provides an almost-surely terminating protocol that is optimally resilient. However his protocol does not scale well with the size of the system, since, like Ben-Or's, the expected number of messages and rounds is exponential in n. Feldman and Micali [9] provide a Byzantine agreement protocol for the synchronous model with optimal resilience and constant expected running time. They extend their result to the asynchronous model, where they provide a polynomial-time algorithm that almost-surely terminates, but does not have optimal resilience; their protocol requires that $n > 4t$. Canetti and Rabin [4, 5] provide a protocol that is optimally resilient ($n > 3t$) and polynomially efficient. Their result uses ideas from Rabin and Ben-Or [14] on verifiable secret sharing (VSS) in synchronous systems equipped with a broadcast channel. The techniques of [14] have an inherent nonzero probability of failure; as a result, in the asynchronous implementation of [4], the protocol is not almost-surely terminating. Indeed, in [5], the authors explicitly highlighted the problem of finding a protocol that simultaneously achieves optimal resilience, almost-sure termination, and polynomial efficiency. Up to now, despite repeated efforts, this has not been done. The main result of this paper is to provide such a protocol.

Pretty much all protocols following Bracha's [3] used his idea of reducing the problem of Byzantine agreement to that of implementing a shared coin. We do that as well. We obtain a shared coin using an approach that goes back to Feldman and Micali [9, 10], who essentially reduce the problem of efficiently implementing a shared coin to that of efficiently implementing VSS. Roughly speaking, the secrets in VSS are used to generate a shared coin. We refer the reader to Canetti's thesis [6] (Chapters 4 and 5) for a comprehensive account of the rather complex reduction from VSS to Byzantine agreement in the asynchronous model.

The protocol of Canetti and Rabin [4] also uses the reduction from verifiable secret sharing to Byzantine agreement. The only reason that their protocol is not almost-surely terminating is that they use a protocol that they call Asynchronous Verifiable Secret Sharing (AVSS), which has a small (but nonzero) probability of not terminating. Our protocol has essentially the same structure as the Canetti-Rabin protocol. Indeed, it uses the same reduction from AVSS to Byzantine agreement as in [4], except that the use of AVSS is replaced by a variant of AVSS that we call *shunning (asynchronous) verifiable secret sharing* (SVSS). which is guaranteed to terminate almost-surely.

To explain the properties of SVSS, we first review the properties of standard VSS (verifiable secret sharing). VSS involves a dealer who has a value to share, which we think of as the dealer's secret. It has two key properties, known as *validity* and *binding*. Informally, the validity property guarantees that, if the dealer is nonfaulty, then all nonfaulty processes will reconstruct the dealer's value; the binding property guarantees that a faulty dealer must commit to a value during what is called the *share phase* of the protocol. Our SVSS scheme has weaker validity and binding properties. Specifically, we require that in each invocation where the validity or binding properties do not hold, at least one nonfaulty process ignores at least one new faulty process from that invocation on. The key observation is that this limits the adversary to breaking the validity and binding properties at most a polynomial number of times.

The SVSS protocol uses a weaker protocol called *moderated weak shunning (asynchronous) VSS* (MW-SVSS). The MW-SVSS protocol is a variant of VSS with a dealer and an additional entity called a *moderator*. In MW-SVSS the dealer has some input value s and the moderator has some input value s'. The nonfaluty moderator's task is to enforce during the share phase that the value that the dealer shares is s' (hence $s = s'$ if both are nonfaulty). The initials MWS characterize how MW-SVSS differs from standard VSS:

- **Moderated**. A faulty dealer must commit to the value of the nonfaulty moderator in order to complete the share protocol. (Katz and Koo [12] use a moderator for VSS in a somewhat similar way.)
- **Weak**. As in *weak VSS* [4, 14], the binding property of VSS is weakened so that each process reconstructs either the committed value or a default value (denoted $\perp$).
- **Shunning**. Like SVSS, it is possible that neither validity nor the weaker binding property hold, but in that case at least one nonfaulty process ignores at least one new faulty process from this stage on.

As in the VSS scheme used in [2, 9, 10], the SVSS scheme starts with a dealer who shares a degree-t bivariate polynomial $f(x, y)$ such that $f(0, 0)$ is the secret. Each process i gets $t+1$ values of each of the polynomials $g(y) = f(i, y)$ and $h(x) = f(x, i)$, which is enough to reconstruct them, since they both have degree t. Then, roughly speaking, each pair (i, j) of processes uses MW-SVSS to commit to $f(i, j)$ and $f(j, i)$. This ensures that if either i or j is nonfaulty then the reconstructed values of the MW-SVSS protocol will be either $\perp$ or the required values $(f(i, j), f(j, i))$. We then use this fact to prove the properties of the SVSS protocol.

The key property of the MW-SVSS protocol is its use of a fault-detection mechanism. The mechanism has the property that a nonfaulty process might not explicitly know it has detected a faulty process. The only guarantee is that it will act as if it has detected a faulty process, by ignoring all messages from the detected process for the rest of the protocol. This behavior is somewhat reminiscent of the failure detector $\diamond\mathcal{W}$ [7] in the sense that a nonfaulty process might reach a state of permanently suspecting a faulty process without being explicitly aware of this fact. Since the details of the MW-SVSS protocol are somewhat technical, we refer the reader to Section 3.1 for a high-level description.

The rest of this paper is organized as follows. In Section 2, we state the properties of SVSS and MW-SVSS. In Section 3, we provide an implementation of MW-SVSS and prove that it has the required properties. In Section 4, we do the same for SVSS, using MW-SVSS as a subroutine. A description of Bracha's *Reliable Broadcast* protocol, which we use as a subroutine, is given in the appendix.

2. SHUNNING VSS

As we mentioned above, in SVSS, if either the binding property or the validity property does not hold, then a new faulty process is ignored in all future invocations by some nonfaulty process. To implement this, each process needs to keep track of the processes it knows to be faulty. Thus, the SVSS scheme actually has two components: a *detection and message management protocol* (DMM protocol) and a *VSS protocol*. Each process uses its DMM protocol to decide which messages to discard, which to ignore for now, and which to act on, and to keep track of the processes it knows to be faulty. The DMM protocol is invoked when the SVSS scheme is initialized, and then runs indefinitely and concurrently with all the invocations of the VSS protocols. The VSS protocol may be invoked a number of times while the SVSS scheme runs, and several invocations may be running concurrently. The VSS protocol is composed of a pair of protocols $\mathcal{S}$ (for *share*) and $\mathcal{R}$ (for *reconstruct*). These protocols are called separately; $\mathcal{R}$ is never called unless $\mathcal{S}$ completes, but $\mathcal{R}$ may not be called at all even if $\mathcal{S}$ completes. We associate with each VSS invocation a unique session identifier (c,i) that is composed of a counter c and the dealer's identifier i. We tag all events of that invocation with its session identifier, so that it is always clear which invocation of the VSS protocol an event belongs to.

We say that a VSS invocation has *completed for process* j if process j completed the reconstruct associated with that session. Given a process j and two VSS invocations with session identifiers (c,i) and (c',i'), we write $(c,i) \rightarrow_j (c',i')$ if process j completes the invocation of the VSS (c,i) before process j begins the invocation of the VSS (c',i').

As we said in the introduction, our VSS scheme is *shunning*. Process i may start shunning j well before i is sure that j is faulty; indeed, i may shun j without ever knowing that j is faulty.

Definition 1. *Process j is* shunned *by process i starting in session (c,l) of MW-SVSS (resp., SVSS) if process i does not ignore some message from j during session (c,l), but ignores or discards all messages from j associated with every session (c',l') of MW-SVSS (resp., SVSS) such that $(c,l) \rightarrow_i (c',l')$.*

2.1 Properties of SVSS

Each VSS invocation has one process d designated as the *dealer*; the dealer has some input value s. For ease of exposition, we do not include the session identifier in our description of the properties of the VSS protocol when they are clear from the context, although we do include them in the description of the protocols. Each VSS invocation must satisfy the following properties (in runs with at most t faulty processes); we call these the *SVSS* properties.

1. **Validity of Termination**. If a nonfaulty dealer initiates protocol $\mathcal{S}$, then each nonfaulty process will eventually complete protocol $\mathcal{S}$.
2. **Termination**. If a nonfaulty process completes protocol $\mathcal{S}$, then all nonfaulty processes will eventually complete protocol $\mathcal{S}$. Moreover, if all nonfaulty processes begin protocol $\mathcal{R}$, then all nonfaulty processes will eventually complete protocol $\mathcal{R}$ (note, however, that if only some but not all nonfaulty processes begin protocol $\mathcal{R}$, then there is no termination requirement).
3. **Binding**. Once the first nonfaulty process completes an invocation of $\mathcal{S}$ with session id (c,d), there is a value r such that either
   - the output of each nonfaulty process that completes protocol $\mathcal{R}$ is r; or
   - there exists a nonfaulty process i and a faulty process j such that j is shunned by i starting in session (c,d).
4. **Validity**. If the dealer is nonfaulty, then either
   - the output of each nonfaulty process that completes protocol $\mathcal{R}$ is s; or
   - there exists a nonfaulty process i and a faulty process j such that j is shunned by i starting in session (c,d).
5. **Hiding**. If the dealer is nonfaulty and no nonfaulty process invokes protocol $\mathcal{R}$, then the faulty processes learn nothing about the dealer's value.[1]

2.2 Properties of MW-SVSS

In order to implement the VSS protocol, we use a weaker protocol called *moderated weak shunning (asynchronous) VSS* (MW-SVSS). Just as VSS, the MW-SVSS protocol is composed of a share protocol $\mathcal{S}'$ and a reconstruction protocol $\mathcal{R}'$. As in weak VSS, we weaken the Binding property so that each nonfaulty process reconstructs either r or $\perp$. But now, in addition to having one process d designated as the dealer, there is an additional process designated as the *moderator*. Both the dealer and the moderator have (possibly different) input values, denoted s and s', respectively. Each MW-SVSS invocation must satisfy Termination and Validity, just like VSS, and the following variants of the properties of VSS (in runs with at most t faulty processes); we call these the *MW-SVSS* properties.

1′. **Moderated Validity of Termination**. If a nonfaulty dealer initiates protocol $\mathcal{S}'$, the moderator is nonfaulty, and $s = s'$, then each nonfaulty process will eventually complete protocol $\mathcal{S}'$.

3′. **Weak and Moderated Binding**. Once the first nonfaulty process completes an invocation of protocol $\mathcal{S}'$ with session id (c,d), there is a value r (possibly $\perp$) such that

   - if the moderator is nonfaulty, then $r = s'$.

   In addition, either

   - the output of each nonfaulty process that completes protocol $\mathcal{R}'$ is either r or $\perp$; or
   - there exists a nonfaulty process i and a faulty process j such that j is shunned by i starting in session (c,d).

5′. **Moderated Hiding**. If the dealer and moderator are nonfaulty and no nonfaulty process invokes protocol $\mathcal{R}'$, then the faulty processes learn nothing about the dealer's value.

It might seem surprising that in the second condition of Validity and (Weak and Moderated) Binding, we talk about shunning rather than just saying that a faulty process is detected. The reason is that, as we show in Example 1 (after we give the implementation of the MW-SVSS protocol), it is possible that two nonfaulty processes will complete an invocation of the MW-SVSS protocol with different values without (explicitly) detecting a new faulty process; however, in that case, at least one of them will shun a faulty process that was not shunned before.

3. IMPLEMENTING DMM AND MW-SVSS

3.1 A high-level description

In this section, we provide an implementation of DMM and MW-SVSS. We start with a high-level description of both. Both protocols use the Reliable Broadcast protocol (RB) of Bracha [3]. RB guarantees that messages are indeed broadcast; if a nonfaulty sender sends a message m, then all

[1] To make this precise, assume that the adversary determines the *scheduling protocol*: how long each message will take to arrive as a function of the history. Note that once we fix the inputs, the faulty processes, the protocols used by the faulty processes, and the scheduling protocol, the VSS protocol (which is used by the nonfaulty processes) determines a distribution on runs. Formally, hiding requires that for all distributions determined this way, the dealer's value is independent of the histories of the faulty processes.

nonfaulty processes eventually receive m, and nothing else. (The properties of RB are stated carefully in the appendix, where, for completeness, an implementation is provided.)

We assume that the dealer creates $n+1$ degree-t polynomials $f, f_1, \ldots, f_n$ over some finite field F with $|F| > n$ such that $f(0)$ is the secret (i.e., $f(0) = s$) and $f_l(0) = f(l)$. Then the dealer shares the polynomials $f_1, \ldots, f_n$ and also gives each process j the polynomial f_j. We can think of process j as a potential "monitor" for f_j. The dealer shares the polynomial f_j by sending each process k the value $f_j(k)$. This means that, if the dealer is correct, any $t+1$ nonfaulty processes can reconstruct f_j. In addition, the dealer sends f to the moderator. Each process k that receives $f_j(k)$ sends this value to j and broadcasts a confirmation. In this case, we can think of process k as a "confirmer" for $f_j(k)$. When j receives confirmations and values that agree with the polynomial f_j sent by the dealer from at least $n-t$ processes, j becomes a "monitor" for f_j, sends $f_j(0)$ to the moderator, and broadcasts the set L_j of at least $n-t$ confirmers whose value it accepted. Intuitively, each monitor j is responsible for validating the value of one point on the polynomial f, namely, $f(j) = f_j(0)$. When the moderator receives at least $n-t$ values all of which agree with the polynomial f from different monitors and receives confirmations from their associated L_j sets, then the moderator broadcasts the set of $n-t$ monitors' indexes it accepted. The dealer broadcasts a confirmation when it learns that the moderator, its monitors, and their confirmers have acted in a nonfaulty manner. This allows nonfaulty processes to know which confirmers they need to wait for in order to complete their execution of the share protocol.

In the reconstruct phase, processes send their values using the RB protocol. If the dealer is nonfaulty, then it can check the values sent by all processes and detect faulty processes. If the dealer is faulty, then there are at least $t+1$ nonfaulty monitors l that can monitor their polynomial f_l. If they do not detect problems with their confirmers, then the Weak Binding property must hold.

We now explain how processes shun other processes if a problem is detected. Before a process i "sees" a message in the MW-SVSS protocol (or the SVSS protocol that we present later), the message is filtered by the DMM protocol. The DMM$_i$ protocol decides whether to discard the message, ignore it for now, or pass it on for action. In order to do this, DMM$_i$ must maintain a number of data structures. First, it maintains the partial order $\rightarrow_i$ on sessions described above, where $(c_1, j_1) \rightarrow_i (c_2, j_2)$ if i started the share protocol of VSS session (c_2, j_2) after completing the reconstruct protocol of VSS session (c_1, j_1). In addition, the DMM$_i$ protocol uses a variable D_i that represents a set of processes. Intuitively, the processes in D_i are ones known by i to be faulty. Any message sent by a process $j \in D_i$ is discarded by i. To decide which messages to ignore for now and which to pass on for action, DMM$_i$ maintains two arrays. The first array, denoted ACK$_i$, consists of tuples in $\{1, \ldots, n\} \times \{1, \ldots, n\} \times I\!N \times F$. Intuitively, $(j, l, c, x) \in$ ACK$_i$ if i is expecting to receive a broadcast sent by j using RB saying $f_l(j) = x$ as part of a VSS session (c, i) (thus, this is a session for which i is the dealer). The second array, denoted DEAL$_i$, consists of tuples in $\{1, \ldots, n\} \times I\!N \times \{1, \ldots, n\} \times F$. Intuitively, $(j, c, l, x) \in$ DEAL$_i$ if i is expecting to receive a message broadcast by j (using RB) saying $f_i(j) = x$ as part of VSS session (c, l). Both ACK$_i$ and DEAL$_i$ are initially empty. We will explain how tuples are added to ACK$_i$ and DEAL$_i$ when we describe the MW-SVSS protocol.

Process i ignores (that is, saves but does not act on) all messages from process j that are part of a session (c', k) such that either $(j, l, c, s) \in$ ACK$_i$ and $(c, i) \rightarrow_i (c', k)$ or $(j, c, l, s) \in$ DEAL$_i$ and $(c, l) \rightarrow (c', k)$. That is, newer messages from j are ignored by i if i is expecting to receive something from j that it has not yet received. When a message that i expects to hear from j that is associated with either with $(j, l, c, s) \in$ ACK$_i$ or with $(j, c, l, s) \in$ DEAL$_i$, then the relevant tuple is removed from ACK$_i$ or DEAL$_i$. Once there are no messages that i expects to hear from j from a session that precedes (c', k), then the DMM$_i$ protocol enables the MW-SVSS protocol to act on messages from session (c', k).

Finally, process j is added to D_i if a message is received from j that is inconsistent with what is expected according to a tuple in ACK$_i$ or DEAL$_i$. For example, if $(j, l, c, s) \in$ ACK$_i$ and i receives a message as part of session (c, i) from j saying $f_l(j) = s'$, with $s' \neq s$, then j is added to D_i, and messages sent by j in all sessions (c', k) such that $(c, l) \rightarrow_i (c', k)$ will be discarded by i.

3.2 Implementing MW-SVSS

We now show how to implement MW-SVSS. We start with the share protocol $\mathcal{S}'$. We assume that the field F being used is common knowledge and $|F| > n$. In the $\mathcal{S}'$ protocol (and a number of our later protocols), we have variables that are tagged by the session id (c, d). If the session id is clear from context, we omit it.

Share protocol $\mathcal{S}'$:

1. If a dealer i wants to invoke $\mathcal{S}'$ with a secret s it first updates c to $c+1$ and then selects $n+1$ random degree-t polynomials $f(x), f_1(x), \ldots, f_n(x)$ over field F such that $f(0) = s$ and $f_l(0) = f(l)$ for all $l \in \{1, \ldots, n\}$. It sends each process j a message $f_1(j), \ldots, f_n(j), (c, i)$. In addition, it sends each process l a message $f_l(1), \ldots, f_l(t+1), (c, i)$ (note that this allows l to compute f_l, so we sometimes say "l receives f_l" in this message), and sends the moderator yet another message, $f(1), \ldots, f(t+1), (c, i)$ (so that the moderator can compute f).

2. If process j receives values $\hat{f}^j_1, \ldots, \hat{f}^j_n$ and polynomial $\hat{f}_j$ from a dealer i in session (c, i), then, for each process l, j sends $\hat{f}^j_l, (c, i)$ to l. (Note that $\hat{f}^j_k$ is supposed to be $f_k(j)$, but if the dealer is faulty, it may not be. We continue to use the notation $\hat{f}$ and $\hat{f}^j_k$ to denote the polynomials and values actually received.) It also broadcasts $ack, (c, i)$ to all processes using RB.

3. If process j receives $\hat{f}^l_j, (c, i)$ and $ack, (c, i)$ from process l, receives $\hat{f}_j, (c, i)$ from the dealer i, and $\hat{f}^l_j = \hat{f}_j(l)$, it adds $(l, c, i, \hat{f}_j(l))$ to DEAL$_j$. Intuitively, the message $\hat{f}^l_j = \hat{f}_j(l)$ provides confirmation to j that the dealer sent $f_l(j)$ to both j and l. The fact that j adds $(l, c, i, \hat{f}^l_j)$ to DEAL$_j$ means that j expects l to confirm publicly (using RB) that indeed it received $\hat{f}^l_j$ from the dealer i, which is what l told j privately.

4. Let $L_{j,(c,i)} = \{l : (l, c, i, \hat{f}_j(l)) \in \text{DEAL}_j\}$. If $|L_j| \geq n-t$, then j sends $L_j, (c, i)$ to all processes using RB, It

also sends $\hat{f}_j(0), (c,i)$ to the moderator. Intuitively, if $|L_j| \geq n-t$, then j has gotten as much confirmation as it can expect to get that the dealer i correctly shared the polynomial f_j. By broadcasting L_j, it is broadcasting the set of processes from which it expects to hear public confirmation of this fact. By sending $\hat{f}_j(0)$ to the moderator, j is giving the moderator a share of the information that the moderator needs for computing the secret.

5. If the moderator receives $\hat{f}, (c,i)$ from the dealer, $\hat{f}_0^j, (c,i)$ and $\hat{L}_j, (c,i)$ from process j, and $ack, (c,i)$ message from all processes $l \in \hat{L}_j$, $\hat{f}_0^j = \hat{f}(j)$, and $\hat{f}(0) = s'$, the moderator adds j to the set $M_{(c,i)}$, which is initialized to $\emptyset$. Intuitively, if the values that the moderator receives from j are compatible with the values the moderator received from the dealer, and the dealer's values are compatible with the moderator's value s', then the moderator adds j for the session (c,i) to M.

6. If $|M_{(c,i)}| \geq n-t$, the moderator sends $M_{(c,i)}, (c,i)$ to all processes using RB.

7. If the dealer i receives $\hat{M}, (c,i)$ from the moderator, receives $\hat{L}_j, (c,i)$ from each process $j \in \hat{M}$, and receives $ack, (c,i)$ from each process $l \in \hat{L}_j$ such that $j \in \hat{M}$, then it adds $(l, j, c, f_j(l))$ to ACK_i for all $j \in \hat{M}$ and $l \in \hat{L}_j$, and sends $OK, (c,i)$ using RB. Note that if the moderator is nonfaulty and it sends these messages to the dealer, then it really did receive $\hat{L}_j, (c,i)$ from each process $j \in \hat{M}$ and $ack, (c,i)$ from each process l in $\hat{L}_j$, and these messages were sent using RB. Thus, the dealer will eventually receive all these messages too and, if nonfaulty, will broadcast the OK message.

8. If process j receives $\hat{M}, (c,i)$ from the moderator and $j \notin \hat{M}$ then j removes from DEAL_j all entries of the form $(\cdot, c, i, \cdot)$ that are associated with session (c,i). Intuitively, since $j \notin \hat{M}$ for session (c,i), we do not care about the values of f_j for this session.

9. If process j receives an $OK, (c,i)$ message from the dealer, $\hat{M}, (c,i)$ from the moderator, $\hat{L}_l, (c,i)$ from each process $l \in \hat{M}$, and $ack, (c,i)$ from each $k \in \hat{L}_l$ such that $l \in \hat{M}$, it completes this invocation of the share protocol $\mathcal{S}'$.

Reconstruct protocol $\mathcal{R}'$:

1. If process $j \in \hat{L}_l$ for $l \in \hat{M}$, then j broadcasts $l, \hat{f}_l^j, (c,i)$ using RB, where $\hat{f}_l^j$ is what j received from the dealer at step 2 of $\mathcal{S}'$.

2. Process j initializes $K_{j,l,(c,i)}$ to $\emptyset$ for each process l for which it has received a set L_l. If j receives a message $l, \bar{f}_l^k, (c,i)$ from process k at step 1, and $k \in \hat{L}_l$, then j adds $(l, \bar{f}_l^k)$ to $K_{j,l}$. Intuitively, $(l, \bar{f}_l^k)$ should be the point $(k, f_l(k))$ on the polynomial f_k.

3. If $|K_{j,l}| = t+1$, then j finds the unique degree t polynomial $\bar{f}_l$ that interpolates the points in $|K_{j,l}|$.

4. After computing $\bar{f}_l$ for all $l \in \hat{M}$, j tries to interpolate a polynomial $\bar{f}$ such that $\bar{f}(l) = \bar{f}_l(0)$ for all $l \in \hat{M}$. If $\bar{f}$ exists, j outputs $\bar{f}(0)$; otherwise, j outputs $\perp$.

3.3 Implementing DMM

We now describe the implementation of DMM_i.

Protocol DMM_i

1. Initialize an empty set of processes D_i, an empty array ACK_i consisting of tuples in $\{1,\ldots,n\} \times \{1,\ldots,n\} \times \mathbb{N} \times F$, and an empty array DEAL_i consisting of tuples in $\{1,\ldots,n\} \times \mathbb{N} \times \{1,\ldots,n\} \times F$. As we said earlier, intuitively, $(j,l,c,x) \in \mathrm{ACK}_i$ if i is expecting to receive a broadcast sent by j using RB saying $f_l(j) = x$ as part of VSS session (c,i) and $(j,c,l,x) \in \mathrm{DEAL}_i$ if i is expecting to receive a message broadcast by j using RB saying $f_i(j) = x$ as part of VSS session (c,l).
2. If $(j,l,c,x) \in \mathrm{ACK}_i$ and a broadcast message $x', j, (c,i)$ is received then
   - if $x = x'$, then remove (j,l,c,x) from ACK_i;
   - otherwise, add j to D_i.

   (See line 7 of protocol $\mathcal{S}'$ for the condition that causes a tuple (j,l,c,x) to be added to ACK_i.)
3. If $(j,c,l,x) \in \mathrm{DEAL}_i$ and a broadcast message $x', i, (c,j)$ is received, then
   - if $x = x'$ then remove (j,c,l,x) from DEAL_i;
   - otherwise, add j to D_i.

   (See line 3 of protocol $\mathcal{S}'$ for the condition that causes a tuple (j,c,l,x) to be added to DEAL_i.)
4. If a message sent from j is received and $j \in D_i$, then discard the message.
5. If a message with session identifier (c', i') sent from $j \notin D_i$ is received, then delay this message if there is a tuple $(j,l,c,x) \in \mathrm{ACK}_i$ such that $(c,i) \rightarrow_i (c',i')$ or a tuple $(j,c,l,x) \in \mathrm{DEAL}_i$ such that $(c,j) \rightarrow_i (c',i')$. If there is no such tuple in ACK_i or DEAL_i (or after all such tuples have been removed), then forward the message to the VSS invocation of session (c',i').

We now show that the MW-SVSS protocol satisfies the MW-SVSS properties. To do this, we first must establish two key properties of the DMM protocol.

LEMMA 1. *If i is nonfaulty, then DMM_i satisfies the following two properties:*

(a) if $j \in D_i$, then j is a faulty process;
(b) if j is nonfaulty, $(j,l,c,x) \in ACK_i$ (resp., $(j,c,l,x) \in DEAL_i$), and all nonfaulty processes complete session (c,i) (resp. (c,l)), then eventually (j,l,c,x) is removed from ACK_i (resp., (j,c,l,x) is removed from $DEAL_i$).

PROOF. For part (a), note that the only reason that i adds j to D_i is if $(j,l,c,x) \in \mathrm{ACK}_i$ (resp., $(j,c,l,x) \in \mathrm{DEAL}_i$) and the DMM protocol detects that process j sent a message $\bar{f}_l^j, l, (c,i)$ (resp., $\bar{f}_i^j, i, (c,l)$) using RB such that $\bar{f}_l^j \neq x$ (resp., $\bar{f}_i^j \neq x$). If j is nonfaulty then $x = f_l(j)$ (resp., $x = \hat{f}_i(i)$), hence i would not add j to D_i if j is nonfaulty.

Part (b) follows from the observation that if $(j,l,c,x) \in \mathrm{ACK}_i$ or $(j,c,l,x) \in \mathrm{DEAL}_i$, then the tuple was added during the share phase. If $(j,l,c,x) \in \mathrm{ACK}_i$ and session (c,i) completed, then it must be the case that $j \in \hat{L}_l$ and $l \in \hat{M}_{(c,i)}$. Since j is nonfaulty, then the message required to remove the tuple from ACK_i will be sent using RB by

j during the reconstruct phase, and will eventually be received by i. If $(j, c, l, x) \in \mathrm{DEAL}_i$ then there are two cases. If this entry was removed in line 8 of protocol $\mathcal{S}'$, then we are done. Otherwise, since session (c, l) completed, it must be the case that $j \in \hat{L}_i$ and $i \in M_{(c,l)}$. Hence the message required to remove the tuple from DEAL_i will be sent using RB by j during the reconstruct phase, and will eventually be received by i. $\square$

We now prove that all the MW-SVSS properties hold.

LEMMA 2. *The MW-SVSS protocol satisfies the MW-SVSS properties.*

PROOF. We consider the properties in turn.

Moderated Validity of Termination. If the dealer and the moderator are nonfaulty and $s = s'$ then, for all nonfaulty processes j and l, eventually $(j, c, i, \hat{f}_l^j)$ will be in DEAL_l. Hence, eventually $|L_l|$ will be at least $n - t$. Thus, eventually l will complete step 4 of the share protocol. (For future reference, note that although the first $n - t$ elements of L_l may not all be nonfaulty, at least $t+1$ of the elements of L_l will be nonfaulty.) Moreover, since $j' \in L_l$ only if j' sent an $ack, (c, i)$ message using RB, eventually the moderator will receive an $ack, (c, i)$ message from all $j \in L_l$. Thus, if l is nonfaulty, a nonfaulty moderator will eventually add l to M in step 5 of the share protocol. Since there are $n - t$ nonfaulty processes, eventually we must have $|M| \geq n - t$, so the moderator completes step 6 of the share protocol. We already gave the intuition that a nonfaulty dealer will then broadcast *OK* at step 7. Thus, all nonfaulty processes will eventually complete protocol $\mathcal{S}'$.

Termination. If a nonfaulty process j completes protocol $\mathcal{S}'$, then, since all the messages that caused j to complete the protocol are sent using RB, it follows that all nonfaulty processes eventually complete $\mathcal{S}'$. The fact that they all complete $\mathcal{R}'$ follows since, as observed above, the set L_l for each $l \in M$ contains at least $t + 1$ nonfaulty processes, each of which eventually sends its value in step 1 of $\mathcal{R}'$. Thus, each nonfaulty process outputs either some value in F or $\perp$ at step 3 of $\mathcal{R}'$.

Validity. Suppose that the dealer i is nonfaulty. There are two cases. If some faulty process j such that $(j, l, c, x) \in \mathrm{ACK}_i$ sends a message $x', l, (c, i)$ at step 1 of $\mathcal{R}'$ such that $x \neq x'$, then i did not ignore some message from j during session (c, i), (j, l, c, x) will never be removed from ACK_i, and eventually j will be added to D_i by line 2 in the DMM_i protocol. Hence, j is shunned by i starting in session (c, i). Thus, if no process is shunned by i for the first time in (c, i), it must be the case that, for each process $l \in \hat{M}$, all the values broadcast by processes in $\hat{L}_l$ agree with f_l. Since there will eventually be at least $t + 1$ values broadcast from processes in $\hat{L}_l$, all nonfaulty processes will interpolate f_l for all $l \in \hat{M}$, and subsequently will interpolate f and the secret s.

Weak and Moderated Binding. If the dealer is nonfaulty, it follows from Validity that Weak Binding holds, taking $r = s$. So suppose that the dealer i is faulty. If there is a faulty process j such that $(j, c, i, x) \in \mathrm{DEAL}_l$ for a nonfaulty process l and j sends a message $l, x', (c, i)$ in step 1 of $\mathcal{R}'$ such that $x \neq x'$. In this case l did not ignore a message from j during session (c, i), (j, c, i, x) will never be removed from DEAL_l, and eventually j will be added to D_l by line 3 in the DMM_l protocol. Hence, j is shunned by l starting in session (c, i), so weak and moderated binding holds. On the other hand, if, for each nonfaulty process $l \in \hat{M}$, all the values broadcast by processes in $\hat{L}_l$ are what they were expected to be then, at the time that the first nonfaulty process completes protocol $\mathcal{S}'$, the set $\hat{M}$ is fixed. Let $H \subseteq \hat{M}$ be the set of nonfaulty processes in $\hat{M}$. For each $l \in H$, the value $\hat{f}_l(0)$ is also fixed. If there exists a degree-t polynomial h that interpolates the points in $\{(l, \hat{f}_l(0)) \mid l \in \hat{M}\}$, then let $r = h(0)$; otherwise, let $r = \perp$. We claim that each nonfaulty process will output either r or $\perp$ at the reconstruct phase. This is true since all nonfaulty processes will interpolate $\hat{f}_l$ for all $l \in \hat{M}$ correctly. Since $|H| \geq t + 1$, the values $\{(l, \hat{f}_l(0)) \mid l \in \hat{M}\}$ determine a polynomial h. If all remaining values $\bar{f}_l(0)$ obtained from the polynomials $\bar{f}_l$ for $l \in \hat{M} \setminus H$ agree with h, then r is output; otherwise, $\perp$ is output.

It easily follows from step 5 of $\mathcal{S}'$ that if the moderator is nonfaulty, then the values $\{(l, \hat{f}_l(0)) \mid l \in \hat{M}\}$ can be interpolated only by a polynomial h such that $h(0)$ is the moderator's value s'; that is, $r = s'$.

Moderated Hiding. If the dealer and moderator are nonfaulty then, as long as no nonfaulty process has invoked protocol $\mathcal{R}$, the combined view of any t faulty processes is distributed independently of the value of the shared secret, s. This follows since the dealer uses random degree-t polynomials, so no set of size t learns any information. $\square$

As promised, we now show that it is possible that two nonfaulty processes will complete an invocation of MW-SVSS with different values without detecting a new faulty process.

EXAMPLE 1. *Let $n = 4$ and $t = 1$. Consider an invocation of the MW-SVSS protocol with processes 1, 2, 3, and 4, where 2 is the dealer and 1 is the moderator. Suppose that, in the share protocol $\mathcal{S}'$, process 4 is delayed. Hence, processes 1, 2, and 3 hear only from each other before completing the share protocol. Thus, $L_1 = L_2 = L_3 = M = \{1, 2, 3\}$. Now suppose that in the reconstruct protocol $\mathcal{R}'$, process 3 hears the values sent by 2 according to line 1 of $\mathcal{R}'$ before hearing from 1 or 4. Since it clearly hears from itself as well, $K_{3,1}$, $K_{3,2}$, and $K_{3,3}$ will each have two points—one from 2 and one from 3. Since $t + 1 = 2$ in this case, it follows from step 3 that 3 will then find the unique degree 1 polynomials $\hat{f}_1$, $\hat{f}_2$, and $\hat{f}_3$ that interpolate the points in $K_{3,1}$, $K_{3,2}$, and $K_{3,3}$, respectively. If $\hat{f}_1(0)$, $\hat{f}_2(0)$, and $\hat{f}_3(0)$ are collinear, and $\bar{f}$ is the polynomial that interpolates them, then 3 outputs $\bar{f}(0)$. If 2 is faulty, then by choosing the values it sends appropriately, 2 can make $\bar{f}(0)$ an arbitrary element of F. Now if 1 hears from 3 before hearing from 2 or 4, 1 will also output a value, which may be different from 3's.*

Of course, to get 3 to output a value different from 1's, 2 must send a value $\hat{f}_1^2$ that is different from the one that 1 expects to hear. Once 1 gets this value, it will realize that 2 is faulty, and add 2 to its set D_1. However, this may happen after both 2 and 3 have completed the invocation of MW-SVSS. Notice that this argument relies on the fact that processes use RB to send their values. $\square$

4. IMPLEMENTING SVSS

In this section, we show how to implement SVSS, and then prove that our implementation satisfies the SVSS properties. The difficulties of doing this are illustrated by Example 1:

it is possible that two nonfaulty processes output different values in an invocation (c, i) of the MW-SVSS protocol. Of course, by the Weak Binding property, this can happen only if a new faulty process is eventually detected (and is shunned in all invocations that follow (c, i)). Nevertheless, this detection can come after all processes have completed (c, i). Thus, we must show that the inconsistency cannot cause problems.

Share protocol $\mathcal{S}$:

1. If a dealer i wants to invoke $\mathcal{S}$ with a secret s, it first updates c to $c+1$, initializes sets of processes $G_{(c,i)}$ and $G_{j,(c,i)}$, $j \neq i$, to $\emptyset$ and chooses a random degree-t bivariate polynomial $f(x, y)$ over the field F such that $f(0, 0) = s$.[2] Let $g_j(y) = f(j, y)$ and let $h_j(x) = f(x, j)$, for $j = 1, \ldots, n$. Dealer i sends each process j the message $g_j(1), \ldots, g_j(t+1), h_j(1), \ldots, h_j(t+1), (c, i)$ (so j can reconstruct g_j and h_j).

2. If a process j receives g_j and h_j from dealer i for a session (c, i), then for each process $l \neq j$, process j participates in four invocations of MW-SVSS protocol $\mathcal{S}'$:

   (a) as a dealer with secret $f(l, j)$ and moderator l (who should also have value $f(l, j)$ if i and l are nonfaulty);
   (b) as a dealer with secret $f(j, l)$ and moderator l (who should also have value $f(j, l)$ if i and l are nonfaulty);
   (c) as a moderator with secret $f(l, j)$ and dealer l (who should also have value $f(l, j)$ if i and l are nonfaulty); and
   (d) as a moderator with secret $f(j, l)$ and dealer l (who should also have value $f(j, l)$ if i and l are nonfaulty).

3. The dealer i adds j to the set $G_{l,(c,i)}$ and l to the set $G_{j,(c,i)}$ if the dealer completes all four invocations of the share part of MW-SVSS $\mathcal{S}'$ with j and l playing the roles of dealer and moderator.

4. The dealer i adds j to the set $G_{(c,i)}$ if $|G_{j,(c,i)}| \geq n-t$.

5. If $|G_{(c,i)}| \geq n-t$, the dealer sends $G_{(c,i)}, \{G_{j,(c,i)} \mid j \in G\}, (c, i)$ using RB.

6. When process l receives $\hat{G}, \{\hat{G}_j \mid j \in G\}, (c, i)$ from the dealer and completes all four $\mathcal{S}'$ protocols for each pair j, l such that $j \in \hat{G}$ and $l \in \hat{G}_j$, then it completes this invocation of $\mathcal{S}$.

Reconstruct protocol $\mathcal{R}$:

1. Each process j initializes the set $I_{j,(c,i)}$ to $\emptyset$ and invokes the reconstruct protocol $\mathcal{R}'$ for each of the four invocations of MW-SVSS for each pair (k, l) such that $k \in G_{(c,i)}$ and $l \in G_{k,(c,i)}$. After the four reconstruct protocols associated with k and l are complete, j sets $r^j_{x,k,l,(c,i)}$ to the reconstructed output value for the entry $f(k, l)$ where x was the dealer in the MW-SVSS protocol (so that x is either k or l).

2. For each $k \in G$, process j adds k to $I_{j,(c,i)}$ if

   - there exists $l \in G_k$ such that r_{kkl} or r_{klk} are $\perp$; or
   - there do not exist degree-t polynomials that interpolate $\{(l, r^j_{kkl}) : l \in G_k\}$ or $\{(l, r^j_{klk}) : l \in G_k\}$.

   Intuitively, $I_{j,(c,i)}$ consists of those processes that j ignores in invocation (c, i).

3. For each $k \in G \setminus I_j$, process j computes the degree-t polynomials g_k and h_k that interpolate $\{(l, r^j_{kkl}) : l \in G_k\}$ and $\{(l, r^j_{klk}) : l \in G_k\}$. If there exist $k, l \in G \setminus I_j$ such that $h_k(l) \neq g_l(k)$, then j outputs $\perp$. Otherwise, if there is a unique degree-t bivariate polynomial $\bar{f}$ such that for all $k, l \in G \setminus I_j$, $\bar{f}(k, l) = g_k(l) = h_l(k)$, then j outputs $\bar{f}(0, 0)$; otherwise, j outputs $\perp$.

This completes the description of the SVSS protocol.

LEMMA 3. *The SVSS protocol satisfies the SVSS properties.*

PROOF. For any SVSS session (c, i), if k, j are nonfaulty processes, then all messages sent from k to j will eventually not be ignored. This is true since, if $(c', i') \rightarrow_j (c, i)$, then j completed all $\mathcal{R}'$ invocations associated with (c, i). From the way we use MW-SVSS in $\mathcal{R}$, all processes will also invoke all $\mathcal{R}'$ sessions associated with (c, i). Hence from the Termination property of MW-SVSS and Lemma 1, it follows that all messages that j expects k to send in session (c', i') will eventually be received. We now go through SVSS properties in turn.

Validity of Termination. If the dealer is nonfaulty, then for any two nonfaulty processes k and l, eventually all four invocations of $\mathcal{S}'$ will complete. So eventually the set G_l will be of size at least $n-t$ for each nonfaulty l, the set G will eventually contain at least $n-t$ elements, and all four $\mathcal{S}'$ invocations for each $j \in G$ and $l \in G_j$ will complete. By the properties of RB, all processes will eventually receive the sets G and $\{G_j : j \in G\}$ and, by the Termination property of MW-SVSS, for each $j \in G$ and $l \in G_j$, all processes will eventually complete all four invocations of $\mathcal{S}'$. Hence, all nonfaulty processes will complete protocol $\mathcal{S}$.

Termination. If a nonfaulty process completes protocol $\mathcal{S}$, then it follows from the Termination property of the MW-SVSS protocol and the Reliable Broadcast properties that all nonfaulty processes complete $\mathcal{S}$. The fact that they all complete $\mathcal{R}$ follows from the Termination property of the MW-SVSS protocol.

Validity. Suppose that the dealer i is nonfaulty in an invocation of $\mathcal{S}$ with session id (c, i). There are two cases. If a faulty process j is first shunned by a nonfaulty process l in some MW-SVSS invocation with session (c', i') that is part of the SVSS invocation with session id (c, i), then, because l started (c, i) before starting (c', i') and l completes (c', i') before completing (c, i), j is also first shunned by l starting in session (c, i) of SVSS. On the other hand, if no faulty process is shunned starting in session (c, i), then all invocations of MW-SVSS must satisfy the first clause of the Validity and

[2] Specifically, since a bivariate polynomial of degree t has the form $\sum_{i=0}^{t} \sum_{j=0}^{t} a_{ij} x^i y^j$, we simply set $a_{00} = s$ and choose the remaining coefficients at random from F. Of course, the same ideas apply to choosing a random univariate polynomial f such that $f(0) = s$.

Weak and Moderated Binding properties. It follows from (the first clause of) the Validity property that if $k \in G_{(c,i)}$ is nonfaulty, then for all $l \in G_k$, it must be the case that $r^j_{kkl} = f(k,l)$ and $r^j_{klk} = f(l,k)$ (since k acts as the dealer in computing these values, l acts as the moderator and the values themselves are correct, since they were received from i). Thus, it follows that $k \notin I_{j,(c,i)}$. Similarly, it follows from (the first clause of) the Weak and Moderated Binding property that, for all $k \in G$ and $l \in G_k$, if either l or k are nonfaulty, then it must be the case that r^j_{llk} and r^j_{klk} are each either $f(l,k)$ or $\perp$, and that r^j_{lkl} and r^j_{kkl} are each either $f(k,l)$ or $\perp$. (Here we use the fact that the nonfaulty process —either k or l—is acting as either dealer or moderator in the invocations of MW-SVSS during which these values are computed.) Thus, even if l is faulty, if $l \notin I_j$, then we must have $h_k(l) = g_l(k)$ for all nonfaulty $k \in G$. It follows that, in step 3 of $\mathcal{R}$, j correctly reconstructs h_l and g_l for all $l \in G \setminus I_j$. Thus, the polynomial $\bar{f}$ computed by j will be f, and j will output $f(0,0)$.

Binding. If the dealer is nonfaulty, it follows from Validity that Binding holds, taking $r = s$. If the dealer is faulty, there are again two cases. If a faulty process j is shunned by a nonfaulty process l in some MW-SVSS invocation with session (c',i) that is part of the SVSS invocation session (c,i), then, as argued in the proof of Validity, j is also first shunned by l in invocation (c,i). On the other hand, if no faulty process is shunned starting in session (c,i), then all invocations of MW-SVSS must satisfy the first clause of the Validity and Moderated Weak Binding properties. Consider the time that the first nonfaulty process completes protocol $\mathcal{S}$. At this time, the set G is fixed. Let H be the set of nonfaulty processes in G. Since $|G| \geq n-t$, we must have that $|H| \geq t+1$. If there is a unique degree-t bivariate polynomial $\bar{f}(x,y)$ induced by the entries r_{jjl}, r_{jlj} for all $j \in H$ and $l \in G_j$, then set $r = \bar{f}(0,0)$; otherwise, set $r = \perp$.

We claim that each nonfaulty process will output r at the reconstruct phase. As in the proof of the Validity property for SVSS, it follows from (the first clause of) the Validity property for MW-SVSS that if $k \in G_{(c,i)}$ is nonfaulty, then for all nonfaulty $l \in G_k$, we have that $r^j_{kkl} = \hat{g}_k(l)$ and $r^j_{klk} = \hat{h}_k(l)$, where $\hat{g}_k$ and $\hat{h}_k$ are the polynomials sent by i to k. Thus, $k \notin I_{j,(c,i)}$. Hence, if $r = \perp$, then all nonfaulty processes will output $\perp$. Moreover, if $r \neq \perp$, then, as in the proof of Validity for SVSS, by the Weak and Moderated Binding property, and from the fact that $|H| \geq t+1$, for all $l \in G \setminus I_j$, it must be the case that g_l and h_l agree with $\bar{f}$. Therefore j will interpolate $\bar{f}$ and output r.

Hiding. If the dealer is nonfaulty and no nonfaulty process has invoked protocol $\mathcal{R}$, then the combined view of any t processes is distributed independently of the dealer's value s, because every polynomial h_j and g_j has degree t, and no process learns more than t values of these polynomials. ☐

This completes the construction of the SVSS protocol. We now briefly sketch how, using ideas from Canetti and Rabin [4], we can use SVSS to construct the required asynchronous Byzantine agreement protocol.

5. FROM SVSS TO BYZANTINE AGREEMENT

Once we have SVSS, we can get an almost-surely terminating polynomial protocol for Byzantine agreement with optimal resilience, following the ideas outlined in Canetti's [6] thesis. We proceed in two steps. The first step is to get a common coin. Canetti and Rabin showed that, given $\epsilon > 0$, an AVSS protocol that terminates with probability $1-\epsilon$ could be used to construct a protocol CC that gives a common coin and terminates with probability $1-\epsilon$. We use SVSS to get a *shunning Common Coin* (SCC) protocol.

DEFINITION 2 (SCC). *Let π be a protocol where each party has a random input and a binary output. As in SVSS, we tag each invocation of π with a unique session identifier c. We say that π is a* shunning, terminating, t-resilient Common Coin protocol *(SCC protocol) if the following properties, called the* SCC properties, *hold (in runs with at most t faulty processes in some session tagged c):*

1. ***Termination**. All nonfaulty processes terminate.*
2. ***Correctness**. For every invocation either*
   - *for each $\sigma \in \{0,1\}$, with probability at least $1/4$, all nonfaulty processes output σ; or*
   - *there exists a nonfaulty process i and a faulty process j such that j is shunned by i starting in session c.*

LEMMA 4. *For $n > 3t$ there exists a shunning, terminating, t-resilient Common Coin protocol.*

PROOF. The protocol to implement SCC is exactly the protocol in Figure 5–9 in [6], except that we replace the AVSS protocol with our SVSS. The proof that this protocol satisfies the SCC properties follows from Lemmas 5.27–5.31 in [6], together with the observation that if a process is shunned starting at a SVSS invocation whose reconstruct protocol competes before the SCC protocol invocation completes, then this process is shunned starting at this SCC protocol invocation. ☐

The second step is to use the common coin protocol to get the Byzantine agreement protocol. Canetti and Rabin use their common coin protocol CC that terminates with probability $1-\epsilon$ to get a Byzantine agreement protocol that terminates with probability $1-\epsilon$. We replace the use of CC by SCC to get an almost-surely terminating protocol. The key observation is that in the protocol of Figure 5-11 in [6], if a nonfaulty process j participates in rounds r and r' (and hence, in our setting, it participles in the SCC protocol with session identifiers r and r'), and $r < r'$, then it must be the case that $r \rightarrow_j r'$. Therefore, there can be at most $t(n-t) = O(n^2)$ rounds r such that a nonfaulty process i shuns a faulty process j starting in round r. Hence, there are at most $O(n^2)$ rounds where the SCC protocol does not succeed. In all the remaining rounds, the first clause of the SCC Correctness property holds, so we essentially have a common coin that is sufficiently strong for Byzantine agreement. It therefore follows from Lemma 5.38 and 5.29 of [6] that the expected running time of the protocol is $O(n^2)$. Thus we have the following result.

THEOREM 1 (BYZANTINE AGREEMENT). *There is an almost-surly terminating, polynomial protocol for asynchronous Byzantine agreement protocol with optimal resilience.*

6. CONCLUSIONS

We have shown how to use SVSS to give a protocol for asynchronous Byzantine agreement that has optimal resilience, almost-surely terminates, and is polynomially efficient. Our SVSS protocol has implications for asynchronous Secure Multiparty Computation (ASMPC) of certain functionalities. In the full paper we define a family of functionalities for which the use of SVSS gives a protocol for ASMPC that has optimal resilience, terminates almost surely, and has perfect security (the ideal and real worlds are statistically indistinguishable). Perhaps the major open question remaining is whether there exists an asynchronous Byzantine agreement protocol with optimal resilience and constant expected running time.

APPENDIX

A. BASIC TOOLS

A.1 Weak Reliable Broadcast

A protocol $\mathcal{B}$ with a distinguished dealer holding input s is a t-tolerant *Weak Reliable Broadcast protocol* if the following holds for every execution with at most t faulty processes:

1. **Weak termination**. If the dealer is nonfaulty, then every nonfaulty process will eventually complete protocol $\mathcal{B}$.

2. **Correctness**.

   (a) if a nonfaulty process completes protocol $\mathcal{B}$, then once the first nonfaulty process completes the protocol there is a value r such that each nonfaulty process that completes protocol $\mathcal{B}$ accepts r;

   (b) if the dealer is nonfaulty, then each nonfaulty process that completes protocol $\mathcal{B}$ accepts s.

LEMMA 5. *For $n > 3t$ there exists a t-tolerant* Weak Reliable Broadcast protocol.

PROOF. This protocol, which we call WRB, is essentially Dolev's [8] *crusader agreement*. It uses two types of messages; *type 1 messages* have the form $(r, 1)$ and *type 2 messages* have the form $(r, 2)$. WRB proceeds as follows:

1. The dealer sends $(s, 1)$ to all processes.

2. If process i receives a type 1 message $(r, 1)$ from the dealer and it never sent a type 2 message, then process i sends $(r, 2)$ to all processes.

3. If process i receives $n-t$ distinct type 2 messages $(r, 2)$, all with value r, then it accepts the value r.

If the dealer is nonfaulty, then it is immediate that every nonfaulty process will send $(s, 2)$, and thus will accept s (since there are at most t faulty processes, by assumption). Moreover, if the dealer is nonfaulty, the only type 2 message sent by a nonfaulty process is $(s, 2)$, so no nonfaulty process will receive more than t type 2 messages $(r, 2)$ with $r \neq s$.[3]

[3]We assume that, as in VSS, if there are multiple invocations of WRB, messages are tagged with an invocation number, so that messages from old invocations will not be confused with messages from the current invocation.

To see that WRB satisfies the correctness property, suppose, by way of contradiction, that one nonfaulty process i accepts r and another nonfaulty process j accepts r', with $r \neq r'$. Then i must have received $n - t$ type 2 messages with value r and j must have received $n-t$ type 2 messages with value r'. Thus, at least $n - 2t \geq t + 1$ processes must have sent a type 2 message to both i and j. At least one of these processes must be nonfaulty. But the protocol ensures that a nonfaulty process will send only one type 2 message. This gives us the desired contradiction. $\square$

A.2 Reliable Broadcast

A protocol $\mathcal{B}$ with a distinguished dealer holding input s is a t-tolerant *Reliable Broadcast protocol* if the weak termination and correctness properties of the Weak Reliable Broadcast holds, and in addition, the following property holds:

3. **Termination**. For every execution with at most t faulty processes, if some nonfaulty process completes protocol $\mathcal{B}$ then all nonfaulty processes will eventually complete protocol $\mathcal{B}$.

LEMMA 6. *For $n > 3t$ there exists a t-tolerant* Reliable Broadcast (RB) protocol.

PROOF. This protocol, which we call RB, is essentially Bracha's *echo broadcast*. It uses WRB as a subroutine. In addition to type 1 and type 2 messages, it uses *type 3* messages, which have the form $(r, 3)$. RB proceeds as follows:

1. The dealer sends $(s, 1)$ to all processes using Weak Reliable Broadcast (WRB).

2. If process i accepts message r from the dealer using WRB, then process i sends $(r, 3)$ to all processes.

3. if process i receives at least $t + 1$ distinct type 3 messages with the same value r, then process i sends $(r, 3)$ to all processes.

4. if process i receives at least $n - t$ distinct type 3 messages with the same value r, then it accepts the value r.

To see that RB is correct, first observe that, from the correctness property of WRB, it follows that it cannot be the case that two type 3 message with different values are sent by nonfaulty processes at step 2. Moreover, if a nonfaulty process sends a type 3 message at step 3, it must be because it got a type 3 message from a nonfaulty process. It easily follows that all the type 3 messages sent by nonfaulty processes at either step 2 or step 3 have the same value.

If the dealer is nonfaulty, then it is easy to see that all nonfaulty processes terminate and accept value s, as in WRB. To see that termination holds for RB, suppose that a nonfaulty process completes the protocol. It thus must have received $n - t$ type 3 messages with the same value r. Each other nonfaulty process will eventually have received at least $n - 2t \geq t + 1$ of these messages, and so will send a type 3 message by step 3, if it has not already done so by step 2. As we argued above, all the type 3 messages sent by nonfaulty processes must have the same value. Thus, each nonfaulty process will end up receiving $n - t$ type 3 messages with value r.

Finally, part (b) of correctness follows easily from our observation above that all the type 3 messages sent by nonfaulty processes have the same value r. $\square$

B. REFERENCES

[1] M. Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proc. 2nd ACM Symposium on Principles of Distributed Computing*, pages 27-30, 1983.

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th ACM Symp. Theory of Computing*, pages 1–10, 1988.

[3] G. Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *Proc. 3rd ACM Symp. Principles of Distributed Computing*, pages 154–162, 1984.

[4] R. Canetti and T. Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *Proc. 25th ACM Symp. Theory of Computing*, pages 42–51, 1993.

[5] R. Canetti and T. Rabin. Fast asynchronous Byzantine agreement with optimal resilience, 1993. http://people.csail.mit.edu/canetti/materials/cr93.ps.

[6] R. Canetti. Studies in secure multiparty computation and applications, 1996. http://people.csail.mit.edu/canetti/materials/thesis.ps.

[7] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43:685–722, 1996.

[8] D. Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3:14–30, 1982.

[9] P. Feldman and S. Micali. Optimal algorithms for Byzantine agreement. In *Proc. 20th ACM Symp. Theory of Computing*, pages 148–161, 1988.

[10] P. Feldman and S. Micali. An optimal probabilistic protocol for synchronous Byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.

[11] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty processor. *Journal of the ACM*, 32(2):374–382, 1985.

[12] J. Katz and C.-Y. Koo. On expected constant-round protocols for Byzantine agreement. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 445–462. Springer, 2006.

[13] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. of the ACM*, 27(2):228–234, 1980.

[14] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. 21st ACM Symp. Theory of Computing*, pages 73–85, 1989.