OPTIMAL ALGORITHMS FOR BYZANTINE AGREEMENT

by

PAUL NEIL FELDMAN

B.A. Mathematics, Yale University
(1983)

Submitted to the Department of Mathematics
in Partial Fulfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY IN MATHEMATICS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1988

© Massachusetts Institue of Technology 1988

Signature of Author [illegible]
Department of Mathematics
May 13, 1988

Certified by [illegible]
Frank T Leighton
Professor of Applied Mathematics

Certified by
Silvio Micali
Professor of Computer Science
Thesis Supervisor

Accepted by
Professor Willem V R Malkus, Chairman
Committee on Applied Mathematics

Accepted by [illegible]
Prof[illegible] Helgason, Chairman
Departmental Graduate Committee

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
AUG 19 1988
LIBRARIES

OPTIMAL ALGORITHMS FOR BYZANTINE AGREEMENT

by

PAUL NEIL FELDMAN

B.A. Mathematics, Yale University
(1983)

Submitted to the Department of Mathematics
on May 13, 1988 in Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy in Mathematics

Abstract

We exhibit randomized Byzantine agreement algorithms achieving optimal running time and fault tolerance.

Our algorithms do not require trusted parties, preprocessing, non-constructive arguments, or cryptography.

For a synchronous, complete network with private communication lines, our Byzantine agreement algorithm runs in constant expected time, against all types of adversaries considered in the literature.

We remark that a modification of our solution enables an asynchronous network to reach Byzantine agreement in constant expected running time. The only previously known solution had exponential expected running time in the worst case.

Thesis Supervisor: Silvio Micali

Title: Professor of Computer Science

Acknowledgement

The thesis represents joint research with my advisor, Silvio Micali. I wish to thank Professor Micali for his vast contribution to my development as a computer scientist. It has been an honor and a pleasure to do research with him. His broad experience in the field and his keen insight in pinpointing the fundamental open questions have been invaluable assets. His concern for clear presentation of results has influenced this thesis beyond measure.

I also thank the other members of my thesis committee, Tom Leighton, Nancy Lynch, Ron Rivest, and David Shmoys. Special thanks are due to David Shmoys for going above and beyond the call of reader. His very careful reading and constructive criticisms led to numerous improvements in the presentation of both formal and non-technical sections. Nancy Lynch helped formalize certain points which had been vague in previous drafts. Many valuable insights were gained in conversations with Michael Ben-Or, Benny Chor, Danny Dolev, Cynthia Dwork and Shafi Goldwasser.

I cannot imagine a more stimulating place for theoretical computer science than the MIT Lab for Computer Science. Working with the professors, graduate students and visitors has been a most enjoyable way of acquiring familiarity with a broad class of research areas and approaches. I owe a great deal to the theory group in general. Beyond academics, I owe thanks to Baruch Awerbuch, Bard Bloom, Claude Crepeau, Ray Hirschfeld, and Mark Reinhold for helping me in various aspects of typesetting.

Financial support for this research has been provided by NSF Grant DCR-84-13577 and ARO Grant DAAL03-86-K-0171.

Last but not least, I thank my teachers, family and friends for their counsel, their example, and their encouragement through this and all my endeavors.

Table of Contents

1. Introduction

The problem of Byzantine agreement (BA) was introduced by Pease, Shostak and Lamport [Pease, Shostak and Lamport 1980]. It may be the most important problem in distributed computation among fallible processors. Processor faults may range from simple mistakes to total breakdown to skillful adversarial intent. Trying to maintain a common view of the world is difficult when one does not know whom to trust. BA is a key step in this direction: it enables all *good* processors (those that follow the protocol) to coordinate themselves. Consider a situation in which each processor holds an initial value. Informally, for any set of initial values, BA should give us the following properties:

(1) *Consistency*: All good processors adopt a common value.

(2) *Meaningfulness*: If all good processors start with the same value, then they adopt that value.

Moreover, these properties should hold even if some processors are *maliciously* faulty, and try to ensure that they will not be satisfied. In general, good processors do not know which are the faulty processors. Faulty processors may coordinate their messages in such a way as to try to mislead the good processors into disagreement.

The classical example which led to the name *Byzantine* agreement is the problem of coordinating divisions of an army. There is one commanding general; each division is led by a lieutenant. It is possible that the general and/or some minority of the lieutenants are traitors, and wish that the army suffer a crushing defeat. Each division may attack or retreat. The basic assumption is that an attack by all loyal lieutenants will probably succeed, but any partial attack will get trounced. A proper general would never order a partial attack, but a treasonous general might. The loyal lieutenants must have some way of reaching a common decision, attack or retreat (consistency). Moreover, when the general is loyal, his orders should be carried out (meaningfulness).

More generally, BA is required whenever a value must be *broadcast* in a situation where all communication is person-to-person. When a processor tries to broadcast a message by sending it to all other processors, a processor receiving the message does not know whether or not all other processors received the same message. If the other processors are queried, and some of them report getting different versions of the message, it is not clear whether the fault lies in the sender or the receivers or both. In such a situation, one version must be selected and agreed upon as the "real" message, or we throw out the message entirely; this is the primary application of a BA algorithm. If the algorithm lacks consistency, then a faulty broadcaster might mislead some of the processors into computing the wrong function, working with the wrong data, etc. An algorithm without meaningfulness could enable faulty processors to halt progress by invalidating good broadcasts.

BA is needed for many contemporary situations. For example, components of an automatic pilot must make decisions based on readings from numerous gauges, which may give, say, contradictory readings of altitude. It would not be pleasant if the left engine shut off for landing, while the right engine aborted the idea to land and turned on full force. Geologists may have to reach consensus on the timing of shock waves recorded at various locations, even though the clocks are not perfectly synchronized; they must select one peak from each seismograph to correspond to the same tremor. Party bosses who have pledged allegiance to different candidates in simultaneous primaries may have to consolidate forces and decide to back one of them.

The examples we are most concerned with come from distributed computing, where large numbers of processors must work together on a problem. A major problem in such computer systems is the unreliability of the processors, which may cause them to deviate from a specified protocol. There are many senses in which a processor may fail to follow a protocol. A "benevolent" faulty processor, when it broke down, would cease communications. If all faults are benevolent, then all delivered communications are from good processors. More realistically, we might expect a faulty processor to continue communicating with some processors but not with others, corresponding to broken communication links. Alternatively, there might be "random" errors in its communications, for example, if a memory register was shortcircuited; it is likely that such errors would occur only in certain subroutines. Effectively, the processor is running its own version of the protocol it is supposed to be running. Most generally, a processor might send messages maliciously, intending to wreak havoc on the network. (The fact that systems crash from time to time suggests that faults which we would expect to be random sometimes end up being malicious.) Furthermore, when different people control different processors, then we must be prepared for malicious deviation.

Sometimes, a faulty party may be satisfied if it can delay agreement. Consider negotiations between nations in a cartel trying to agree on a new policy regarding production quotas and pricing policy. Imagine that a few countries prefer the status quo to either of the two alternatives being voted on. Their desire might be to simply delay agreement as long as possible. Let us assume that these countries represent the swing votes (i.e., there is no majority without them). They might vote one way to half the countries, and the opposite way to the others, and thereby cause disagreement on which policy was decided on. They can further confuse the issue by simply not sending any votes to certain countries, and claiming not to have received votes from others. For such situations, we need some way to reach a decision everyone will agree on; we need a Byzantine agreement protocol (BAP), and a quick one.

The above problems are not limited to voting. Any time processors try to cooperate, they must have a way to resolve discrepancies. As it is hard to predict what type of faults will occur in a real network, perhaps the best way to guarantee success in this endeavor is to prepare for the worst possible scenario. For our problem, we have nothing to lose by doing worst-case analysis, since we shall see that even the worst case can be handled efficiently.

1.1 Previous Solutions

The fundamental parameters of a BA protocol are: n, the size of the network; and t, the maximum number of faulty processors. In the most important and difficult case, $t=\theta(n)$; we call this the *worst scenario*. Enormous attention has been devoted to the BA problem, and many weaker scenarios (e.g., $t=O(\sqrt{n})$) have been considered; we address the worst scenario. We also adopt a worst-case analysis regarding the ***adversary*** that may coordinate the faulty processors; this shall be formalized in the next section.

The worst scenario is actually the most realistic. In any production line, quality does not increase with quantity; at best, we may hope that the frequency of defects is a fixed percentage. In many cases, we may intentionally increase the network size to decrease the chance that the fraction of faulty components significantly exceeds the expected value. The same reasoning applies to the case where people are deliberately malicious. We would expect a group of 1000 people to have ten times as many who would like to beat the system as a group of 100; during a mid-season strike, all players of a team currently in first place may stand to gain from stalled negotiations, whether there are nine or ninety players on the team. Of course, the best justification for restricting attention to the worst scenario is the fact that our solution is optimal for all scenarios.

Remark 1: When we speak of optimal running times, we shall always mean within a constant factor. By contrast, optimal fault tolerance implies exactly matching the proven bound on the number of faulty processors tolerable by a BA protocol.

For deterministic protocols, [Pease, Shostak and Lamport 1980] prove that no BAP can tolerate $n/3$ faults. They also show that BA can be reached in $t+1$ rounds of communication for any $t<n/3$, but using exponential communication. Dolev, Fischer, Fowler, Lynch and Strong [Dolev, Fischer, Fowler, Lynch and Strong 1982] present a $2t+3$ round BAP with polynomially bounded communication for any $t<n/3$. This protocol is optimal, up to a constant factor in the running time. Fischer and Lynch [Fischer and Lynch 1982] show that $t+1$ rounds is a lower bound on the running time of any deterministic BAP. (Srikanth and Toueg [Srikanth and Toueg 1984] present a $2t+1$ round protocol; Coan [Coan 1987] shows how to achieve running times between $t+1$ and $2t$, at the expense of increased communication complexity.) Thus, we must resort to probabilistic solutions for faster running times.

Randomization for reaching BA was first used by Ben-Or [Ben-Or 1983] and then perfected by Rabin [Rabin 1983] using the idea of a ***common coin***, a tool that has been essential to all subsequent solutions. A probabilistic BA algorithm tolerating $t<n/3$ faults and running in expected time $O(t/\log n)$ was found by by Chor and Coan [Chor and Coan 1985]. Although this algorithm beats the lower-bound for deterministic protocols, the running time is high in the worst scenario.

Faster algorithms (also tolerating $(n-1)/3$ faults) are known if both *cryptography* is used (in which case we must assume a computationally-bounded adversary) and a "trusted party" has suitably initialized the network. Three main protocols are known. [Rabin 1983] showed that if a trusted dealer distributes, beforehand, $O(d)$ pieces of information to each processor, then d Byzantine agreements can be reached subsequently, each in expected constant time. Bracha [Bracha 1985] shows, by a non-constructive argument, how a *properly initialized* network may reach an *unlimited* number of agreements, each in expected time $O(\log n)$. Building on [Bracha 1985], Feldman and Micali [Feldman and Micali 1985] finally show how a properly initialized network can reach an unlimited number of agreements, each in constant expected time.

It is important to notice that the above three algorithms all *require cryptography*, even if the lines in the network guarantee private communication.[1]

The assumption of a trusted dealer (though available only for a limited amount of time) is very strong. If a single totally reliable processor would be available throughout the protocol, BA would be trivial: merely adopt his value! In [Feldman and Micali 1985], the authors succeed in eliminating this bold assumption: they show how the network can "initialize itself", without trusted parties or non-constructive arguments, by running an initial deterministic agreement. (This requires $O(t)$ rounds of pre-processing.) However, preprocessing is also a strong assumption. The first agreement is often the most important one (e.g., whether or not to hold a meeting). For many applications, such as a network which gains or loses processors, we cannot assume preprocessing. Thus, there is a need to be able to reach fast BA *from scratch*, i.e., without preprocessing or a trusted dealer. None of the previous algorithms achieve this goal in the worst scenario. The most sophisticated BA from scratch in the literature is presented by Dwork, Shmoys and Stockmeyer [Dwork, Shmoys and Stockmeyer 1986], although it does not address the worst scenario: there exists a constant c such that whenever $t<(cn/\log n)$, their protocol runs in constant expected time.

1 In [Rabin 1983], secure signature schemes are required. In [Bracha 1985] and [Feldman and Micali 1985], processors must commit themselves to values by revealing encryptions of them; cryptography is used to hide the values until they are to be revealed.

1.2 Our Result

We exhibit protocols to reach BA *from scratch* in constant expected time, tolerating $(n-1)/3$ faults. Our protocols simultaneously achieve optimal running time and optimal fault tolerance; this follows from the result of Karlin and Yao [Karlin and Yao 1987], who extend the bound of [Pease, Shostak and Lamport 1980] to show that even probabilistic BA protocols cannot tolerate $n/3$ bad processors.

Given private channels,[2] our protocols are non-cryptographic, and withstand even an adversary with infinite computing power.

Our basic protocol may be adapted to run in asynchronous networks. In Feldman [Feldman 1988], we present the asynchronous version of our protocol, which runs in constant expected asynchronous time. This protocol tolerates up to $t<n/4$ faults. We remark that using cryptography, and assuming a computationally-bounded adversary, we can regain optimal fault tolerance $t<n/3$ in the asynchronous case as well.

In [Feldman 1988], it is shown how to compile *any* protocol assuming private channels to a cryptographic protocol that does not assume private channels, which performs just as the original protocol. Our compiled BA algorithms are optimal in the cryptographic setting. This follows from the work of Dolev and Dwork [Dolev and Dwork 1987], who extend the result of [Karlin and Yao 1987] to the setting in which cryptography may be used against a polynomial-time adversary. They show that $n/3$ faulty processors may not be tolerated, unless the network has previously agreed on public keys for signatures.

For the cryptographic scenario in which the network *has* previously agreed on public keys for signatures, an extension of our algorithm reaches BA in constant expected time whenever the majority of processors are good [Feldman 1988].

2. Preliminaries

2.1 Model of Computation

We consider a network N of n processors with identities $1,2,...,n$, where $n \geq 4$. For convenience of discourse, we shall often refer to processors as *players*; the variables g,h,i and j will denote identities of players. Each processor is an *interactive probabilistic polynomial-time Turing machine*, as we explain. This consists of a finite state control; read-only input tapes; exclusive-read, exclusive-write work tapes; exclusive-write output tapes; and an exclusive read-only *random tape*.

2 Communication channels that guarantee private communication.

A *communication link* from processor i to j is an output tape of processor i which is also an input tape of j; we shall assume it is labelled l_{ij}. Processor i *sends* a message to j by writing it on l_{ij}. This link is exclusive-write for i (i.e., no processor other than i can write to it). If it is exclusive-read for j, then we call it a *private channel*. We shall assume the network is *complete* with private channels; there is a private link from each processor to every other processor.

A *broadcast channel* is an output tape of one processor that is an input tape of all processors, i.e., everyone can read what is written to the tape. We do *not* assume broadcast channels (instead, we shall show how to simulate them).

We assume a *synchronous* network. This says that all communications are sent during time intervals defined by pulses of a clock accessible to all players. The period between the r-th and $r+1$st pulses is called the *r-th round*. For each i,j, and r, we let m_{ijr} denote the characters i writes on link l_{ij} in the r-th round, and call this the *$r-$ th round message* of i for j. Processor j inputs m_{ijr} at the $r+1$-st pulse.

Implicit in the model is the fact that every processor knows the sender of each received message. This is a prerequisite for BA; if, instead, messages arrived "unlabelled", a single bad processor could impersonate n processors, making itself the majority and making BA impossible. Also implicit in the model is that the identities of all processors are commonly known to all processors.

The random tape is an infinite string of bits selected with independent uniform probability which are read as input sequentially. Reading a bit on the random tape enables a processor to enter either of two states with probability 1/2. We define the (initial/instantaneous) *configuration* of a processor as its (initial/current) state and the contents of all tapes *excluding* the random tape.

A (distributed) *protocol* is an n-tuple of programs (i.e., finite state controls).[3] We may view each processor i as reserving three special tapes for each protocol P. One tape stores the *common parameters*; these values are the same for all processors at the start of P. As we are considering a BA from scratch, the only "given" common parameter will be n, the size of the network. (To simplify the definition of polynomial-time computation given below, we assume that n is written in unary). Other common parameters to P may be specified by the code of a protocol Q which *calls* P (as described below).

Remark 2: In presenting the code for protocols, we shall sometimes find it expedient to define additional common parameters in terms of already defined common parameters. If a protocol Q

3 We may view the i-th program as a multi-valued function F_i whose arguments correspond to the input tapes, work tapes, and the initial unread section of the random tape of processor i; the values returned by F_i are written on the output tapes and work tapes of processor i.

calls many executions of P to be run, it may index these executions by assigning an additional common parameter, which we call an ***execution label***, to each execution. An execution label for P does not appear in the code of P.

A second tape stores i's ***private input*** (e.g., a value i received from the commanding general, or i's output from some other protocol). We define i's ***input*** to P to consist of the common parameters and i's private input to P. The third tape is an output tape which is not a communication link, on which i's (private) output of P is written. All tapes, other than the common parameter tape, the private input tape, and the random tape, are initially blank; i starts P in a distinguished state *Start*.

We impose conditions on the initial configuration of a processor to guarantee that events that should be independent truly are. Consider a protocol Q that calls protocol P; Q specifies the values of i's input to P. We implicitly assume that all relevant ***return*** information for processor i (e.g., the step of Q from which P was called, the current state of i, the value of all variables in Q) is appended to i's input to P. Processor i surrounds the return information for Q by special characters; i does not read between these characters until terminating P. All tape cells i accesses during P, other than i's input to P and random tape cells, are blank when i starts P. Upon ending P, i erases all tape cells accessed during P, other than the private output of P. At this time, i reads the return information and resumes executing Q. We allow nested calls of protocols.

We shall also need the idea of "hiding away" information for a future call of a protocol. During a protocol P, a processor may ***store*** an input for a protocol Q. This stored input is not erased upon ending P; however, it may only be accessed by Q. We shall used stored inputs in ***compound*** protocols $P=(Q,R)$, where Q and R are ***component*** protocols. The common parameters of P are the common parameters of Q and R. The private input to P is the private input to Q; during Q, private inputs may be stored for R. A call of P starts Q alone; R may be called, from Q or any other protocol, at any subsequent time (or possibly never).

A processor is ***polynomial- time*** if the number of computational steps it takes in any round is bounded by a polynomial in the length of its input. That is, there exists a polynomial Q such that $Q(k)$ is an upper bound on the number of computational steps the processor may perform in a round, when its input has length k. (Since we assume that the unary representation of n is a common parameter, $k \geq n$.) Notice that $Q(k)$ is an upper bound on the length of any message when the input has length k.

2.2 Faulty Processors

We say that processor i is *good* in an execution of a protocol P if i follows every step of P correctly. Processor i is considered ***faulty***, or ***bad***, if it deviates from P in any way. The most general (and difficult to overcome) kind of faulty behavior occurs when the faulty processors are selected and coordinated by an ***adversary***. Informally, an adversary is an algorithm that may ***act on*** a network running a protocol and can ***corrupt*** (make faulty) a fixed fraction of the processors.

Definition 1: Let A be an interactive probabilistic Turing machine; let r be a constant between 0 and 1. We say that A is an ***r− adversary*** if A may ***act*** on any network of n processors, running any protocol P, as follows:

(1) A has read-access to every communication link that is not a private channel.

(2) A may ***corrupt*** any $t<r\cdot n$ processors: when A corrupts processor i, A gets read-access to all of i's tapes and seizes exclusive-write control of i's output tapes.

(3) *Dynamicity*: A may corrupt processors at arbitrary times during P.

(4) *Scheduling* (often referred to as ***rushing*** in the literature): All d-th round messages sent by uncorrupted players are written at the start of the d-th round. A immediately inputs all non-private channel messages, and private channel messages sent to corrupted players. Upon corrupting the sender or intended recipient of a private channel message during this round, A immediately inputs the message. A may corrupt additional processors (but not more than t overall) and may output the round-d messages of the bad players at arbitrary times during round d.

(5) A can perform an unlimited number of computational steps instantaneously.

Remark 3: A processor is considered faulty for the mere act of allowing another party to read its internal tapes, as this is not part of any protocol.

In effect, a corrupted processor i becomes an extension of A; A may replace i's finite state control with any other finite state control.

Property (3), dynamicity, is an important capability when acting on a randomized protocol. For deterministic protocols, A's optimal strategy may be calculated beforehand. For randomized protocols, the optimal strategy may change during the protocol. For example, consider a protocol which randomly selects a leader for some task. A dynamic adversary can wait until the selection is completed, and then corrupt the leader!

We consider the ***resiliency*** of a protocol to be the greatest fraction of bad players it can tolerate.

Definition 2: Let P be a protocol satisfying a set of properties Ψ. We say that P is $r-$ *resilient* (with respect to Ψ) if Ψ is satisfied when any r-adversary acts on P.

(The set of properties will always be clear from context). Ψ need not hold when a more powerful adversary acts on P. When we mention an (unspecified) *adversary* acting on an r-resilient protocol P, any statements made need only apply to an $r-$ adversary. (For every protocol we present, r will be a constant; in describing some protocols appearing in the literature, we shall stretch our definition to let the resiliency be a function of n.)

We shall let t, an upper bound on the number of faulty processors (for which the protocol should work correctly) be a common parameter of P; since n is a common parameter of P, and the resilience of P is a property of P, such a bound is specified in terms of previously defined common parameters. A protocol calling P may itself specify a smaller upper bound t; if Q, a 1/4-resilient protocol, calls P, a 1/3-resilient protocol, Q may specify that $t<n/4$ (since if $t\geq n/4$, Q is not guaranteed to work, anyway).

We let *good* denote a processor that is uncorrupted throughout the entire execution of a protocol or compound protocol P. (Often, it will be simpler and sufficient to prove statements concerning the good processors, even though these statements hold for all currently uncorrupted processors.) A processor that *correctly* terminates P is good, although the converse need not be true. Good processors cannot directly observe the corruption of a processor, although in some cases they may be able to deduce it from subsequent messages (or the lack thereof).

The *start* of a protocol P is the earliest round in which an uncorrupted processor starts P. (Usually, all uncorrupted processors start P at the same time.) A protocol *terminates*, or *ends*, as soon as every good processor terminates. A protocol is *terminating* if for any input, there exists a finite upper bound on the number of rounds until it ends. It is *jointly* terminating if all good processors always terminate in the same round. A compound protocol is jointly terminating if each of its components is. The *duration* of an execution of P is the time interval from the start of P to the end of P (or forever, if P never ends).

2.3 Analysis of Probabilistic Protocols

A rich theory of probabilistic uniprocessor algorithms has been developed. Probabilities are naturally taken over the random bits read by the processor. It is less obvious how to define the probabilities for a distributed protocol on which an adversary acts. Ideally, we would like these probabilities to be properties of the protocol, and independent of the actual adversary.

We observe that the entire execution of a protocol P is determined by:

(1) The adversary A acting on P, and its configuration at the start of P. (Note: since A may read all tapes of bad players, we shall consider A's configuration to include the configurations

of corrupted players.)

(2) The inputs to P of all players that are uncorrupted at the start of P.

(3) All random bits read during the duration of P (any bits read before the start of P would only affect the initial configurations).

(By assumption, all other tapes are initially blank.)

Let E be an event. For fixed A and fixed values of the initial configurations (i.e., of all uncorrupted processors and A), ((1) and (2) above), the probability that E occurs during P is computed over all random tapes (i.e., of uncorrupted players and A). Whereas these probabilities *do* depend on A and A's initial configuration, we can prove bounds on the probabilities which are valid for any adversary.

Definition 3: Let P be an r-resilient protocol. We say that *E occurs in P with probability at least p* if for any $r-$adversary A, for any set of initial configurations, the fraction of random tapes for which E occurs is at least p. An immediate and crucial consequence of our definition is that if P and Q are protocols of disjoint (non-overlapping) duration, and events E_P and E_Q occur during P and Q with probabilities at least p and q respectively, then the probability that both occur is at least pq. These events are, in general, *not*, independent, since the initial configurations of the latter protocol may depend on events which transpire in the former; nevertheless, we may multiply the bounds on the probabilities, since they hold for any initial configurations.

Similarly, at any time during P, all future events in P are determined by A, the instantaneous configurations, and the unread portions of all random tapes. Let E be an event which may occur after time T. In a particular execution of P, we say that E is *fixed at time T* if for the *actual* instantaneous configurations of all uncorrupted players at time T, for any A and any instantaneous configurations of A at time T, for all possible (unread sections of the) random tapes, E later occurs. The probability that E is fixed at time T is computed over random bits read from the start of P until T.

2.4 Complexity of a Protocol

There are three complexity measures of a distributed protocol; in general, they are functions of the size of the network, the adversary acting on it, the initial configurations, and all random tapes. One measure is the *global* (*distributed*) running time, that is, the number of rounds of communication before the protocol terminates. The *communication complexity* is the total number of bits good processors send during the protocol. Finally, the (local) computation complexity is the number of bit operations performed by good processors. For an r-resilient protocol, we define these complexities to be the maximum expected values (averaged over all possible random tapes) for any $r-$adversary, for any initial configurations.

For BA. which is primarily a distributed problem, research has concentrated on minimizing the global running time, provided that the communication and computation complexity remain polynomially-bounded. Our solution runs in $O(1)$ expected global time, and thus is optimal.

We now give the formal statement of the problem.

2.5 Definition of Byzantine Agreement

In most situations for which BA is needed, the values to be agreed upon may have arbitrary length. Nevertheless, it is known that the problem of reaching Byzantine agreement on arbitrary values easily reduces to reaching agreement on binary values (see Section 4.2); we define this latter problem as BA.

Definition 4: Let P be a protocol in which each good player i has a private input bit, B_i. Each i that correctly terminates outputs a bit, d_i. P is a BA protocol if

(1) For any i and j that correctly terminate. $d_i = d_j$.

(2) If $B_i = B_j$ for all good players i and j, then for each i that correctly terminates, $d_i = B_i$.

Our definition does not require that the protocol terminate. A BAP is interesting only if it terminates with positive probability. In light of Remark 2, a BA protocol is interesting only if has positive resiliency.

2.6 Organization

In Section 3, we present ***verifiable secret sharing,*** an interesting protocol in its own right, but one which apparently cannot be used to reach BA. In Section 4, we build a BAP from modular protocols. The first is a simple primitive used to weakly simulate broadcasts. This primitive is used to adapt verifiable secret sharing to a form which may be employed to reach BA. We then define a ***common coin*** protocol, and show a rather involved reduction from a common coin protocol to the adapted verifiable secret sharing. Finally, we show that BA is reducible to a common coin protocol.

For simplicity, we first present a 1/4-resilient verifiable secret sharing. The reductions of Section 4 are (at least) 1/3-resilient; accordingly, this gives a 1/4-resilient BA protocol. In Section 5, we present a 1/3-resilient verifiable secret sharing, which yields an optimal, 1/3-resilient BAP. All protocols we present run in constant expected time. In the appendix, we analyze the complexity of the most efficient implementations. There, we also state results concerning asynchronous and cryptographic versions of our protocols.

3. Verifiable Secret Sharing

Verifiable secret sharing is a very useful and powerful protocol. It may be run on a broadcast network, i.e., a network in which each processor has a broadcast channel. In Section 4, we shall define and present a weaker version of verifiable secret sharing which does not require a broadcast network. In this section, in which we briefly survey the development of verifiable secret sharing, we shall assume a broadcast network. (We also retain the assumption that the network is complete with private channels.)

3.1 Secret Sharing

The concept of ***secret sharing*** was introduced by Shamir [Shamir 1979] and independently by Blakely [Blakely 1979]. Simply stated, secret sharing allows a distinguished processor, the ***dealer***, to give a "time-release" message to a network of n processors, at most t of which may be bad. We shall assume that $t<n/2$.

A ***secret sharing*** is a compound protocol, $(Share, Recover)$. In $Share$, the dealer privately sends a ***piece*** of a ***secret*** message to each processor. Each processor saves the piece he receives as a stored private input to $Recover$. Any piece, by itself, is useless; in fact, any collection of t pieces provides no information about the secret. However, any $n-t$ players, by running $Recover$, can recover the secret. If t bounds the number of bad players, then we see that the secret may not be recovered without the cooperation of (at least) one good player. By contrast, the pieces of the good players alone determine the secret. We propose a formal definition of secret sharing geared towards motivating verifiable secret sharing. We shall only use secret sharing in situations in which the dealer randomly selects the secret to be shared from an interval; we simplify the definition accordingly.

Definition 5: Let $P=(Share, Recover)$ be a jointly terminating compound protocol to be executed on a broadcast network, including as common parameters:[1] the identity of the dealer, h; and the number of possible secrets, m. Player i outputs an element of $[0,m-1]$, a_i, at the end of $Recover$. P is a ***secret sharing*** if the following properties are satisfied:

(1) (Recoverability) Let E_σ be the event that all good processors output σ in $Recover$. If the dealer correctly terminates $Share$, then there exists $\sigma\in[0,m-1]$ such that E_σ is fixed at the end of $Share$.

(2) (Unpredictability) Let A be an adversary acting on P which never corrupts the dealer and outputs a value $\tau\in[0,m-1]$ (as its prediction of the secret) before the start of $Recover$. Then the probability that E_τ is fixed at the end of $Share$ (i.e., A's prediction is correct) is $1/m$.

1 We need not list n and t, which are common parameters of every protocol.

Intuitively, the secret is σ whenever E_σ is fixed at the end of *Share*. A good dealer randomly picks $\sigma \in [0, m-1]$ to be the secret.

[Shamir 1979] proposes an elegant secret sharing scheme. Consider, as an additional common parameter, a prime p, where $p > n$, $p \geq m$; all calculations shall take place in Z_p, a finite field of p elements. (Note: we may specify p to be the least prime exceeding $m-1$ and n.) The dealer uniformly chooses a $t-$th degree polynomial[2], S, whose constant term, $S(0)$, is between 0 and $m-1$. This implicitly defines the secret as $\sigma = S(0)$. Processor i receives the value of S evaluated at i, i.e., $S(i)$, as his piece of the secret, and saves this as a stored private input to *Recover*. In *Recover*, all good players broadcast their pieces. Using polynomial interpolation, it is easy to find S, and hence σ, from the first $t+1$ pieces (actually, any set of $t+1$ pieces may be used).

We informally argue why unpredictability (2) should hold, i.e., the adversary's *prediction* is independent of the secret. Let A be an adversary that does not corrupt the dealer, and outputs a value $\tau \in Z_m$ before the start of *Recover*. The pieces of the bad players alone specify neither S nor σ. In fact, the pieces of bad players are uniformly distributed, independent of the secret. This is because the dealer chose S uniformly; for any $\sigma \in Z_p$, for any t-tuple of bad players, $\{j_1, ..., j_t\}$, for any t-tuple of elements of Z_p $\{q_1, ..., q_t\}$, there is a unique $t-$th degree polynomial S such that $S(0) = \sigma$ and $S(j_f) = q_f$ for each $1 \leq f \leq t$. Since A's *view* (all his inputs) have a distribution independent of σ, the same is true of his output, and hence $\tau = \sigma$ with probability $1/m$.

Remark 4: Since the pieces of any t players have uniform distribution independent of the secret, this applies *a fortiori* to any smaller set of pieces. Formally, let $d \leq t$, and consider a set of d bad players $\{j_1, ..., j_d\}$, and their pieces $\{q_1, ..., q_d\}$. Let G denote the *first* $t-d$ good players (i.e., of smallest index), $\{g_1, ..., g_{t-d}\}$. For any σ, for any $(t-d)$-tuple of elements of Z_p $\{v_1, ..., v_{t-d}\}$ (corresponding to the pieces of G), there is a unique S such that $S(0) = \sigma$ and $S(j_f) = q_f$ for each $1 \leq f \leq d$ and $S(g_f) = v_f$ for each $1 \leq f \leq t-d$. Thus, for any secret, there are p^{t-d} polynomials consistent with the secret and the pieces of any d bad players; since all polynomials are equally likely to be chosen, the secret is independent of the pieces of the bad players.

As outlined, this scheme does not satisfy recoverability (1), due to the possibility of *dirty pieces* (i.e., incorrect evaluations of the polynomial). That is, even if the dealer properly shared a secret σ, bad players may interfere with good players trying to recover the secret. The interpolation used to recover the secret requires that all valuations of the polynomial are correct. If a

2 We let "$t-$th degree polynomial" denote a polynomial of degree at most t. Accordingly, there is a $1/p$ chance that the leading coefficient of S is 0.

single value is wrong, then the interpolation will give the wrong result. Good players, without knowing S, cannot distinguish correct values from incorrect values. Therefore, if bad players broadcast incorrect values, good players may use them in the interpolation and get the wrong result.

Remark 5: When $t<n/3$, the secret "shines through" the dirty pieces, but it may be infeasible to recover it. If the dealer properly gives shares of a polynomial S, then during *Recover*, at least $n-t$ good players broadcast pieces lying on S. Since at most t of these may lie on any other $t-$th degree polynomial, regardless of what the bad players broadcast, at most $2t$ pieces overall may lie on any other $t-$th degree polynomial Q. Since $n>3t$, S interpolates *more* than $2t$ of the pieces broadcast; this condition uniquely defines S as the correct polynomial, and hence the secret is well-defined as $S(0)$. Unfortunately, when $t=O(n)$, this polynomial may be hard to find. If a random set of $t+1$ players is chosen, the probability that all are good, in which case the interpolation returns the correct polynomial, vanishes exponentially in n. When $t \geq n/3$, the correct polynomial is not even determined by n pieces, t of which are incorrect, and hence no amount of computation can guarantee recovery of the secret.

If cryptography may be used, ***signatures*** could overcome the problem of dirty pieces. Assume we have schema for signing messages such that any player can verify whether a certain string represents h's signature of a message, but no player other than h can produce a valid $h-$signature of any message. Given such a scheme, a good dealer overcomes the problem of dirty pieces by signing all pieces of the secret. When the secret is to be recovered, any piece with an invalid signature is ignored; only good pieces, given and signed by the dealer, are used to recover the secret.

Our definition of secret sharing admits a problem limiting its possible applications, namely, ***non-existence of secret***: if the dealer is bad, the pieces he gives might not lie on a t-th degree polynomial. In this case, the secret is not even defined; any two sets of $t+1$ pieces, when interpolated, could yield different values for $S(0)$. Signatures would not help, since a bad dealer could properly sign dirty pieces. (Even if the dealer correctly terminates *Share*, but is corrupted before pieces are broadcast in *Recover*, he may give dirty, properly signed pieces to bad players, who then interfere with recovery of the secret.) The very goal of secret sharing - to commit the dealer to a value in advance - has been defeated.

3.2 Verifiable Secret Sharing (With Broadcast Channels)

Chor, Goldwasser, Micali, and Awerbuch [Chor, Goldwasser, Micali, and Awerbuch

1985] introduced the notion of verifiable secret sharing (VSS). This is an enriched secret sharing free of the problems mentioned above. It appends a *decision* protocol to the sharing protocol,[3] in which the players decide whether or not the secret is well-defined. The decision protocol must not enable the adversary to predict the secret better than random guessing. The recovery protocol must guarantee, with a high level of confidence, that any well-defined secret is recovered, no matter what the bad players do.

Definition 6: Let $P=(Share/Decide, Recover)$ be a jointly terminating compound protocol to be executed on a broadcast network, including as common parameters: the identity of the dealer, h; a confidence parameter, k; and the number of possible secrets, m. Each good player i outputs "Accept" or "Reject" at the end of *Share/Decide*; he outputs an element of $[0,m-1]$, a_i, at the end of *Recover*. P is a *VSS* if the following properties are satisfied:

(1) (Unanimity) All good players terminate *Share/Decide* with a common output (Accept/Reject).

(2) (Acceptance of good secrets) If the dealer correctly terminates *Share/Decide*, then all good players terminate *Share/Decide* by accepting.

(3) (Recoverability) Let E_σ be the event that every good player i that accepts in *Share/Decide* outputs σ in *Recover*. With probability at least $1-2^{-k}$, there exists a value $\sigma \in [0,m-1]$ such that E_σ is fixed at the end of *Share/Decide*.

(4) (Unpredictability) Let A be an adversary acting on P which never corrupts the dealer and outputs a value $\tau \in [0,m-1]$ (as its prediction of the secret) before the start of *Recover*. Then E_τ is fixed at the end of *Share/Decide* (i.e., A's prediction is correct) with probability $1/m$.

Remark 6: We have adopted the simplest definition of VSS which we can use. Technically, cryptographic VSS schemes do not satisfy our definition, since they must allow the adversary to predict the secret with a slight advantage. When the dealer correctly terminates *Share/Decide*, the event of property (3) (E_σ is fixed at the end of *Share/Decide* for some σ) is guaranteed to hold. Recall that all bounds on probabilities apply for any $r-$adversary A, for any fixed initial configurations; the probabilities are computed over all random tapes. By our convention, a VSS P is r-resilient if these properties are satisfied for any $r-$adversary acting on P.

[Chor, Goldwasser, Micali, and Awerbuch 1985] gave the first of various cryptographic VSS schemes. Based on a cryptographic assumption, they present a VSS with resiliency $(1/\log n)$. Benaloh [Benaloh 1986], Goldreich, Micali and Wigderson [Goldreich, Micali and Wigderson 1986], and Feldman [Feldman 1987] gave cryptographic 1/2-resilient VSS schemes.

3 From a technical standpoint, the sharing protocol is just longer, and has new properties. For clarity, we present the added steps as a separate protocol.

Chaum, Crepeau and Damgaard [Chaum, Crepeau and Damgaard 1988], and independently Ben-Or, Goldwasser and Wigderson [Ben-Or, Goldwasser and Wigderson 1988] found the first non-cryptographic VSS schemes, which tolerate up to $(n-1)/3$ bad processors.

Zero-knowledge proof techniques, as introduced by Golwasser, Micali and Rackoff [Golwasser, Micali and Rackoff 1985], may be used to ensure that the secret is well-defined. Benaloh [Benaloh 1986] was the first to design a VSS based on this observation; the protocols of [Chaum, Crepeau and Damgaard 1988] and [Ben-Or, Goldwasser and Wigderson 1988] are clever extensions of his method. The underlying idea of zero-knowledge proofs is captured by the following game.

Alice claims that she speeds to work, taking less than an hour on the thruway, *every* day; Bob suspects that Alice *never* spends less than an hour on the thruway. (We take it as given that one of the above is the case.) Bob asks her to save the time-stamped tickets she gets on entering and exiting the thruway. Alice is afraid that if Bob sees exactly how fast she goes, he won't let her borrow the car anymore, so she refuses to show him two tickets for the same day. Instead, she proposes the following game. Alice specifies an hour, and declares that both tickets are stamped in that hour. Bob may ask to see either ticket, entrance or exit, but not both. If the tickets are stamped over an hour apart, Bob has half a chance of catching her; on the other hand, if he doesn't catch her, he is not thoroughly convinced. Therefore, he asks to try again tomorrow (and the next day...). If Alice is telling the truth, she will never get caught, nor will Bob ever see how fast she went. If she is lying, Bob will almost certainly catch her fairly soon.

We play a similar game regarding a secret, which we identify with the polynomial used to share it (as in [Shamir 79]). Besides sharing the *main* secret S, the dealer shares pairs of *test* secrets (T_a, T_a+S). These have the property that if S is not properly shared, at least one of each pair of test secrets is not properly shared. The players randomly pick one test secret of each pair to be revealed. A good dealer "passes" without revealing anything about S; a bad dealer who did not share the main secret properly is caught with overwhelming probability.

[Ben-Or, Goldwasser and Wigderson 1988] show how a special case of Shamir's scheme overcomes the problem of dirty pieces using error correcting codes. We now give a simple version of their protocol.

3.3 A Simple VSS (Using Broadcast Channels)

We present a 1/4-resilient VSS, based on the protocol of [Ben-Or, Goldwasser and Wigderson 1988], which we call *SimpleVSS*. (In Section 5, we present a 1/3-resilient version; that protocol is self-contained, i.e., does not rely on error correcting code techniques.) We shall assume that the confidence parameter, k, is at least 4. *SimpleVSS* uses two additional common parameters:[4] a prime p, such that $p \geq m$ and $p \equiv 1 \bmod n$; and a primitive n-th root of unity mod p, w. (I.e., $w^n \equiv 1 \bmod p$, $w^d \not\equiv 1 \bmod p$ for $d < n$.) All calculations are done in Z_p.

Notation: When processor j is instructed to broadcast a message X, we may let X denote what is written on j's broadcast channel in that round, since it is the same for all processors. Such a broadcast is *proper* if it has the correct form (e.g., j was supposed to broadcast a message that round, X has the right length, etc.). Lowercase variables remain internal to a processor. Variables internal to processor i will *sometimes* carry the subscript i to facilitate the comparison of internal variables of different processors.

To simplify analysis of protocols, we assume that whenever processor i should perform an instruction for all j, this includes $j = i$. For example, when i sends a message to all players, he also sends a message to himself; any count of how many processors sent a certain message to i will include i (when applicable). A distinguished processor follows the code for all players in addition to his special code.

Any step of a protocol (e.g., the last) that consists solely of internal computation does not require a separate round, as it may be merged with the next round (e.g., of any subsequent protocol). Accordingly, such steps receive labels such as "Step 1.5".

Protocol *SimpleShare*

Common Parameters:

- n, the size of the network (in unary)
- t, an upper bound on the number of bad players($t < n/4$)
- h, the identity of a distinguished processor, the dealer
- k, the confidence parameter
- m, the number of possible messages
- p, a prime congruent to 1 mod n ($p \geq m$); all calculations are in Z_p
- w, an n-th root of unity mod p

Private Input for Every Player i: None

4 For our application, in which $m = n$, we may specify that the smallest possible values for p and w are to be used; each processor could compute these values itself. By a result of Linnik [Linnik 1944], the smallest possible p is bounded by a polynomial in n (if we assume the ERH, $p = O(n^2 \log n)$), hence p and w are polynomial-time computable functions of n.

Additional Code for Player h | Code for Every Player i

Step 1: Uniformly pick a $t-$th degree polynomial S such that $0 \leq S(0) < m$. Uniformly and independently pick $t-$th degree polynomials $T_{11}, \ldots, T_{1k}, T_{21}, \ldots, T_{nk}$. For each i, send $S(w^i), T_{11}(w^i), \ldots, T_{1k}(w^i), T_{21}(w^i), \ldots, T_{nk}(w^i)$ on the private channel to player i.

Step 1.5: Let $s, t_{11}, \ldots, t_{1k}, t_{21}, \ldots, t_{nk}$ denote the values privately received. (Set them to 0, if a proper message was not received.)

Protocol *SimpleDecide*

Step 1: Randomly pick k bits, $Q_{i1}, \ldots, Q_{ik}$. Broadcast $(i, Q_{i1}, \ldots, Q_{ik})$.

Step 2: For each j: if $(j, Q_{j1}, \ldots, Q_{jk})$ was not properly broadcast, set each $Q_{jf} = 0$. For each $1 \leq f \leq k$: if $Q_{jf} = 0$, set $P_{jf} = T_{jf}$; if $Q_{jf} = 1$, set $P_{jf} = T_{jf} + S$. Broadcast $(P_{11}, \ldots, P_{nk})$.

Step 3: Check that the dealer gave a proper broadcast. For each $(j, Q_{j1}, \ldots, Q_{jk})$ properly broadcast in Step 1: for each f, check that P_{jf} is a $t-$th degree polynomial, and that $P_{jf}(w^i) = t_{jf} + Q_{jf} \cdot s$. If all of these check, broadcast "Goodpiece"; and set $S_i = s$; otherwise, set $S_i = \emptyset$.

Step 3.5: If at least $n-t$ players broadcast "Goodpiece", output "Accept"; otherwise, output "Reject". Save S_i as a stored private input to *SimpleRecover*.

Protocol *SimpleRecover*

Common Parameters:

n, the size of the network (in unary)

t, an upper bound on the number of bad players($t < n/4$)

m, the number of possible messages

p, a prime congruent to 1 mod n $(p \geq m)$; all calculations are in Z_p

w, an n-th root of unity mod p

(Stored) Private Input for Every Player i: S_i, a value from *SimpleShare /Decide*

Code for Every Player i

Step 1: Broadcast (i,S_i).

Step 1.5: For each improper broadcast of (j,S_j), set $S_j=\emptyset$. Find (using standard error correcting code techniques) the unique $t-$th degree polynomial U interpolating at least $n-2t$ of the points (w^j,S_j); if none exists, set $U=0$. Let a_i be $U(0)$ reduced [5] mod m. Output a_i.

Theorem 1: SimpleVSS is a 1/4-resilient VSS.

Proof: Property (1), unanimity, holds because acceptance depends only upon the number of messages "Goodpiece"; these are broadcast, so all players receive the same number of them.

Property (2), acceptance of good secrets, holds because an uncorrupted dealer follows the protocol, so every good player broadcasts "Goodpiece".

We preface the proof of property (3), recoverability, by a brief discussion of error correcting code techniques [Peterson and Weldon 1982].

The most common application of error correcting codes arises when we wish to transmit ***codewords*** (messages of a special form) over an unreliable channel. Suppose that all possible codewords have c bits, and any two possible codewords differ in at least $2d+e+1$ bit positions. If at most d bits are switched (flipped) by the channel, and at most e bits are not received at all, then when we send a codeword, it is the ***closest*** codeword to the string actually received, i.e., it is the only codeword that can be derived by flipping at most d bits and filling in up to e more. Error correcting codes provide a polynomial time algorithm to find the closest codeword. A similar technique may be used when codewords consist of c-tuples of values in Z_p. If any two codewords differ in at least $2d+e+1$ values, then we may recover a codeword even if d of the values are incorrect and e are missing. For our application, $c=n$, $d=t$ and $e=t$. A codeword is an n-tuple of evaluations of a $t-$th degree polynomial at the n-th roots of unity. Any two codewords may agree in at most t positions, hence they disagree in at least $n-t$, which is at least $3t+1$. Therefore, the above technique may be used to recover a $t-$th degree polynomial from its values at the n-th roots of unity, if at most t of these values are incorrect and another t are missing.

5 A bad dealer may have picked S such that $S(0)\geq m$.

We refer to $(T_{jf}, S+T_{jf})$ as ***test secret pair jf***, and Q_{jf} as ***query bit jf***. If Q_{jf} is properly broadcast, we call $T_{jf}+Q_{jf}\cdot S$ the ***selected*** test secret jf.

Lemma 2: Let E be the event that

(a) The secret is accepted (i.e., by the good players), AND

(b) The main pieces (i.e., of the main secret) of all good players that broadcast "Goodpiece" do not lie on a $t-$th degree polynomial.

E occurs with probability at most 2^{-k}.

Proof: Let G be any fixed set of $n-2t$ players. Let U denote the minimum degree polynomial interpolating the main pieces of good players in G; let U_{jf} denote the minimum degree polynomial interpolating the pieces they received from the dealer for test secret pair jf. Suppose that the degree of U is greater than t.

Claim: The probability that all players in G are good and broadcast "Goodpiece" is at most $2^{-kn/2}$.

Proof: We may restrict ourselves to the case where all players in G are good. For every j and f, either U_{jf} or $U_{jf}+U$ (or both) is not a $t-$th degree polynomial. We say Q_{jf} is ***lucky*** if $U_{jf}+Q_{jf}\cdot U$ is a $t-$th degree polynomial. Say that a polynomial V ***fits***, or is ***consistent*** with, a piece t_{ij} held by player i if and only if $V(w^i)=t_{ij}$. If, during *SimpleDecide*, an unlucky query bit Q_{jf} is properly broadcast, then no matter what $t-$th degree polynomial the dealer reveals as P_{jf}, it does not fit the pieces of selected test secret jf of all members of G, and hence not all members of G broadcast "Goodpiece". Thus, either every properly broadcast query bit is lucky, or some member of G does not broadcast "Goodpiece". Each query bit picked by a member of G (or any good player) is picked randomly, and is lucky with probability 1/2 (or 0, if neither choice is lucky). Since over half of the players are in G, they pick (and properly broadcast) over half of the query bits; the chance that all are lucky is at most $2^{-kn/2}$. QED

We now observe that if the secret is accepted, at least $n-2t$ good players broadcast "Goodpiece". We have just shown that the chance that any fixed set of $n-2t$ players are good and broadcast "Goodpiece", even though their main pieces do not lie on a $t-$th degree polynomial, is at most $2^{-kn/2}$. We may overestimate the chance that such a set exists by multiplying by the number of sets of $n-2t$ players, $\binom{n}{n-2t}$. The product is $\binom{n}{n-2t}2^{-kn/2}<2^n 2^{-kn/2}=2^{n(1-k/2)}\leq 2^{-k}$ (for $n\geq 4$, $k\geq 4$). Finally, we observe that if E occurs, such a set must exist; this proves the lemma. QED

Say a player is ***satisfied*** if he is good and broadcasts "Goodpiece". Consider the case when the secret is accepted and (b) does not occur: that is, the main pieces of all satisfied players lie on a $t-$th degree polynomial. Since there are at least $n-2t$ satisfied players, there is a unique such

$t-$th degree polynomial, U. In this case, in *SimpleRecover*, at most t of the non-null values broadcast may be incorrect (i.e., those of the bad players); additionally, up to t good unsatisfied players may broadcast $\emptyset$. Therefore, the error correcting technique is guaranteed to return U for every good player. To finish the proof of (3), we observe that $\sigma = U(0)$ is determined at the end of *SimpleShare/Decide*.

For property (4), unpredictability, we consider the adversary's view on *SimpleShare/Decide*. This consists of the main pieces of bad players, test pieces of bad players, query bits, revealed test polynomials, and "Goodpiece" messages. Consider any set of t players $\{j_1,...,j_t\}$, any ordered t-tuple of elements of Z_p representing their main pieces, $\{s_{j_1},...,s_{j_t}\}$, any set of query bits $Q_{11},...,Q_{nk}$, and any set of $t-$th degree polynomial representing the revealed test polynomials $P_{11},...,P_{nk}$. For any possible secret σ, there is a unique polynomial S consistent with the main pieces of these t players such that $S(0)=\sigma$. S, in conjunction with A's view, determines each shared test polynomial, $T_{jl}=P_{jl}-Q_{jl}\cdot S$, which determines all test pieces. All good players broadcast "Goodpiece". It follows that for any σ, there is exactly one choice of polynomials $S,T_{11},...,T_{nk}$ which is consistent with A's view. Since all polynomials are chosen uniformly and independently, all secrets are equally likely, given A's view, so A's prediction is independent of σ. (As before, if $d \leq t$ players were corrupted, p^{t-d} possible $K+1$-tuples of polynomials correspond to each possible secret, so all are equally likely.) QED

4. Our BA Protocol

We are ready to build the BAP. We first present a simple primitive which gives a weak simulation of a broadcast channel. Using this primitive, we design a protocol with virtually the same properties as VSS. We define a *common coin*, and give a rather involved reduction from a common coin to our adaptation of VSS. Finally, we show that BA is reducible to a common coin.

4.1 A Graded-broadcast Primitive

A broadcast channel is a very useful feature in a network; a processor receiving a message on a broadcast channel is guaranteed that all other processors are receiving the same message. Broadcast channels may be used for much more than VSS; in particular, it is trivial to reach BA in a broadcast network, e.g., each processor broadcasts his input value, then outputs the majority value broadcast (with, say, default 0 in case of a tie). We shall define a *Graded-broadcast* primitive, which is nearly as powerful as a broadcast channel, and shall use it as a stepping stone towards BA.

Crusader agreement, as introduced by Dolev [Dolev 1982] and refined by Turpin and Coan [Turpin and Coan 1984], simulates an unreliable broadcast channel. Basically, sending a message using Crusader agreement is like sending a message on a broadcast channel in which some messages may not get delivered: recipients of the message are guaranteed that all other recipients receive the same messages. However, recipients have no guarantee that any other player receives the message. We introduce *Graded- broadcast* to partially resolve this problem. As in Crusader agreement for any message sent using *Graded- broadcast*, all received messages are the same. The added feature is that when a good player sends a message using *Graded- broadcast*, all good players can verify that all good players receive the same message.

Definition 7: Let P be a jointly terminating protocol including as a common parameter h, the identity of a distinguished processor, the *sender*. Only the sender has a private input, σ. Each good player i outputs a pair $(code_i, value_i)$, where $code_i$ is 0, 1, or 2. We say that P is a *Graded- broadcast* if the following conditions are satisfied:

(1) If i and j are good players, and $code_i \neq 0$ and $code_j \neq 0$, then $value_i = value_j$.

(2) If i and j are good players, then $|code_i - code_j| \leq 1$.

(3) If the sender is good, then for every good player i, $code_i = 2$ and $value_i = \sigma$.

Notation: Let *distribute* denote an instruction to send a particular message to all processors. (As mentioned earlier, a player sends messages to himself and counts himself in all tallies.)

Protocol *Grade- Cast*

Common Parameters:

n, the size of the network (in unary)

t, an upper bound on the number of bad players($t < n/3$)

h, the identity of a distinguished processor, the sender

Extra Private Input for Player h : σ, a binary string

Private Input for Every Player i: None

Additional Code for Player h

Step 1: Distribute σ.

Code for Every Player i

Step 2: Let V_i denote the Step 1 message received from h. Distribute V_i.

Step 3: If a common string Z was received as the Step 2 message of at least $n - t$ players, distribute Z; otherwise, send no messages.

Step 3.5: Let $tally_x$ denote the number of players that sent x to i in Step 3.

a) If for some x, $tally_x \geq 2t+1$, set $value_i = x$ and $code_i = 2$.

b) If for some x, $2t \geq tally_x > t$, set $value_i = x$ and $code_i = 1$.

c) If for all x, $tally_x \leq t$ set $value_i = \emptyset$ and $code_i = 0$.

Output $(code_i, value_i)$.

Theorem 3: *Grade- Cast* is a 1/3-resilient *Graded- broadcast* protocol.

Proof: We first show that $value_i$ is well defined in Step 3.5 . The key to proving this is showing that if g and j are good players that distribute X and Y respectively at Step 3, then $X=Y$. We show this by a simple counting argument. Assume that at least $n-t$ players send X to g in Step 2. Suppose w of these players are bad, where $w \leq t$, so at least $n-t-w$ good players send X in Step 2. Therefore, at most $(n-w)-(n-t-w)=t$ good players can distribute any other value at Step 2. Thus, if $X \neq Y$, at most t good players and hence at most $2t$ players overall send Y to j in Step 2. Since $t<n/3$, $2t<n-t$, so if j is good, he does not distribute Y in Step 3.

We have shown that all good players that distribute in Step 3 are distributing a common message X. Thus, any $Y \neq X$ may be distributed only by bad players, and hence $value_i$ is well defined in Step 3.5. Moreover, any good i that decides according to 3.5a or 3.5b will set $value_i = X$, proving property (1) of Definition 7. Property (2) follows from the fact that if any good player i sets $code_i = 2$, then at least $2t+1$ players send X to i in Step 3, and at least $t+1$ of these players are good, so all good players receive X from at least $t+1$ players, so they all decide according to 3.5a or 3.5b. Property (3) is easily verified, since if h is good, all good players receive and redistribute σ in Steps 2 and 3. QED

Notation: In *Grade- Cast*, we say that i *accepts* if $code_i = 2$; i *semi- accepts* if $code_i = 1$; i *rejects* if $code_i = 0$. We say i *acknowledges* if $code_i > 0$. We shall use this notation for all protocols in which a graded, three-level decision is made.

We shall let *Grade- Cast* denote an execution of *Grade- Cast*; an instruction for player i to *Grade- Cast* V is equivalent to saying that the network runs *Grade- Cast* with common parameter i, in which i's private input is V.

In general, when a protocol instructs h to distribute V, it would be misleading to let V denote the message i receives, since this suggests that every good player receives the same value. When h is instructed to broadcast V (on a broadcast channel), we have seen that this notation is justified, since we may define V to be whatever h broadcasts. When a protocol instructs h to *Grade- Cast* V, we shall likewise let V denote the unique non-null output $value_i$ of any good player i (if one exists). A player rejecting the *Grade- Cast* does not output this value (but rather $\emptyset$), but such a player never accesses V in any case.

4.2 From (One-Bit) BA to General (Multi-Bit) BA

In the general BA problem, each processor holds an arbitrary input value. Standard Crusader agreement reduces the general BA problem to BA as defined in Definition 4 (reaching agreement when all inputs are bits). The reduction is even simpler using (all but the first step of) *Grade- Cast*.

Definition 8: Let P be a protocol in which each processor i has a private input, σ_i. Each i that correctly terminates outputs a value, α_i. P is a multi-bit BA protocol if

(1) For any i and j that correctly terminate, $\alpha_i = \alpha_j$.

(2) If $\sigma_i = \sigma_j$ for all i and j uncorrupted at the start of P, then $\alpha_i = \sigma_i$ for every correctly terminating i.

Theorem 4: ([Dolev 1982], [Turpin and Coan 1984]) Let P be a BA protocol running in (expected) time $F(n)$ on a network of size n. Then there exists a multi-bit BA protocol Q running in (expected) time $F(n)+O(1)$. If P is $r-$resilient, then the resiliency of Q is $min(1/3,r)$.

Proof: Consider the following protocol for multi-bit BA.

Protocol Q

Common Parameters:

n, the size of the network (in unary)

t, an upper bound on the number of bad players($t<n/3$)

Private Input for Every Player i: σ_i, a binary string

Code for Every Player i

Step 1: Execute steps 2,3, and 3.5 of *Grade- Cast*, setting $V_i = \sigma_i$ in Step 2; let $(code_i, value_i)$ denote the private output. If $code_i = 2$, set $B_i = 1$; otherwise, set $B_i = 0$.

Step 2: Run P, using B_i as private input. Let d_i be the private output.

Step 2.5: If $d_i = 1$, then set $\alpha_i = value_i$; otherwise, $\alpha_i = \emptyset$. Output α_i.

The key to the proof is the observation that the properties regarding the outputs of *Grade- Cast* apply for any sender, in particular, a sender that sends σ_i to player i.

If the good processors output 1 in P, then some good processor j must start P with $B_j = 1$. This implies that $code_j = 2$, so all processors that correctly terminate Q_P acknowledge the same message in *Grade- Cast* and output it in Q_P. If all processors that correctly terminate Q_P start with the same input to *Grade- Cast*, x, they all end *Grade- Cast* with output $(2,x)$. Therefore, they all start P with bit 1, hence they all end P with bit 1, and output x in Q_P. QED

4.3 Graded Verifiable Secret Sharing

Our definition of VSS requires that all good processors reach a common decision regarding the acceptability of a secret; for this, broadcast channels (or BA) are needed. Obviously, we cannot use VSS as a tool to reach BA! Fortunately, we find that a weaker version of VSS which does not require broadcast channels is sufficient to help reach BA in constant expected time. We call this weaker version Graded-VSS, for in it, the players reach a graded decision (as in *Grade-Cast*). A player may accept, semi-accept, or reject the secret. A player that semi-accepts has concluded that the dealer is bad, but nevertheless knows that the secret is recoverable. Graded-VSS retains all the properties of VSS, with the exception of unanimity, which is weakened to semi-unanimity: if any good player accepts, all good players acknowledge. (We highlight this one difference by calling this property 1'.)

Definition 9: Let P=(*Graded-Share/Decide*, *Graded-Recover*) be a jointly terminating compound protocol including as common parameters: the identity of the dealer, h; a confidence parameter, k; and the number of possible secrets, m. Each good player i outputs $accept_i \in \{0,1,2\}$ at the end of *Graded-Share/Decide*, and outputs $a_i \in [0, m-1]$ at the end of *Graded-Recover*. We call P a *Graded-VSS* if the following properties are satisfied:

(1') (Semi-unanimity) If any good player i outputs $accept_i = 2$ in *Graded-Share/Decide*, then $accept_j > 0$ for every good player j.

(2) (Acceptance of good secrets) If the dealer correctly terminates *Graded-Share/Decide*, then each good player j outputs $accept_j = 2$.

(3) (Recoverability) Let E_σ be the event that every good player i that sets $accept_i > 0$ outputs σ in *Graded-Recover*. With probability at least $1-2^{-k}$, there exists a value $\sigma \in [0, m-1]$ such that E_σ is fixed at the end of *Graded-Share/Decide*.

(4) (Unpredictability) Let A be an adversary acting on P which never corrupts the dealer and outputs a value $\tau \in [0, m-1]$ (as its prediction of the secret) before the start of *Graded-Recover*. Then E_τ is fixed at the end of *Graded-Share/Decide* (i.e., A's prediction is correct) with probability $1/m$.

The Graded-VSS protocol we exhibit below is based on SimpleVSS. However, there are two key differences:

(1) The employment of *Grade-Cast* in place of broadcasts.

(2) The branched responses to *Grade-Casts* in the decision protocol.

We briefly discuss these changes. The power of broadcast channels guaranteed that for any good processors h and i, for every broadcast received by i, h received the same broadcast. For any broadcast requiring a specific response -- for example, in *SimpleDecide*, the dealer reveals test polynomials corresponding to the broadcast of query bits -- if h does not give the proper response, i concludes that h is faulty. What happens without broadcast channels?

We may be tempted to try substituting, in place of an instruction to broadcast V, an instruction to distribute V, followed by a Crusader agreement on the value distributed. Although this would guarantee that i would not receive a different value than h, it is possible that i would receive a value, and h would receive nothing. Thus, a non-response from h proves nothing; this attempted simulation fails. The substitution of *Grade–Cast* instructions, with branched responses, will succeed; the three-level acceptance code is the crucial feature. It guarantees that any *Grade–Cast* **accepted** by i is **acknowledged** by h. Consider any response h is required to make to a proper broadcast in VSS. If, in *Graded–VSS*, the corresponding *Grade–Cast* is accepted by any good player i, then h acknowledges the same *Grade–Cast*, and makes the appropriate response. A *Grade–Cast* semi-accepted by i did not come from a good player, and hence h (who may not have acknowledged such a *Grade–Cast*) is not penalized for not responding.

We employ this substitution in *SimpleVSS* and get *SimpleGraded–VSS*. *SimpleShare*, which does not use broadcasts, is unchanged. Although pieces are broadcast in *SimpleRecover*, it suffices to distribute them. The essential changes are in the decision protocol. There are three reasons why we only have semi-unanimity. Firstly, players may differ slightly on the acceptability of *Grade–Casts* of the dealer. Secondly, players may disagree on whether or not a query vector was acceptably *Grade–Cast*. Thirdly, players may disagree on which players distributed "Good-piece". This substitution technique also works for the 1/3-resilient VSS we present in Section 5.[1]

Protocol *SimpleGraded–Share*

Common Parameters:

- n, the size of the network (in unary)
- t, an upper bound on the number of bad players($t < n/4$)
- h, the identity of a distinguished processor, the dealer
- k, the confidence parameter

1 More generally, this substitution works for every VSS scheme in the literature. We believe that this technique may be used to convert any VSS to a Graded-VSS. We do point out that in theory, a VSS scheme *might* require the full power of broadcast channels or BA. Although one may contrive a protocol to embed BA, we cannot conceive of a case where this is needed to achieve properties (1′), (2), (3) and (4).

m, the number of possible messages

p, a prime congruent to 1 mod n ($p \geq m$); all calculations are in Z_p

w, an n-th root of unity mod p

Private Input for Every Player i: None

Additional Code for Player h | Code for Every Player i

Step 1: Uniformly pick a t-th degree polynomial S such that $0 \leq S(0) < m$. Uniformly and independently pick t-th degree polynomials $T_{11}, \ldots, T_{nk}$. For each i, send $S(w^i), T_{11}(w^i), \ldots, T_{nk}(w^i)$ on the private channel to player i.

Step 1.5: Let $s, t_{11}, \ldots, t_{nk}$ denote the values privately received.

Protocol *SimpleGraded-Decide*

Step 1: Randomly pick k query bits, $Q_{i1}, \ldots, Q_{ik}$. *Grade-Cast*$(i, Q_{i1}, \ldots, Q_{ik})$.

Step 2: For each j: if the *Grade-Cast* of $(j, Q_{j1}, \ldots, Q_{jk})$ was **acknowledged**, for each $1 \leq f \leq k$, set $R_{jf} = Q_{jf}$; otherwise, set $R_{jf} = 0$. Let $P_{jf} = T_{jf} + R_{jf} \cdot S$. *Grade-Cast* $P_{11}, \ldots, P_{nk}$.

Step 3: Check that h's *Grade-Cast* is accepted and proper. For each proper, **accepted** *Grade-Cast* of $(j, Q_{j1}, \ldots, Q_{jk})$ (from Step 1), for each $1 \leq f \leq k$, check that P_{jf} is a t-th degree polynomial, and $P_{jf}(w^i) = t_{jf} + Q_{jf} \cdot s$. If these all check, distribute "Goodpiece" and set $S_i = s$; otherwise, set $S_i = \emptyset$.

Step 4: If at least $n-t$ messages "Goodpiece" were received, distribute "Passable"

Step 4.5: (a) If at least $2t+1$ messages "Passable" were received, set $accept_i = 2$.

(b) If between $t+1$ and $2t$ messages "Passable" were received, set $accept_i = 1$.

(c) Otherwise, set $accept_i = 0$.

Output $accept_i$. Save S_i as stored private input to *SimpleGraded-Recover*.

Protocol *SimpleGraded-Recover*

Common Parameters:

n, the size of the network (in unary)

t, an upper bound on the number of bad players($t<n/4$)

m, the number of possible messages

p, a prime congruent to 1 mod n ($p \geq m$); all calculations are in Z_p

w, an n-th root of unity mod p

(Stored) Private Input for Every Player i: S_i, a value from *SimpleGraded-Share/Decide*

Code for Every Player i

Step 1: Distribute (i,S_i).

Step 1.5: Let the set of received pieces be $\{(1,d_1),...,(n,d_n)\}$, where $d_j = \emptyset$ if nothing was received from player j. Find the unique t-th degree polynomial U interpolating at least $n-2t$ of the points (w^j,d_j) (if none exists, $U=0$). Let a_i be $U(0)$ reduced mod m. Output a_i.

Theorem 5: SimpleGraded-VSS is a 1/4-resilient Graded-VSS.

Proof: Property (1′) (semi-unanimity) is immediate, since if any good player accepts, he receives at least $2t+1$ "Passable"s, so all good players receive at least $t+1$ "Passable"s, so they all acknowledge.

For properties (2), (3), and (4), we can apply the same proof used for SimpleVSS, by checking that the substitution of *Grade-Casts* for broadcasts suffices. For property (2) (acceptance of good secrets), the crucial fact is that if h and i correctly terminate *SimpleGraded-Share/Decide*, any proper *Grade-Cast* of $(j,Q_{j1},...,Q_{jk})$ accepted by i is acknowledged by h, and hence properly answered by h, so i distributes "Goodpiece". Since at least $n-t$ players are good, each good player distributes "Passable".

In the proof of property (3) (recoverability) for *SimpleVSS*, we only used the fact that if the dealer fails to properly respond to a query bit of a good player, then no good player sends "Goodpiece". This is also true for *SimpleGraded-VSS*, since all good players accept the *Grade-Cast* of the query bits of every good player. *SimpleRecover* does not rely on the fact that pieces are broadcast, just the fact that at most $2t$ received pieces are incorrect, of which at most t

are non-null. Thus, the same argument shows that with probability at least $1-2^{-k}$, either the secret is rejected by all good players, or it is fixed at the end of *Graded-Share/Decide*.

Property (4) (unpredictability) still holds, since, as in *SimpleVSS*, the view of any $r-$ adversary is determined by values with a uniform distribution independent of $S(0)$.

Notation: We shall refer to an execution of *Graded-Share/Decide* as a *stored secret*. A stored secret is *passable* if at least one good processor acknowledges. A stored secret is *recoverable* if, for some $\sigma \in [0,m-1]$, E_σ is fixed at the end of *Graded-Share/Decide*, i.e., no matter what the bad players do, every good player that acknowledges the secret outputs σ in *Graded-Recover*.

4.4 A Common Coin Protocol

[Rabin 1983] showed that the problem of reaching BA is reducible to finding a *common coin protocol*. Intuitively, a common coin may be viewed as a random, unpredictable bit which, somehow, suddenly becomes a common input of all players. For example, the parity of the Monday's market closing (Dow Jones average) is reasonably unpredictable on Monday morning, but anyone may read it in Tuesday's newspaper. This bit need not be perfectly random and unpredictable, as indicated by the following definition.

Definition 10: Let P be a jointly terminating protocol in which each good player i outputs a bit d_i. We say that P is a *common coin protocol* if for a constant $p>0$, for any $b \in \{0,1\}$, with probability at least p, $d_i=b$ for all good processors i. (We say P is $p-fair$.)

We call an execution of P a *common coin*; the common coin is *unanimously* b if $d_i=b$ for every good player i.

(In Section 4.5, we will present a reduction from BA to a common coin protocol.)

How may one design a common coin protocol? Our method will be to consider the tallies in a "random election". Random elections of a different flavor were used in the agreement protocols of Chor, Merritt and Shmoys [Chor, Merritt and Shmoys 1985] and Dwork, Shmoys and Stockmeyer [Dwork, Shmoys and Stockmeyer 1986] to obtain a common coin in "more benevolent" scenarios.

In the election, each player is both a voter and a candidate. Unlike standard elections, a candidate chooses which votes he wishes to accept. Each player uses Graded-VSS to share n secret *votes*, one for each candidate, chosen randomly and independently mod n. Each candidate must accept (both in the formal and informal sense!) a set of at least $n-t$ votes. Subsequently, all secrets are recovered. The *tally* for j is the sum of the votes j accepted, reduced mod n. If some candidate has tally 0, the common coin is 0; if no candidate has tally 0, the output bit is 1.

Remark 7: If a tally includes unrecoverable votes, it may be undefined. Therefore, any candidate accepting non-passable votes is *scratched* during *Vote*. This is the utility of the three-level acceptance. Every vote **accepted** by a good candidate is **acknowledged** by all good players, so each good player knows that the vote is passable.

A bad player may select the values of his votes non-randomly. However, each candidate must accept at least $n-t$ votes, some of which were shared by good players and hence were picked randomly, hence the tally of every candidate is random.

Remark 8: If each of k independent random variables (in our application, tallies) has probability q/n of being 0, then the probability that none are 0 is $(1-q/n)^k$, which approximates $e^{-kq/n}$ for large n. We shall state, without proof, bounds on such quantities valid for any $n \geq 4$.

Theorem 6: Let (*Graded-Share/Decide*, *Graded-Recover*) be a Graded-VSS running in time $F(n)$ on a network of size n. Then there exists a .27-fair common coin protocol running in time $F(n)+O(1)$. If the Graded-VSS is r-resilient, then the common coin is $min(1/3,r)$-resilient.

Consider the following protocol. We consider as a common parameter $k = \log 10n^2$ (log denotes logarithm to the base 2).

Protocol *Vote*

Common Parameters:

n, the size of the network (in unary)

t, an upper bound on the number of bad players($t<n/3$)

$k = \log 10n^2$, the confidence parameter

Private Input for Every Player i: None

Code for Every Player i

Step 1: For each $1 \leq h \leq n$, for each $1 \leq j \leq n$, run *Graded-Share/Decide*, specifying as common parameters: $h=$ dealer; $n=$ the number of possible secrets; $k=$ confidence parameter; and j as an execution label (denoting that this vote is for j; we shall refer to this execution as secret h,j). Let $accept_{hj}$ denote the output of *Graded-Share/Decide*. (Note: *Graded-VSS* surely utilizes stored private inputs, but they are only accessed by *Graded-Recover*, hence they do not appear in the code for *Vote*; we treat *Graded-VSS* as a black box satisfying Definition 9.)

Step 2: Let A_{ji} denote $accept_{ji}$ for each j. (Note: by convention, lowercase variables remain internal). *Grade-Cast* the *list* $(A_{1i},\ldots,A_{ni})$.

Step 2.5: For each j, if j's *Grade-Cast* is accepted, and $(\max_h (A_{hj} - accept_{hj}) < 2)$ and $(A_{hj}=2$ for at least $n-t$ values of $h)$, set $candidate_j = 1$; otherwise, set $candidate_j = 0$.

Step 3: For each h and j, run *Graded- Recover* on secret h,j. Let v_{ihj} be the output.

Step 3.5: For each j such that $candidate_j=0$, set $tally_{ij}=\emptyset$. For each j such that $candidate_j=1$, set $tally_{ij}=\left(\sum_{h:\,A_{hj}=2} v_{ihj}\right) \bmod m$. If $tally_{ij}=0$ for some j, set $d_i=0$; otherwise, set $d_i=1$. Output d_i.

We first give an overview of how the election works. The *Grade- Cast* of i's *list* indicates which votes are to be included in i's tally, namely, each secret h,i for which $A_{hi}=2$. A good player i ***keeps*** candidate j by setting $candidate_j=1$ if three conditions are met:

(1) i accepts the *Grade- Cast* of j's list;

(2) i acknowledges every secret that j accepts;

(3) j accepts at least $n-t$ votes.

If any of these fail, i ***scratches*** candidate j by setting $candidate_j=0$. We say that j is ***totally*** scratched if all good players scratch j; otherwise, j is ***in the running***. If j is in the running, then j's list accepts at least $n-t$ votes, all of which are passable (acknowledged by a good player), and all good players acknowledge the *Grade- Cast* of j's list.

For every pair of good players (i,j), i accepts j's *Grade- Cast*, j accepts all secrets dealt by good players, and i acknowledges every secret that j accepts, so i sets $candidate_j=1$ (i.e., i keeps j).

Intuitively, the tally of every candidate in the running should be a sum of *recoverable* secrets (i.e., the value was fixed at the end of *Graded- Share/Decide*). Each such tally includes a vote by a good player; accordingly, all tallies are random, independent of each other, and unpredictable to the adversary. The probability that any particular player in the running has tally 0 is $1/n$. Since a good candidate is kept by all good players, if any good player has tally 0, all good players output 0 as the common coin. If no player in the running has tally 0, then all good players output 1 as the common coin. Since there are at least $2n/3$ good players, and at most n players may be in the running, each of these events has probability exceeding .27. If j, a bad candidate in the running, has the only tally 0, the coin may not be unanimous. In this case, good players that keep j output 0, and good players that scratch j output 1.

Reality is a fair approximation to this intuition. There is a small chance that a passable secret is not recoverable; the adversary may be able to affect a tally including such a secret even after secret recovery has begun. We must show that the adversary's ability to affect and predict tallies is small enough to guarantee that the probability that the coin is unanimously 0 (1) remains large enough.

Proof of Theorem 6: We show that *Vote* has the desired properties. Let A be any $r-$adversary acting on *Vote*. Let E be the event that some passable secret is not recoverable.

Claim 1: E occurs with probability at most .1.

For each h,j, let E_{hj} be the event that secret h,j is passable but not recoverable. By property (3) of Graded-VSS, the probability of E_{hj} is at most $2^{-k} \leq 1/10n^2$. The probability that this happens for at least one of the n^2 secrets shared in *Vote* is at most $n^2 \cdot 1/10n^2 = .1$.

Claim 2: When $\bar{E}$ holds (i.e., E does not occur), for every j, there exists a value $tally_j$ such that for each good player i, $tally_{ij}$ is either $tally_j$ or $\emptyset$.

Any good player g that scratches j sets $tally_{gj}=\emptyset$. All good players that keep j acknowledge a common *list* for j, $A_{1j},...,A_{nj}$. If j is not totally scratched, then every secret j accepts is passable; given $\bar{E}$, every passable secret is recoverable. Therefore, for each h,j such that $A_{hj}=2$, there exists a value v_{hj} such that $v_{ihj}=v_{hj}$ for every good player i. If i keeps j, then $tally_{ij}=\left(\sum_{k:\, A_{kj}=2} v_{ikj}\right) \bmod n = \left(\sum_{k:\, A_{kj}=2} v_{kj}\right) \bmod n$; this sum is the same for all good processors i that keep j.

Suppose that the adversary can arrange that for some player j in the running, the probability that $tally_j$ is fixed to be 0 at the start of *Recover* exceeds $1/n$. Notice that $tally_j=0$ if and only if the first vote j accepts from a good dealer is the negative of the sum of all other votes j accepts. Thus, knowledge of all other secrets would permit A to correctly predict a good secret with probability exceeding $1/n$, which is impossible (by property (4) of Graded-VSS). We actually must show that each tally has probability $1/n$ of being 0, independently of all other tallies.

Let $j_1,...,j_a$ denote all players in the running in increasing order of identities. For $0 \leq f \leq a$, let E_f denote the event that $tally_{j_d} \neq 0$ for all $1 \leq d \leq f$. (Note: E_0 is true by tautology.) For $0 \leq d \leq a$, let p_d be the probability of E_d; for $1 \leq d \leq a$, let $c_d = p_d / p_{d-1}$, the conditional probability of E_d given E_{d-1}.

Claim 3: Assume $\bar{E}$ holds. Then for each $1 \leq f \leq a$, $c_f = 1 - 1/n$.

Let $j=j_f$. Since at least $n-t$ votes are accepted by j's list, there is a good player h such that $A_{hj}=2$. The unpredictability property of Graded-VSS guarantees that for any $r-$adversary A that predicts v_{hj} before *Graded-Recover* is run on secret h,j, the prediction is correct with probability $1/n$. In particular, unpredictability still applies even if the network runs *Graded-Recover* for every secret g,i except for h,j.

To make our argument precise, we define a (contrived) protocol *Vote'* as follows. The only difference between *Vote'* and *Vote* is that in *Vote'*, if player i receives a message "Delay (h,j)" sent by any processor in Step 2, i does not run *Graded-Recover* for secret (h,j). (We call *Vote'* contrived because no good player sends a "Delay" message; the code anticipates and invites a specific faulty behavior.) An adversary A acting on *Vote'* can easily arrange that all secrets except h,j are recovered in Step 3, by having a bad player distribute "Delay (h,j)" in Step 2. A can compute all tallies other than j's, and $x = v_{hj} - tally_j$, in Step 3 of *Vote'*.

Vote and *Vote'* are identical through Step 2. All tallies are determined by the end of Step 2 in *Vote*, and hence in *Vote'* as well. Thus, if A acts the same through Step 2 in both, the probabilities of E_f are the same in both cases. We proceed by induction.

Let S_f be the statement: $c_d = 1 - 1/n$ for $1 \leq d \leq f$. S_0 is true by tautology. For $1 \leq f \leq a$, $p_d = \prod_{e=1}^{d} c_e$, so if S_{f-1} is true, then $p_{f-1} = (1 - 1/n)^{f-1} > 0$. A acting on *Vote'* may predict v_{hj} as follows.

(1) When E_{f-1} occurs, A outputs x. Notice that $(x = v_f) \Leftrightarrow (tally_f = 0) \Rightarrow E_f$ does not occur. Given E_{f-1}, E_f is equivalent to the event $x \neq v_{hj}$; we denoted the conditional probability of this event by c_f.

(2) When E_{f-1} does not occur, A outputs a random number mod n; this prediction is correct with probability $1/n$.

Overall, A's prediction is correct with probability $(p_{f-1})(1 - c_f) + (1 - p_{f-1})(1/n) = 1/n + (p_{f-1})(1 - 1/n - c_f)$. By unpredictability (property (3) of Graded-VSS), this probability is $1/n$. Since $p_{f-1} > 0$, $c_f = 1 - 1/n$. Thus, $S_{f-1} \Rightarrow S_f$; since S_0 is true, so are $S_1, \ldots, S_a$, proving the claim.

Thus, when $\bar{E}$ holds, the probability that no candidate in the running has tally 0 is at least $(1 - 1/n)^a$. Since $a \leq n$, $p_a \geq (1 - 1/n)^n > .3$. When this happens, all good players output 1. The probability that no good candidate has tally 0 is at most $(1 - 1/n)^{n-t}$; since $t < n/3$, this is at most $(1 - 1/n)^{2n/3} < .7$. Thus, the probability that some good player has tally 0 is at least .3. When this happens, all good players output 0. Finally, we recall that $\bar{E}$ holds with probability at least .9, so the coin is .27-fair. QED

4.5 BA from Common Coins

[Rabin 1983] shows that BA is constant-expected-time reducible to a common coin protocol. The following theorem adapts the ideas of [Rabin 1983] and subsequent papers to our scenario.

Theorem 7: Let P be a p-fair common coin protocol. Then there exists a jointly terminating protocol Q_P with the following properties:

(1) Each good player i has a private input bit B_i and outputs a bit b_i.

(2) For any $b \in \{0,1\}$, if $B_i = b$ for each good player i, then $b_i = b$ for each good player i.

(3) With probability at least p, there exists $b \in \{0,1\}$ such that $b_i = b$ for each good player i.

(4) The running time of Q_P exceeds that of P by 1 round. Also, if P is r-resilient, then the resilience of Q_P is $min(1/3, r)$.

Effectively, an execution of Q_P reaches BA with probability at least p; if the network starts Q_P in agreement, it stays in agreement.

Proof: Consider the following protocol. Each good player i has a private input bit B_i and outputs a bit b_i.

Protocol Q_P

Common Parameters:

n, the size of the network (in unary)

t, an upper bound on the number of bad players($t < n/3$)

Private Input for Every Player i: B_i, a bit

Code for Every Player i

```
Step 1: Distribute $B_i$.

Step 2: For each $j$, let $c_j$ be the bit received from $j$ in Step 1 ($c_j=0$, if a proper message was
    not received). Let $count_i$ denote the number of $j$ such that $c_j=1$. Run $P$; let $d$ be the out-
    put.
        (a) If $count_i \leq t$ then set $b_i=0$.
        (b) If $count_i > 2t$ then set $b_i=1$.
        (c) If neither (a) nor (b), set $b_i=d$.
    Output $b_i$.
```

We first observe that if $B_i=0$ for all good players i, then for every good player j, $count_j \leq t$, hence j sets $b_j=0$ in Step 2a. Likewise, if $B_i=1$ for all good players i, then for every good player j, $count_j \geq n-t$, hence j sets $b_j=1$ in Step 2b. This proves property (2). Notice that for any $B_1,\dots,B_n$, for any good players i and j, $count_i - count_j$ is at most the number of (necessarily bad) players that send the bit 1 to i and 0 to j, and this is at most t. Therefore, if i executes Step 2b, j cannot execute Step 2a. Thus, all good players that execute Step 2a or 2b execute the same substep, and output a common bit b. This b is determined by the messages sent at Step 1, before the start of P. With probability at least p (taken over random bits read during P), all good processors end P with output b, in which case all good processors end Q_P with output b. (If no good processor executes 2a or 2b, then all good processors end Q_P with a common bit if and only if the coin is unanimous, which has probability at least $2p$.) This proves property

(3). Properties (1) and (4) follow by inspection. QED

We have shown how to use a common coin to protocol to reach BA with a positive, constant probability; moreover, if the network starts in agreement, it stays in agreement. We may iterate this procedure: namely, the processors repeatedly run Q_P, using their outputs of the a-th execution as their private inputs to the $a+1$-st execution. If the good players reach agreement in any iteration, agreement is always maintained. Let the expected number of iterations until agreement is reached be X. Since the probability that disagreement is maintained on any particular iteration is at most $1-p$, *independently* of the fact that no previous iteration reached agreement, $X \leq 1+(1-p)X$, so $X \leq 1/p$.

We have shown that *indefinitely* iterating Q_P brings the network into agreement in a constant expected number of iterations. After any finite number of iterations, there is a non-zero probability that the network is still not in agreement; thus, this approach would not allow for processors to terminate. We can enable processors to stop iterating Q_P by utilizing *proofs of agreement*. Basically, we add steps to the protocol enabling processors to deduce that the network has reached agreement, at which point they terminate.

We recall the defining properties of a BA protocol.

(1) For any i and j that correctly terminate, $d_i = d_j$.

(2) If $b_i = b_j$ for all good players i and j, then for each i that correctly terminates, $d_i = b_i$.

Theorem 8: Let P be a common coin protocol. Let $F(n)$ be the running time of P on a network of size n. Then there exists a BA protocol BA_P with expected running time $O(F(n))$. If P is $r-$resilient, then the resilience of BA_P is $min(1/3, r)$.

Proof: Consider the following protocol. Each player i has a private input bit B_i and outputs a bit b_i.

Protocol BA_P

Common Parameters:

n, the size of the network (in unary)

t, an upper bound on the number of bad players($t < n/3$)

Private Input for Every Player i: B_i, a bit

Code for Every Player i	** Comments
Step .5: For every $1 \leq j \leq n$, initialize $c_j = 0$.	
Step 1: Distribute B_i.	

```
Step 2: For each $j$, if $j$ sent a bit in Step 1, reset
    $c_j$ to that bit. Let $count_i$ denote the number of $j$        ** This corresponds to $Q_P$
    such that $c_j=1$. Run $P$; let $d$ be the output.
        (a) If $count_i \leq t$ then reset $B_i=0$.
        (b) If $count_i > 2t$ then reset $B_i=1$.
        (c) If neither (a) nor (b), reset $B_i=d$.

Step 3: Distribute $B_i$.

Step 3.5: For each $j$, if $j$ sent a bit in Step 3, reset
    $c_j$ to that bit. Let $count_i$ denote the number of $j$        ** Fix the common coin to be
    such that $c_j=1$.                                              ** 1; if there is a strong
        (a) If $count_i \leq t$ then reset $B_i=0$.                 **  majority for 1, terminate.
        (b) If $count_i > 2t$ then reset $B_i=1$ and go to Step 5.
        (c) If neither (a) nor (b), reset $B_i=1$.

Step 4: Distribute $B_i$.

Step 4.5: For each $j$, if $j$ sent a bit in Step 4, reset
    $c_j$ to that bit. Let $count_i$ denote the number of $j$        ** Fix the common coin to be
    such that $c_j=1$.                                              ** 0; if there is a strong
        (a) If $count_i \leq t$ then reset $B_i=0$ and go to Step 5.  **  majority for 0, terminate.
        (b) If $count_i > 2t$ then reset $B_i=1$.
        (c) If neither (a) nor (b), reset $B_i=0$.

    Return to Step 1.

Step 5: Distribute $B_i$. Let $b_i=B_i$. Output $b_i$. Terminate.
```

Observe that all good processors that have not branched to Step 5 are all in the same Step (since the only branch they (may) have performed is from Step 4.5 back to Step 1, which all do in unison).

Claim: Suppose, at the beginning of some iteration of Step 1,3, or 4, for some $b \in \{0,1\}$, $B_i=b$ for each good processor i. Then every good j keeps $B_j=b$ for the rest of the protocol.

Proof: Assume that at the start of an iteration of Step 1, 3, or 4, $B_i=1$ for every good player i. Observe that if processor g has correctly terminated, then the last bit that g distributed was $B_g=1$, so if j is good, j has (internally) set $c_g=1$ and never changes c_g (since g sends no more messages). Every good player i that has not terminated now distributes $B_i=1$, so any good player j (that has not yet terminated) sets $c_i=1$ at the next step, and hence $count_j \geq n-t$. Therefore, j follows the instructions of Step (2b), (3.5b), or (4.5b), all of which leave $B_j=1$. Thus, B_j never changes. Moreover, this shows that all good processors branch to Step 5 after the next time they reach Step 3.5, and output 1.

Likewise, if $B_i=0$ for all good players i, then for each good j, the next time count_j is computed it is at most t, and hence j follows the instructions of Step (2a), (3.5a), or (4.5a), all of which leave $B_j=0$. Similarly to before, all good processors branch to Step 5 after the next time they reach Step 4.5, and output 0. Property (2) follows by observing that if all good processors start with input b, then $B_i=b$ for all good i in the first iteration of Step 1.

To show property (1), consider the earliest time at which a good processor i branches to Step 5. If i branches from Step 3.5b, then $\mathit{count}_i>2t$, so for every good j, $\mathit{count}_j>t$, so j resets $B_j=1$ in Step (3.5b) or (3.5c). Then by our claim, all good processors will output 1. This constitutes a proof of agreement on 1. Likewise, if i branches from Step 4.5a, then $\mathit{count}_i\leq t$, so for every good j, $\mathit{count}_j\leq 2t$, so j resets $B_j=0$ in Step (4.5a) or (4.5c), so by the claim, all good processors will output 0.

All that remains to show is that BA_P terminates in expected time $O(F(n))$. In each iteration of Steps 1-4.5 the good processors end Step 2 in agreement with probability at least p, so the expected number of iterations is at most $1/p$. All good processors terminate, at the latest, on an iteration in which all good processors end Step 2 in agreement. QED

As suggested in the proof, the good processors need not terminate simultaneously. In fact, [Fischer and Lynch 1982] prove that it is impossible to have a fast BA which guarantees that all processors finish in the same round! However, we have seen that all good processors terminate at most one iteration after the first good processor terminates.

We now show that the unpredictability of the coin is crucial. Let P and BA_P be as above. Call an adversary A ***privileged*** if A has an input tape on which the the sequence of common coins is written (for simplicity, we assume that all good processors will always output P with a common bit).

Theorem 9: There exists a privileged $2/n$-adversary A acting on BA_P such that for some set of input bits $B_1,...,B_n$, the protocol never terminates.

Proof: Without loss of generality, assume that the first common coin is 0, and that $B_i=1$ for exactly $2t$ values of i, $1\leq i<n$. A corrupts player n. A initializes $\mathit{Iteration}=1$. We need only specify how A determines the communications of the corrupted player, in Steps 1,3, and 4.

Step 1: Player n sends the bit 1 to players $1,2,...,t$ and the bit 0 to players $t+1,...,n-1$.

Step 3: Player n sends the bit 1 to players $1,2,...,2t$ and the bit 0 to players $2t+1,...,n-1$.

Step 4: A inputs the $\mathit{Iteration}+1$-st coin, b, then resets $\mathit{Iteration}=\mathit{Iteration}+1$. If $b=0$, player n sends the bit 1 to players $1,2,...,2t$ and the bit 0 to players $2t+1,...,n-1$. If $b=1$, player n sends the bit 1 to players $1,2,...,t$ and the bit 0 to players $t+1,...,n-1$.

We show that the network never reaches agreement. In the first iteration of Step 2, players $1,2,...,t$ compute *counts* of $2t+1$, and reset to 1; players $t+1,2,...,n-1$ compute *counts* of $2t$, and reset to the first common coin, 0.

In Step 3.5, players $1,2,...,2t$ compute *counts* of $t+1$, and reset to 1; players $2t+1,2,...,n-1$ compute *counts* of t, and reset to 0.

We break the analysis of Step 4.5 into two cases.

Case 1: If the next common coin is 0, then after step 4, players $1,2,...,2t$ compute *counts* of $2t+1$, and reset to 1; players $2t+1,2,...,n-1$ compute *counts* of $2t$, and reset to 0.

Case 2: If the next common coin is 1, then after step 4, players $1,2,...,t$ compute *counts* of $2t+1$, and reset to 1; players $t+1,2,...,n-1$ compute *counts* of $2t$, and reset to 0.

In Case 1, the processors will start the next iteration just as they started the first iteration, and hence they will not reach agreement on the next iteration, either. In Case 2, in the next iteration of Step 2, players $1,2,...,t$ compute *counts* of $t+1$, and reset to the next common coin, 1; players $t+1,2,...,n-1$ compute *counts* of t, and reset to 0. At that point, the network is in the same state as it was in Case 1. Thus, by following the above program, A ensures that the network never reaches agreement.[2] QED

4.6 The BA Protocol

We now recall the theorems we have proven, and put them together to exhibit a BA protocol running in constant expected time with constant resiliency.

Theorem 3: *Grade-Cast* is a 1/3-resilient *Graded-broadcast* protocol.

Recall that *Grade-Cast* has 4 Steps, three of which takes 1 round each; the last is "free".

Theorem 5: SimpleGraded-VSS is a 1/4-resilient Graded-VSS.

Overall, SimpleGraded-VSS has 2 Grade-Cast instructions, 3 steps which require 1 round each, and 3 free steps, hence its running time is 9 rounds.

Theorem 6: Let (*Graded-Share/Decide*, *Graded-Recover*) be a Graded-VSS running in time $F(n)$ on a network of size n. Then there exists a .27-fair common coin protocol running in time $F(n)+O(1)$. If the Graded-VSS is r-resilient, then the common coin is $min(1/3,r)$-resilient.

Protocol *Vote*, as constructed by the proof of Theorem 6 by using *SimpleGraded-VSS* has running time 12 rounds and is 1/4-resilient.

2 If the first common coin is 1, agreement may be indefinitely delayed if exactly t good players have input bit 1. If agreement is to be reached on a value n was supposed to distribute, then A can specify the initial input bits.

Theorem 8: Let P be a common coin protocol. Let $F(n)$ be the running time of P on a network of size n. Then there exists a BA protocol BA_P with expected running time $O(F(n))$. If P is $r-$ resilient, then the resilience of BA_P is $min(1/3,r)$.

Protocol BA_{Vote}, as constructed by the proof of Theorem 6, is 1/4-resilient . The expected number of iterations is at most 1/.27. Each iteration takes 15 rounds, so the expected running time is at most $1+(15/.27)<57$ rounds.

We remark that it is easy to convert BA_{Vote} into a terminating protocol. Namely, each processor keeps track of how many iterations of the common coin have been run. If, after the n-th iteration, a processor sees a proof of agreement on a bit b (branches to Step 5), it outputs b but does *not* terminate yet. Three rounds after the end of the n-th common coin, each processor (that had not terminated before the n-th common coin) runs the deterministic BAP of [Dolev, Fischer, Fowler, Lynch and Strong 1982], using its current bit as private input,and terminates in $2t+3$ more rounds; any player that had not yet output a bit does so now.

Conceivably, the deterministic BAP may fail, due to non-participation by good processors that already terminated. However, if any good processor terminated before the n-th common coin, then it saw a proof of agreement for a bit b, so all good processors see a proof of agreement on b, at the latest, following the n-th common coin. Thus, all output b without waiting for the deterministic BAP. If all good processors run the deterministic BAP, then it is guaranteed to work. (If some processors saw a proof of agreement on b following the n-th common coin, they need not wait for the result of the deterministic BAP, since all good processors will have private input b, and hence all will output b.)

The probability that the protocol has not terminated after n common coins is at most $.7^n$, so appending the deterministic agreement adds at most $(n)(.7^n)<1$ to the expected running time.

5. A 1/3-Resilient, Error-Free VSS

Of all protocols presented so far, only SimpleGraded-VSS has resiliency less than 1/3. Thus, if we can demonstrate a 1/3-resilient Graded-VSS, this immediately gives a 1/3-resilient BAP. In this section, we present a 1/3-resilient VSS; the conversion to a Graded-VSS is straightforward, as before.

In this VSS, *every* passable secret is recoverable. In it, each player receives both a piece and a *dual* piece of the secret. We offer a conceptual overview of the purpose of *dual* pieces. Each piece of the secret is a function defined on a vertical line; each *dual* piece is a function defined on a horizontal line. Each vertical line intersects each horizontal line at a point. We say that a piece *fits* a dual piece if they determine the same value at this intersection. A good player accepts his piece if and only if it fits the dual pieces of all other players. Any incorrect piece may fit at most t correct dual pieces. In the recovery protocol, each player reveals (distributes) his piece and dual piece. A revealed piece is correct if and only if it fits at least $2t+1$ revealed dual pieces.

Our protocol, *BestVSS*, is based on the protocol of [Ben-Or, Goldwasser and Wigderson 1988]. We present the underlying VSS which assumes broadcast channels; We first describe the ***ordinary*** secret sharing underlying the VSS. To share a secret in $[0,m-1]$, the dealer selects a ***bivariate*** polynomial $S(x,y)$ of degree t in x and in y, such that $0 \leq S(0,0) < m$. Let $\sigma = S(0,0)$. Let $S^0(y)$ denote the (univariate) $t-$th degree polynomial $S(0,y)$. $S^0(y)$ may be used to share the secret σ by the secret sharing outlined in Section 3.1. Accordingly, i's piece of the secret is $S^0(i)$. In fact, we shall share the secret by giving even more information to player i. In particular, i's piece is the $t-$th degree polynomial in x obtained by restricting $S(x,y)$ to $y=i$, which we call $S_i(x)=S(x,i)$. Notice that $S^0(i)$ is the constant term of $S_i(x)$, so i gets no less information in this scheme. It follows that a properly shared secret can be recovered from any $t+1$ good pieces.

i's ***dual*** piece of the secret is the $t-$th degree polynomial in y obtained by restricting $S(x,y)$ to $x=i$, which we call $S^i(y)=S(i,y)$. i's piece $S_i(x)$ intersects j's dual piece $S^j(y)$ at the point (j,i); if the dealer, i, and j are good, then $S_i(j)=S^j(i)=S(j,i)$. We shall have to prove that the pieces and dual pieces of the bad players alone specify neither $S(x,y)$ nor $S^0(y)$ nor σ.

Since every passable secret is recoverable, the confidence parameter is not needed. *BestVSS* does use, as a common parameter, a prime p exceeding n and not less than m (which we may specify to be the least such prime; for $m \leq n$, this may be computed in time polynomial in n).

Protocol *BestShare*

Common Parameters:

n, the size of the network (in unary)

t, an upper bound on the number of bad players($t<n/3$)

h, the identity of a distinguished processor, the dealer

m, the number of possible messages

p, a prime $(p>n, p \geq m)$; all calculations are in Z_p

Private Input for Every Player i: None

Additional Code for Player h

Step 1: Uniformly pick a $t-$th degree polynomial, $S(x,y)$ such that $0 \leq S(0,0) < m$. For each i, let $S_i(x) = S(x,i)$ and $S^i(y) = S(i,y)$. Send $S_i(x), S^i(y)$ on the private channel to player i.

Code for Every Player i

Step 1.5: Receive $U_i(x)$ and $U^i(y)$ on the private channel from h. Begin *BestDecide*.

Protocol *BestDecide*

Step 1: For each j, send $U_i(j)$ and $U^i(j)$ on the private channel to player j.

Step 2: Let u_j, v_j denote the values received from player j. For each j such that $U_i(j) \neq v_j$ or $U^i(j) \neq u_j$, broadcast $(i, j, U_i(j), U^i(j))$.

Step 3: For each proper broadcast $(i, j, U_i(j), U^i(j))$ such that $U_i(j) \neq S(j,i)$ or $U^i(j) \neq S(i,j)$, broadcast $(i, S_i(x), S^i(y))$.

Step 4: If h gave a proper broadcast $(i, S_i(x), S^i(y))$, then reset $U_i(x) = S_i(x)$, $U^i(y) = S^i(y)$, and go to Step 4.5. Otherwise, for each proper broadcast (of Step 2) $(g, j, U_g(j), U^g(j))$: if j gave a proper broadcast $(j, g, U_j(g), U^j(g))$ and $U^g(j) \neq U_j(g)$ or $U_g(j) \neq U^j(g)$, check that h properly broadcast $(g, S_g(x), S^g(y))$ or $(j, S_j(x), S^j(y))$. Check that for each proper broadcast (of Step 3) $(j, S_j(x), S^j(y))$, $S_j(i) = U^i(j)$ and $S^j(i) = U_i(j)$. If all of these check, broadcast "Goodpiece".

Step 4.5: If at least $n-t$ players broadcast "Goodpiece", output "Accept"; otherwise, output "Reject". Save $U_i(x), U^i(y)$ as private retained input to *BestRecover*.

Protocol *BestRecover*

Common Parameters:

n, the size of the network (in unary)

t, an upper bound on the number of bad players($t < n/3$)

m, the number of possible messages

p, a prime ($p > n, p \geq m$); all calculations are in Z_p

(Stored) Private Input for Every Player i: $U_i(x), U^i(y)$, polynomials received in *BestShare*

Code for Every Player i

Step 1: Distribute $(i, U_i(x), U^i(y))$.

Step 1.5: Initialize $V=0$. For each j, receive $(j, V_j(x), V^j(y))$ from j (set $V_j(x) = V^j(y) = 0$ if a proper message was not received). For each j, set $confirm_j = 1$ if $V_j(g) = V^g(j)$ for at least $2t+1$ values of g. If there exists a set $j_1, ..., j_{t+1}$ of indices such that $confirm_{j_a} = 1$ for each a, then let the constant term of V_{j_a} be u_a for $1 \leq a \leq t+1$, and let V be the $t-$th degree polynomial interpolating $(j_1, u_1), ..., (j_{t+1}, u_{t+1})$. Output $V(0)$.

Theorem 10: *BestVSS* is a 1/3-resilient VSS.

To motivate the proof, we begin by assuming, for now, that all players *except* the dealer are *angels*, i.e., uncorruptable. The key idea is that the validity of a piece is determined by how well it *fits* the dual pieces. For every i and j, i's piece "intersects" j's dual piece at the point (j,i); if i's piece and j's dual are correct, $S(j,i) = U_i(j) = U^j(i)$. Player i computes this value, which we call a *subpiece*, according to his received piece, and sends this to j; j computes its value according to his dual piece and sends this to i. We say that i's piece *fits* j's dual piece if they send the same subpiece to each other.

We first consider the pieces and dual pieces received by any $t+1$ angels, $1, 2, ..., t+1$. (We omit subscripts for simplicity; the angels' identities are irrelevant.) We observe that *any* set of $t+1$ pieces $\{U_1(x), ... U_{t+1}(x)\}$ uniquely defines a $t-$th degree polynomial $U(x,y)$ as follows. For any fixed z, let $U^z(y)$ be the unique $t-$th degree polynomial[1] interpolating $\{(1, U_1(z)), (2, U_2(z)), ... (t+1, U_{t+1}(z))\}$; set $U(x,y) = U^x(y)$.

In *BestDecide*, each pair of angels (i,j) exchange the subpiece corresponding to the intersection of i's piece and j's dual piece, $U_i(j)$ and $U^j(i)$. If these are not the same, then they broadcast their *conflict* (i.e., their differing subpieces) in Step 2. A conflict among angels

1 We have used $U_1(z), ... U_{t+1}(z)$ to define a *function* $U(z,y)$. For fixed z, $U(z,y)$ is a $t-$th degree polynomial in y. Likewise, for fixed y, $U(z,y)$ is a $t-$th degree polynomial in z. It follows that $U(z,y)$ is a polynomial whose degree in z and y each is at most t.

exposes the dealer (proves that he is bad). Actually, for *any* $t+1$ **pieces** the dealer sent, $U(x,y)$ is well-defined; when the angels check that all pieces fit all dual pieces, they are merely checking that the dealer sent the correct dual pieces for these pieces.

Let us add an angel g (where $g>t+1$); let $U_g(x)$ and $U^g(y)$ denote the piece and dual piece he receives. $U(x,y)$ has already been defined by the pieces of the first $t+1$ angels, so if the dealer is good, $U_g(x)=U(x,g)$ and $U^g(y)=U(g,y)$. This implies that $U_g(x)$ fits $U^j(y)$, and $U_j(x)$ fits $U^g(y)$ for every $1\leq j\leq t+1$. If, for some j, either does not fit, j and g expose the dealer. Thus, either the dealer sends to g polynomials $U_g(x)$ and $U^g(y)$ passing through $(j,U^j(g))$ and $(j,U_j(g))$ respectively, for every $1\leq j\leq t+1$, or he is exposed. Since there is a unique $t-$th degree polynomial interpolating $t+1$ points, if the dealer is not exposed, we may conclude that he sent the proper piece and dual piece to g. Obviously, the same applies for any number of additional angels. The pieces of any $t+1$ angels determine $U(x,y)$ and all other pieces and dual pieces; if any other piece or dual piece is not given by $U(x,y)$, it cannot fit the pieces or dual pieces of those $t+1$ angels, and the dealer is exposed.

We now return to reality, in which not only the dealer, but any $t<n/3$ players may be corrupted. A conflict between players g and j only implies that either the dealer OR g OR j is bad. The dealer must *resolve* this conflict by *revealing* (broadcasting), in Step 3, the piece and dual piece of g or j (or both). (Note: If g broadcasts a conflict with j, but j does not broadcast a conflict with g, then it is not clear whether g or j is bad. The same applies if they broadcast identical values. In such a case, the dealer need not respond. Bad players are at liberty to fabricate and exchange values which do or do not fit; our only concern is that all conflicts between good players are resolved.)

Player i accepts his piece if the following conditions are met:

(1) the dealer resolves all conflicts;

(2) every revealed piece and dual fits i's dual piece and piece;

(3) i is not revealed.

The good players accept the secret if and only if at least $n-t$ players accept their pieces.

Recognition of dirty pieces follows from superiority of numbers. Namely, suppose that all good players receive good pieces and dual pieces (i.e., consistent with a $t-$th degree polynomial). In *BestRecover*, each player distributes his piece and dual piece. Since a piece is a $t-$th degree polynomial, if it is not the correct one (which passes through all points of intersection with the dual pieces of the good players, which number at least $2t+1$), it may pass through at most t of them. An incorrect piece may additionally fit dual pieces revealed by (at most t) bad players, hence it may fit at most $2t$ dual pieces overall. Thus, a piece is correct if and only if it fits at least $2t+1$ revealed dual pieces.

This argument also proves that if the dealer reveals an incorrect piece or dual piece, the secret is rejected. If a secret is acknowledged by any good processor, there are at least $t+1$ satisfied good players. An incorrect piece or dual piece could not fit the pieces or dual pieces of more than t satisfied good players.

If a good player i is revealed, i knows that the dealer is bad. Nevertheless, i resets his piece and dual piece to whatever the dealer broadcast in Step 3, since if the secret is passable (accepted by a good player), all revealed pieces are correct.

The unpredictability of the secret is not compromised by the system of conflicts. This is because when the dealer is *nc*, every conflict involves a bad player, and any subpiece or piece broadcast in the conflict or its resolution had been given to the bad player, and the bad player could have broadcast his own piece in any event.

Proof of Theorem 10: Property (1), unanimity, holds because acceptance depends only upon the number of broadcasts "Goodpiece", and all players receive the same number.

Property (2), acceptance of good secrets, holds because an uncorrupted dealer corrects any broadcast of an incorrect subpiece, and hence every conflict is properly resolved, so every good player broadcasts "Goodpiece".

For (3) (recoverability), observe that if the secret is passable, then at least $t+1$ good players are *satisfied* (broadcast "Goodpiece"). It follows that all their pieces and dual pieces fit each other, hence they uniquely define a $t-$th degree polynomial $U(x,y)$. Moreover, every conflict between good players was resolved, and the pieces of the satisfied players fit every piece and dual piece the dealer revealed, hence a good piece and dual piece have been given to every good player, either privately in *BestShare*, or publicly in *BestDecide*. Therefore, in *BestRecover*, for each good player i, at least $2t+1$ of the dual pieces i receives are correct. Therefore, i sets $confirm_j=1$ if and only if what i receives as V_j truly equals $U(x,j)$. Therefore, the interpolation is guaranteed to return $U(0,0)$, regardless of what the bad players do. Since $U(x,y)$ is determined during *BestDecide*, $E_{U(0,0)}$ is fixed at the end of *BestDecide*.

The proof of (4), unpredictability, follows the approach used for SimpleVSS, but is more complicated. It may help to view the selection and sharing of the secret in the following manner. The polynomial $S(x,y)$ may be randomly selected by picking $(t+1)$ random $t-$th degree polynomials $S_{y_0}(x),...,S_{y_t}(x)$ (for distinct $y_0,...,y_t$). This says that $S(x,y)$ is determined by its values on any $t+1$ horizontal lines. Likewise, its values on any horizontal line $y=y_i$ are determined by its values at any $t+1$ points on that line, $S(x_0,y_i),...,S(x_t,y_i)$. Thus, $S(x,y)$ may be uniformly selected by uniformly picking its values at $(t+1)^2$ distinct points such that $t+1$ of these points lie on each of $t+1$ horizontal lines. Let the bad players be $i_1,...,i_t$; let $i_0=0$. Let $R=\{(i_h,i_j):0\leq h,j\leq t\}$; let $\overline{R}=R-\{(0,0)\}$ (all points of R except $(0,0)$). There exists a 1-1 correspondence between t-th degree polynomials $S(x,y)$ and ordered $(t+1)^2$-tuples of values in Z_p. We have just argued that for any secret $\sigma=S(0,0)$, the values of S at points of $\overline{R}$ are chosen uniformly. We prove unpredictability by observing that all of A's inputs are determined by the values of S on $\overline{R}$. Let $i\in\{i_1,\ldots,i_t\}$. Since $\overline{R}$ contains $t+1$ points with y-coordinate (x-coordinate) i, i's piece (dual piece) is determined by the values of S on $\overline{R}$. All subpieces sent to i, $U_{gi}=S(g,i)$ and $U_{ig}=S(i,g)$, are determined by i's piece and dual piece. Player i should send these same subpieces, so if he sends anything else, the recipient merely broadcasts the correct values, which A already knew. Likewise, if i broadcasts incorrect values, the dealer merely reveals what he already gave to i. Since good players do not conflict with other good players, it follows that for any set of t bad players, all of A's inputs are determined by the values of S on $\overline{R}$, which are uniformly distributed, independent of the secret chosen. (If only d players are corrupted, then A's view is determined by the values of S at $(2d)(t+1)-d^2$ points in $\overline{R}$, and hence for any secret, there are $m^{(t+1-d)^2-1}$ polynomials consistent with his view.) QED

6. Conclusion

We have demonstrated a BAP with the maximum fault tolerance possible and running in constant expected time. This settles a problem which has attracted quite a bit of attention. Our solution assumes private communication channels. In [Feldman 88], we show that for a computationally bounded adversary, subject to an intractability assumption, cryptography may be used to simulate private communication lines. We leave as an open problem the best BAP possible assuming neither private communication lines nor cryptography.

Appendix

1. Optimizing the Expected Time per Agreement

1.1 A Faster VSS

We address the problem of optimizing the running time of Graded-VSS; we consider the 1/3-resilient protocol *BestVSS* of Section 5. Just as *SimpleVSS*, *BestVSS* may be converted to a Graded-VSS by proper substitution of *Grade- Casts* for broadcasts. The proof that the modified protocol has the desired properties follows just as for *SimpleGraded- VSS*. However, the fastest version does not use the full power of *Grade- Casts*.

Protocol *FastShare*

Common Parameters:

n, the size of the network (in unary)

t, an upper bound on the number of bad players($t<n/3$)

h, the identity of a distinguished processor, the dealer

m, the number of possible messages

p, a prime ($p>n, p\geq m$); all calculations are in Z_p

Private Input for Every Player i: None

Additional Code for Player h

Step 1: Pick a random polynomial $S(x,y)$ such that $0\leq S(0,0)<m$; for each i, send $S_i(x)$ and $S^i(y)$ on the private channel to player i.

Code for Every Player i

Step 1.5: Receive $U_i(x)$ and $U^i(y)$ on the private channel from the dealer.

Protocol *FastDecide*

Step 1: For each j, send $U_i(j)$ and $U^i(j)$ on the private channel to player j.

Step 2: Let u_j, v_j denote the values received from player j. For each j such that $U_i(j)\neq v_j$ or $U^i(j)\neq u_j$, distribute $(i,j,U_i(j),U^i(j))$.

Step 3: Let $\{(g,j,W_{igj},X_{igj})\}$ denote all proper messages received in Step 2. For each (g,j) such that $W_{igj}\neq X_{ijg}$, distribute (g,j,W_{igj},X_{ijg}).

Step 4: Let $\{(g,j,u_{igj},v_{ijg})\}$ denote all proper messages received in Step 3. For each (g,j,u,v) received from at least $t+1$ players: if $u\neq S(g,j)$, then distribute $(g,S_g(x),S^g(y))$; if $v\neq S(g,j)$, then distribute $(j,S_j(x),S^j(y))$.

Step 5: Let $\{(g,j,u_{fgj},v_{fgj})\}$ denote all proper messages received in Step 3. Let $Z_i=\{(g,f_g(x),f_g(y))\}$ denote all proper messages received from h in Step 4. Check that for each $\{(g,j,u,v)\}$ received from at least $n-t$ players, either $(g,f_g(x),f_g(y))$ or $(j,f_j(x),f_j(y))$ was received. Check that for each $(j,f_j(x),f^j(y))$, $f_j(i)=U^i(j)$ and $f^j(i)=U_i(j)$. Check that Z_i does not include a triple $(i,f_i(x),f^i(y))$. If all of these hold, distribute Z_i; otherwise, set $Z_i=\emptyset$.

Step 6: Let z_j denote the message received from j at Step 5. If $z_j=Z_i$ for at least $n-t$ players j, then distribute Z_i.

Step 6.5: For each z, let *count*$_z$ be the number of players that sent z at Step 6.

(a) If, for some z, *count*$_z\geq 2t+1$, then set $Z_i=z$ and *accept*$_i=2$.

(b) Else, if for some z, *count*$_z\geq t+1$, set $Z_i=z$ and *accept*$_i=1$.

(c) Otherwise, set *accept*$_i=0$.

Output *accept*$_i$. Save $Z_i,U_i(x),U^i(y)$ as private retained input to *FastRecover*.

Protocol *FastRecover*

Common Parameters:

n, the size of the network (in unary)

t, an upper bound on the number of bad players($t<n/3$)

m, the number of possible messages

p, a prime ($p>n, p \geq m$); all calculations are in Z_p

(Stored) Private Input for Every Player i: $Z_i, U_i(x), U^i(y)$, values received in *FastShare/Decide*

Code for Every Player i

Step 1: Distribute $(i, U_i(x), U^i(y))$.

Step 1.5: Initialize $V=0$. For each j, receive $(j, f_j(x), f^j(y))$ from j (set $f_j(x)=f^j(y)=0$ if a proper message was not received). If Z_i includes a triple $(j, f'_j(x), f'^j(y))$, set $f_j(x)=f'_j(x)$, $f^j(y)=f'^j(y)$ For each j, set $confirm_j=1$ iff $f_j(g)=f^g(j)$ for at least $2t+1$ values of g. If there exists a set $j_1,...,j_{t+1}$ of indices such that $confirm_{j_a}=1$ for each a, then let the constant term of f_{j_a} be u_a for $1 \leq a \leq t+1$, and let V be the $t-$th degree polynomial interpolating $(j_1,u_1),...,(j_{t+1},u_{t+1})$. Output $V(0)$.

Theorem 11: *FastVSS* is a 1/3-resilient Graded-VSS.

Proof: We begin with a counting argument to show that Z_i is well defined in Step 6.5 of *FastDecide*. Let i and j be good players that distribute Z_i and Z_j, respectively, in Step 6. It must be that at least $n-t$ players send Z_i to i at Step 5 and at least $n-t$ players send Z_j to j at Step 5. Thus, at least $n-2t$ players sent Z_i to i and Z_j to j. At least one such player is good, so $Z_i=Z_j$. Thus, all good players sending a message in Step 6 send a common message z, so any i acknowledging the secret saves $Z_i=z$.

Property (1′), semi-unanimity, is immediate, since if any good player accepts, he receives some z from at least $2t+1$ players, so all players receive z from at least $t+1$ players, so they all acknowledge.

Property (2), acceptance of good secrets, follows from the fact that an uncorrupted dealer distributes a message Z which resolves all conflicts, at Step 4, and every good player redistributes Z at Steps 5 and 6.

To prove (3), recoverability, observe that if any good player i acknowledges the secret, at least one good player g sends a string z at Step 6, so at least $n-2t$ good players distributed z at Step 5. Therefore, z resolves all conflicts between good players. What was argued regarding satisfied players in *BestVSS* applies to the good players that distributed z. Namely, the pieces and dual pieces of all such players are consistent with a $t-$th degree polynomial $U(x,y)$. Moreover, since the pieces of these players fit all pieces revealed in z, all such pieces and dual pieces are also consistent with $U(x,y)$. Hence, for every good player j, the unique piece and dual piece for j determined by $U(x,y)$ are either revealed in z or are held by j himself. Therefore, in *FastRecover*, i can distinguish good pieces from dirty ones, and recovers $U(0,0)$.

Property (4), unpredictability, holds as for *BestVSS*. For any possible secret, there is exactly one polynomial consistent with the adversary's view. (If only d players are corrupted, then there are $m^{(t+1-d)^2-1}$ polynomials consistent with his view.) QED

1.2 A Faster, Fairer Common Coin Protocol

We speed up the BA algorithm in three additional ways. Firstly, we improve the fairness of the common coin. Secondly, we economize on rounds by eliminating the *Grade-Cast* from the common coin protocol. Finally, we show that the common coin is unpredictable until secrets are recovered; this allows us to begin the common coin protocol, and "store" coins, in advance. To accomplish this last objective, we define a ***compound common coin protocol.***

Definition 11: Let $P=(Q,R)$ be a jointly terminating compound protocol. Each good player i outputs a bit d_i at the end of R. Let A be an adversary that acts on P and outputs a bit b before the start of R. Let E_v be the event that $d_i=v$ for every i that correctly terminates R. We say that P is a ***compound common coin protocol*** if for a constant $p>0$, with probability at least p, E_{1-b} is fixed at the end of Q. (We say P is $p-fair$.)

We present *FastCoin*$=$(*Ballot*,*Tally*) as a compound common coin protocol. *FastCoin* takes a common parameter m (to represent the range of votes and tallies), which we set to be $n/\ln(64/27)$ (where ln denotes the logarithm to the base e).

Protocol *Ballot*

Common Parameters:

n, the size of the network (in unary)

t, an upper bound on the number of bad players($t<n/3$)

m, the number of possible messages

Private Input for Every Player i: None

Code for Every Player i

Step 1: For each $1\le h\le n$, for each $1\le j\le n$, run *FastShare/Decide*, specifying as common parameters: $h=$ dealer; $m=$ the number of possible secrets; and j as an execution label (denoting that this vote is for j; we shall refer to this execution as secret h,j). Let $accept_{hj}$ denote the output of *FastDecide*.

Step 2: For each j, let A_{ji} denote $accept_{ji}$. Distribute the *list* $(A_{1i},\ldots,A_{ni})$.

Step 2.5: For each j, let $List_{ij}=a_{1j},\ldots,a_{nj}$ be the message received from j (0,...,0 if a proper message was not received). If $(\max_h (a_{hj}-accept_{hj})<2)$ and $(a_{kj}=2$ for at least $n-t$ values of h), then set $goodlist_j$=1, else set $goodlist_j$=0. Save $goodlist_1,...,goodlist_n,List_{i1},...,List_{in}$ as stored private input to *Tally*.

Protocol *Tally*

Common Parameters:

n, the size of the network (in unary)

t, an upper bound on the number of bad players($t<n/3$)

m, the number of possible messages

(Stored) Private Input for Every Player i: $goodlist_1,...,goodlist_n,List_{i1},...,List_{in}$, values from *Ballot*

Code for Every Player i

Step 1: For each j such that $goodlist_j=1$, distribute $(j,List_{ij})$. For each h and j, run *FastRecover* on secret h,j. Let v_{hj} be the output.

Step 1.5: For each j, if $(j,List_{ij})$ is received from at least $n-t$ players, set $candidate_j=1$; otherwise, set $candidate_j=0$. For each j such that $candidate_j=0$, set $tally_j=\emptyset$. For each j such that $candidate_j=1$, set $tally_j=\left(\sum_{h:\ A_{hj}=2} v_{hj}\right)$ mod m. If $tally_j=0$ for some j, set $d_i=0$; else, set $d_i=1$. Output d_i.

The usual counting argument shows that if g and i are good players that keep j, then $List_{gj}=List_{ij}$. Since, for *FastVSS*, every passable secret is recoverable, all tallies are well-defined. (Note: we only consider tallies of candidates in the running, since only these tallies are relevant.) If a candidate that correctly terminates *Ballot* has tally 0, then E_0 is fixed at the end of *Ballot* (i.e., the common coin is unanimously 0). If no candidate in the running has tally 0, then E_1 is fixed at the end of *Ballot*. We must show that E_v is fixed at the end of *Ballot* with probability at least p, for $v\in\{0,1\}$.

As in *Vote*, each player has probability $1/m$ of having tally 0, independently of all other tallies. The following approximations are true in the limit as $n\to\infty$. (Since the deterministic BAPs are faster for $n<30$, these approximations are fairly accurate for the range of values one might feasibly encounter.) E_1 is fixed if no player in the running has tally 0; this has probability at least $(1-1/m)^n$, which tends towards $e^{-n/m}$. E_0 is fixed if a good player has tally 0; the probability that no good player has tally 0 is at most $(1-1/m)^{n-t}<(1-1/m)^{(2n/3)}$, which approaches $e^{(-2/3)(n/m)}$, hence the probability that a good player has tally 0 is at least $1-e^{(-2/3)(n/m)}$. To optimize the fairness of the coin, we set these probabilities equal, which happens (approximately) at $m=n/\ln(64/27)$; the respective probabilities tend towards $e^{-n/m}=27/64>.42$ and $1-e^{(-2/3)(n/m)}=1-9/16=7/16>.43$. Thus, we have outlined the proof of the following theorem:

Theorem 11: *FastCoin*=(*Ballot*, *Tally*) is a 1/3-resilient, .42-fair compound common coin protocol.

1.3 The Expected Running Time

Since *FastShare/Decide* takes 7 rounds, *Ballot* takes 8 rounds. The fast BA protocol is as follows. As Step 0 (the first round), the network starts an execution of *FastCoin*; for each $k \geq 0$, the network starts execution $k+1$ of *FastCoin* in round $4k$. Protocol $BA_{FastCoin}$ is begun in round 8; in general, the k-th iteration of Step 1 of $BA_{FastCoin}$ occurs in round $4k+4$, coinciding with the end of the k-th execution of *Ballot*. In the k-th iteration of Step 2, *Tally* is run to expose the k-th stored coin. The expected number of iterations needed until agreement is reached is at most 1/.42. $BA_{FastCoin}$ ends by Step 5 of the first iteration in which agreement has been reached in Step 2, so the expected running time is at most $4(1/.42)+8$, which is less than 18. Subsequent agreements may utilize stored coins, and need not wait for *Ballot*; since at most 1/.42 expected iterations are needed, the expected time is at most $(4/.42)+1<11$.

We remark that further reductions in the expected running time are possible if we assume $t<n/4$ or $t<n/5$.

1.4 Computational and Communication Complexity

Here we analyze the local communication/computational complexity and mention how to minimize them.

Nearly all of the communication and computation is required for the *Graded-VSS*'s; in *FastCoin*, each player shares n secrets. We modify *FastCoin* to *CheapCoin* by having each player i share only one secret. $\sigma_i = S_i(0,0)$ will be i's vote for all candidates, *other than i himself*, that accept i's vote.

The tally of any player in the running must include votes by good players, and is random. If i and j are good, then i's tally includes σ_j, but j's tally excludes σ_j, hence they are independent. However, the adversary may correlate tallies of bad players at will. For any adversary strategy, the chance that no good player has tally 0 is at most $(1-1/m)^{n-t}$; thus, with probability at least $1-e^{2n/3m}$, some good player has tally 0, and E_0 is fixed.

The adversary's optimal strategy to bias the coin is to attempt to cause some tally to be 0, and prevent E_1 from occuring. The adversary can ensure that the tallies of t bad players are all different, in which case the chance that none is 0 is $1-t/m$. The chance that no player in the running has tally 0 then approximates $e^{-(n-t)/m}(1-t/m)$, in which case E_1 is fixed. This probability is at least $e^{-2n/3m}(1-n/3m)$. For $m=n/\ln(64/27)$, both probabilities are at least .4, so the common coin is still .4-fair.

The communication complexity of *FastVSS* is dominated by Step 5, in which players distribute Z; Z may contain the pieces and dual pieces of t revealed players. Since a piece, consisting of $t+1$ coefficients in Z_p, takes $O(n\log n)$ bits, this amounts to $O(n^2\log n)$ bits per secret per link. Thus, *CheapCoin* takes $O(n^3\log n)$ bits per channel overall. Since the expected number of iterations of *CheapCoin* is constant, this is the expected complexity of protocol $BA_{CheapCoin}$.

The computational complexity is dominated by *FastRecover*. For each piece $f_j(x)$, i checks that it fits at least $2t+1$ dual pieces, i.e., $f_j(g)=f^g(j)$. Having found $t+1$ valid pieces, the secret is recovered by polynomial interpolation. The interpolation can be done in $O(n^2\log n)$ bit operations, but checking the pieces can require evaluating $O(n)$ polynomials at $O(n)$ points, taking $O(n^3\log n)$ bit operations per secret; the overall complexity of $BA_{CheapCoin}$ is thus $O(n^4\log n)$ bit operations.

2. Asynchronous BA

We can modify our algorithm to work in a totally ***asynchronous*** network. This is a network in which messages may take arbitrarily long to be delivered, although every message sent by a good processor eventually does get delivered. In the spirit of worst-case analysis, we assume that the adversary determines when messages arrive.

Since a synchronous network is a special case of an asynchronous network, all upper bounds for fault tolerance and lower bounds for running times for the synchronous case apply *a fortiori* to the asynchronous case. Perhaps surprisingly, Fischer, Lynch and Paterson [Fischer, Lynch and Paterson 1983] show that no deterministic asynchronous BAP can tolerate the ***death*** of even a single processor (i.e., the processor ceases all communications, which we consider to be a "benevolent" fault). Thus, any solution must be randomized.

Essentially, only two asynchronous BAP's are known. The synchronous algorithm of [Rabin 1983], which assumes initialization by a trusted dealer, also works in asynchronous systems; however, this is not a BA from scratch. The only known asynchronous BAP from scratch is based on the protocol of [Ben-Or 1983]. As for the synchronous version of his protocol, the expected ***asynchronous running time*** (appropriately defined) is constant if the number of faults is small enough; in the worst scenario (a constant fraction of the network is faulty), the expected running time is exponential in n. Ben-Or's protocol has resilience 1/5; Bracha and Toueg [Bracha and Toueg 1983] extend the protocol to optimal fault tolerance $t<n/3$, but do not offer any improvement in the running time.

In [Feldman 88], we show that the BAP of Chapter I can be adapted for asynchronous networks; the expected asynchronous running time is constant. In the synchronous case, we were able to optimize fault tolerance and running time without using cryptography. By contrast, the asynchronous protocol tolerates only $t<n/4$ faults without cryptography. Cryptography may be used to regain optimal fault tolerance $t=(n-1)/3$.

3. BA Starting With Secure Authentication Schemes

[Dolev and Dwork 1987] show that there does not a exist a BA protocol, as we have defined it, with resiliency exceeding 1/3, even given cryptography. However, if we assume that the processors start with secure *authentication* schemes, then the problem is solvable for any resiliency. Pease, Shostak and Lamport 1980] present a $t+1$ round algorithm, where t bounds the number of faults. Dolev and Strong [Dolev and Strong 1982] present a $t+1$ round algorithm with polynomially-bounded communication and computational complexity, which is optimal for deterministic algorithms. For this scenario, in which the processors start with secure authentication schemes, in [Feldman 88] we extend our protocol to a 1/2-resilient, constant expected time BAP.

References

[Benaloh 1986] J. Benaloh, Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret, Proceedings of Crypto 1986.

[Ben-Or 1983] Another Advantage of Free Choice: Completely Asynchroncus Agreement Protocols, Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, August 1983, pp. 27-30.

[Ben-Or, Goldwasser and Wigderson 1988] M. Ben-Or, S. Goldwasser, and A. Wigderson, Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation, Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, May 1988.

[Blakely 1979] Safeguarding Cryptographic Keys, Proc. AFIPS June 1979 NCC Vol. 48, pp.313-317.

[Bracha 1985] G. Bracha, An $O(log\ n)$ Expected Rounds Randomized Byzantine Generals Protocol, Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, May 1985.

[Bracha and Toueg 1983] G. Bracha and S. Toueg, Resilient Consensus Protocols, Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, August 1983, pp. 12-26.

[Chaum, Crepeau and Damgaard 1988] D. Chaum, C. Crepeau, and I. Damgaard, Multiparty Unconditionally Secure Protocols, Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, May 1988.

[Chor and Coan 1985] B. Chor and B. Coan, A Simple and Efficient Randomized Byzantine Agreement Algorithm, IEEE Transactions on Software Engineering, Vol. SE-11, No.6 1985.

[Chor, Goldwasser, Micali and Awerbuch 1985] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults, Proceedings of the Twenty-fifth Annual IEEE Symposium on Foundations of Computer Science, October 1985.

[Chor, Merritt and Shmoys 1985] B. Chor, M. Merritt, and D. Shmoys, Simple Constant-Time Consensus Protocols in Realistic Failure Models, Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing, August 1985, pp. 152-162.

[Coan 1987] B. Coan, Achieving Consensus in Fault-Tolerant Distributed Computer Systems: Protocols, Lower Bounds, and Simulations, PhD. thesis, MIT, 1987.

[Dolev 1982] D. Dolev, The Byzantine Generals Strike Again, Journal of Algorithms, Vol. 3, No. 1, pp. 14-30, March 1982.

[Dolev and Dwork 1987] D. Dolev and C. Dwork, manuscript.

[Dolev, Fischer, Fowler, Lynch, and Strong 1982] D. Dolev, M. Fischer, R. Fowler, N. Lynch, and H. Strong, An Efficient Algorithm for Byzantine Agreement Without Authentication, Information and Control,Vol. 52, Nos. 1-3, pp. 257-274, Jan-March/1982.

[Dolev and Strong 1982] D. Dolev and H. Strong, Authenticated Algorithms for Byzantine Agreement, IBM Technical Report RJ3416, March 1982.

[Dwork, Shmoys and Stockmeyer 1986] C. Dwork, D. Shmoys and L. Stockmeyer, Flipping Persuasively in Constant Expected Time, Proceedings of the Twenty-seventh Annual IEEE Symposium on Foundations of Computer Science, October 1986, pp.222-232.

[Feldman 1987] P. Feldman, A Practical Scheme for Non-Interactive Verifiable Secret Sharing, Proceedings of the Twenty-seventh Annual IEEE Symposium on Foundations of Computer Science, October 1987.

[Feldman 1988] P. Feldman, in preparation.

[Feldman and Micali 1985] P. Feldman and S. Micali, Byzantine Agreement in Constant Expected Time, Proceedings of the Twenty-fifth Annual IEEE Symposium on Foundations of Computer Science, May 1985.

[Fischer and Lynch 1982] A Lower Bound for the Time to Assure Interactive Consistency, Information Processing Letters, 14(4), pp.183-186, 1982.

[Fischer, Lynch and Paterson 1983] Impossibility of Distributed Consensus with One Faulty Process, JACM 32(2), pp.374-382, 1985.

[Goldwasser, Micali and Rackoff 1985] S. Goldwasser, S. Micali and C. Rackoff, The Knowledge Complexity of Interactive Proofs, Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, May 1985.

[Goldreich, Micali, and Wigderson 1986] O. Goldreich, S. Micali, and A. Wigderson, Proofs that Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design, Proceedings of the Twenty-seventh Annual IEEE Symposium on Foundations of Computer Science, May 1986.

[Karlin and Yao 1987] A. Karlin and A. Yao, manuscript.

[Linnik 1944] Result quoted from Encyclopedic Dictionary of Mathematics, Second Edition, MIT Press, 1987 123 D.

[Pease, Shostak and Lamport 1980] M. Pease, R. Shostak, and L. Lamport, Reaching Agreement in the Presence of Faults, JACM 27(2), 1980.

[Peterson and Weldon 1972] Peterson and Weldon, Error Correcting Codes, Second Ed., MIT Press, 1972.

[Rabin 1983] Randomized Byzantine Generals, Proceedings of the Twenty-fourth Annual IEEE Symposium on Foundations of Computer Science, May 1983, pp.403-409.

[Shamir 1979] A. Shamir, How to Share a Secret, CACM Vol.22 No.11, 1979.

[Srikanth and Toueg 1984] T. Srikanth and S. Toueg, Byzantine Agreement Made Simple: Simulating Authentication without Signatures, Cornell Technical Report 84-623, July 1984.

[Turpin and Coan 1984] R. Turpin and B. Coan, Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement, Information Processing Letters, Vol. 18, pp. 73-76, Feb. 1984.