SIAM J. COMPUT.
Vol. 26, No. 4, pp. 873–933, August 1997

AN OPTIMAL PROBABILISTIC PROTOCOL FOR SYNCHRONOUS BYZANTINE AGREEMENT*

PESECH FELDMAN† AND SILVIO MICALI‡

Abstract. Broadcasting guarantees the recipient of a message that everyone else has received the same message. This guarantee no longer exists in a setting in which all communication is person-to-person and some of the people involved are untrustworthy: though he may claim to send the same message to everyone. an untrustworthy sender may send different messages to different people. In such a setting, *Byzantine agreement* offers the "best alternative" to broadcasting. Thus far, however, reaching Byzantine agreement has required either many rounds of communication (i.e., messages had to be sent back and forth a number of times that grew with the size of the network) or the help of some external trusted party.

In this paper, for the standard communication model of synchronous networks in which each pair of processors is connected by a private communication line, we exhibit a protocol that, in probabilistic polynomial time and without relying on any external trusted party, reaches Byzantine agreement in an expected constant number of rounds and in the worst natural fault model. In fact, our protocol successfully tolerates that up to 1/3 of the processors in the network may deviate from their prescribed instructions in an arbitrary way, cooperate with each other, and perform arbitrarily long computations.

Our protocol effectively demonstrates the power of randomization and zero-knowledge computation against errors. Indeed, it proves that "privacy" (a fundamental ingredient of one of our primitives), even when is not a desired goal in itself (as for the Byzantine agreement problem), can be a crucial tool for achieving correctness.

Our protocol also introduces three new primitives—graded broadcast, graded verifiable secret sharing, and oblivious common coin—that are of independent interest, and may be effectively used in more practical protocols than ours.

Key words. broadcasting, Byzantine agreement, fault-tolerant computation, raridomization

AMS subject classifications. 68Q22, 68R05, 68M15, 94A60, 94.499, 94B99

PII. S0097539790187084

1. The problem.

A motivating scenario. We are in Byzantium, the night before a great battle. The Byzantine army, led by a commander in chief, consists of n legions, each one separately encamped with its own general. The empire is declining: up to 1/3 of the generals—including the commander in chief—may be traitors. No radios (sic!) are available: all communication is via messengers on horseback. To make things worse, the loyal generals do not know who the traitors are. During the night each general receives a messenger with the order of the commander for the next day: either "attack" or "retreat." If all the good generals attack, they will be victorious; if they all retreat, they will be safe: but if some of them attack and some retreat they will be defeated. Since a treasonous commander in chief may give different orders to different generals, it is not a good idea for the loyal ones to directly execute his orders. Asking the opinion of other generals may be quite mislezding too: traitors may represent their orders differently to different generals, they may not send any information to someone,

* Received by the editors August 30, 1990; accepted for publication (in revised form) July 31, 1995. An earlier version of this work was presented at the 1988 ACM Symposium on the Theory of Computing (STOC).

http://www.siam.or,a/journals/sicomp/26-4/18708.html

† Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139. Current address: OHR SOMAYACH, 22 Shimon Hatzedik. Jerusalem, Israel.

‡ Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 (silvio@theory.lcs.mit.edu). The research of this author was supported in part by NSF grants DCR-84-13577 and CCR-9121466. XRO grant DAALO3-86-K-0171, and ONR grant NOOO14-92-J-1799.

and they may claim to have received nothing from someone else. On the other hand, should the honest generals always—say—attack (independently of the received orders and of any discussion), they would not follow any meaningful strategy. What they need is a way to exchange messages so as to always reach a common decision while respecting the chief's order, should he happen to be honest. They need Byzantine agreement.

Byzantine agreement. As insightfully defined by Pease, Shostak, and Lamport [32], Byzantine agreement essentially consists of providing "the best alternative" to broadcasting when all communication is person-to-person (as in an ordinary telephone network) and some of the people involved are untrustworthy. In order to briefly describe what this alternative is, we must first sketch its classic underlying communication model, the most convenient and simplest one in which the need for Byzantine agreement arises.

Modernizing the motivating scenario a bit, generals are processors of a computer network. Every two processors in the network are joined by a separate communication line, but no way exists to broadcast messages. (Thus, though a processor can directly send a given message to all other processors, each recipient has no way to know whether everyone else has received the same message.) The network otherwise has some positive features. Each processor in it has a distinct identity and knows the identities of the processors on the other end of its lines. The network is synchronous, that is, messages are reliably delivered in a sufficiently timely fashion: there is a common clock, messages are sent at each clock tick (say, on the hour) and are guaranteed to be delivered by the next tick (though not necessarily simultaneously). Each communication line is private, that is, no one can alter, inject! or read messages traveling along it. Indeed, the only way for an adversary to disturb the communication of two good processors is by corrupting one of them. We will refer to such a network as a standard network since it is the one generally adopted for discussing the problem of Byzantine agreement.[1]

Now assume that each of the processors of a standard network has an initial value. Then, speaking informally, a Byzantine agreement protocol should guarantee that for *any* set of initial values, the following two properties hold:

1. Consensus: All honest (i.e., following the protocol) processors adopt a common vaiue.

e

2. Validity: If all honest processors start with the same value, then they adopt that value.[2]

Byzantine faults. Having briefly discussed our communication model, are must now mention our fault model. Processors become faulty when they deviate from their prescribed programs. "Crashing (i.e., ceasing all activities) is a benign way for a processor in a network to become faulty. The faulty behavior considered in this paper is instead much more adversarial: faulty processors may deviate from their prescribed programs in an arbitrary fashion, perform an arbitrary amount of computation, and

[1] Standard networks are advantageous to consider in that they allow one to focus on the novel characteristics of Byzantine agreement without. being distracted by legitimate but "orthogonal" concerns. We wish to stress, however, that, while the absence of broadcasting is crucial for the problem of Byzantine agreement to be meaningful, we shall see in section 9.1 that most of the fine details of the adopted communication model can be significantly relaxed without affecting our result.

[2] Notice that we have stated Byzantine agreement a bit more generally than in the motivating scenario; namely, the processors are not given their initial values by a distinguished member of their group but have their own individual sources for these values. Consider, for instance, the case of party bosses who, before an election, call each other on the phone to select a common candidate to back: even though their initial choices do not, arise from the suggestion of a distinguished boss, they still need Byzantine agreement.

even coordinate their disrupting actions. Modeling software and hardware faults as malicious behavior may not be unreasonable when dealing with the interaction of very complex programs, but it is actually very meaningful, and even mandatory, if there are people behind their computers. Indeed. whenever we wish to emphasize the possibility of human control—and thus that of malicious behavior—we do employ the term "player" instead of processor.

The goal of the faulty players is to disrupt either the consensus or the validity requirement or simply to delay reaching Byzantine agreement for as long as possible (as when, say, they prefer the status quo to any of the two alternatives being voted on). Here is an example of what malicious players may do against a simple-minded protocol.

Assume that the honest generals of the motivating scenario try to reach agreement as follows: they send their initial orders to each other and then execute the most "popular" order. Then the dishonest generals can easily cause disagreement. To make our example more dramatic, let us suppose that 2/3 of the generals are loyal, that half of the loyal ones start with the value "attack," and that the other loyal half start with "retreat."[3] In this situation, the traitors simply tell every loyal general in the first half that their initial value is "attack" and every loyal one in the second half that their value is "retreat." Consensus is then disrupted in a most dramatic way: half of the loyal generals will attack and the other half will retreat. Indeed, reaching Byzantine agreement is a tricky business.[4]

The *significance* of Byzantine agreement. Byzantine agreement is widely considered the standard bearer in the field of fault-tolerant distributed computation. While it is indisputable that this problem has attracted an enormous amount of attention, we are skeptical about its relevance in the context of errors naturally occurring in a distributed computation. In our opinion, Byzantine agreement is relevant to the field of secure *computation* protocols, which includes problems such as electronic elections, electronic negotiations, or electronic bids.

Secure protocols (see [30] for a satisfactory and general definition) is a new and exciting branch of mathematics that has experienced impressive growth in recent years. A problem in this field consists of enabling a group of mutually distrustful parties to achieve, in a interaction in which some of the players do not follow the rules of the game, the same results that are obtainable by exchanging messages in a prescribed manner when there is total and honest collaboration. Indeed, it is thanks to insights from the field of secure prolocols that we have succeeded in finding our optimal probabilistic solution to the synchronous Byzantine agreement problem.

Byzantine agreement plays an important role in secure protocol theory; essentially, it dispenses with the need to hold a meeting when, because of the presence of adversaries among us, it is useful to establish in a public manner who said what or what was decided upon. In the simplest secure protocol or in the most complex one,

[3] These initial values are not at all unlikely if they represent (as in our motivating scenario) the individual version of an alleged unique message sent by a dishonest party. In any case, consensus and validity are very strong requirements: they should hold for any initial values!

[4] As we shall mention in the next section, at least t rounds of communications are needed to reach Byzantine agreement whenever (1) t parties are dishonest and (2) the honest ones follow a deterministic protocol. This fact immediately yields an alternative way to dismiss the simple-minded strategy discussed above: it is deterministic and can be implemented in two rounds, no matter how many players there are and no matter how many of them can be faulty. The same fact allows one to dismiss a good deal of other simple-minded strategies as well. As we shall see, it is only through a careful use of randomization that a strong majority of honest players may reach Byzantine agreement very fast.

the honest players cannot possibly make any progress without keeping a meaningful and consistent view of the world. This is what Byzantine agreement gives us.

The quality of a Byzantine agreement protocol. Several aspects are relevant in determining the quality of a Byzantine agreement protocol. As for most protocols, the amount of local computation and the total number of message bits exchanged continue to be important. But in this archetypal problem in distributed adversarial computation, two are the most relevant (and most investigated) aspects: the *round complexity* and the *fault model.*

The round complexity measures the amount of interaction that a protocol requires.[5] Since, at each clock tick, a player may send messages to more than one processor (and their recipients will receive them by the next tick), the round complexity of a protocol naturally consists of the total number of *rounds* (i.e., clock pulses and thus "waves" of messages) necessary for completing the protocol.

The fault model specifies what can go wrong (while still being tolerable somehow) in executing a protocol, namely the following: How many processors can become faulty? How much can they deviate from their prescribed programs? How long can faulty processors compute to pursue their disruptive goals?

In this light: the goal of a Byzantine agreement protocol naturally consists of simultaneously "decreasing" the round complexity while "increasing" the fault model.

Our solution. We present *a* probabilistic-polynomial-time protocol that, reaches Byzantine agreement in an expected constant number of rounds (thus minimizing the round complexity) while tolerating the maximum possible number of faulty players and letting them exhibit a most malicious behavior.

2. Previous solutions and ours.

2.1. The worst natural fault model.

Though several weaker models for Byzantine agreement can be considered (see the excellent surveys of Fischer [19] and Chor and Dwork [11] for a more comprehensive history of this subject), in this paper, we concentrate on a most adversarial setting. Speaking informally for now, the *worst (natural) fault model* is characterized by the following three conditions:

1. the good players are bound to polynomial-time computation;
2. a constant fraction of the total number of players may become faulty; and
3. the faulty players can deviate from their prescribed instructions in any arbitrary way, perform arbitrarily long computations, and perfectly coordinate their actions.

The worst fault model is not only the most difficult one to handle but, also, in our opinion, the most meaningful one to consider. Condition 1 essentially expresses that for a Byzantine agreement protocol to be useful, the computational effort required by the honest processors should be reasonable. Condition 2 properly captures our intuition about the nature of faults, independently of whether we consider players as machines or people controlling machines. Indeed, while we do expect that the number of faulty players grows with the size of the network, it would be quite counterintuitive to expect that it grows sublinearly in this size. (For instance, assume that in a network of n players the number of bad ones is $n/\log n$. Then this would mean that, while we expect 1% of a group of 1000 players to be faulty, we expect a *smaller* percentage of faulty players in a much larger group.) Condition 3 essentially captures that there may be people behind their computers: dishonest people follow whatever strategy is

[5] In a distributed setting, this is the most expensive resource. Typically, the time invested by the processors for performing their local computation is negligible with respect to the time necessary to send electronic mail back and forth several times.

best for them, try much harder than honest ones, and effectively cooperate with one another. In any case, by successfully taming *malicious* faults, we would a fortiori succeed in taming all other more benign—though not necessarily more reasonable—ones.

Let us thus review the main protocols in this difficult model.

2.2. Previous solutions. Dolev et al. [12] exhibited the first solution in the worst fault model. Letting n denote the total number of players in the network and t denote an upperbound on the number of faulty players, they showed that as long as $t < n/3$, Byzantine agreement can be reached deterministically in $\Theta(t)$ rounds. Recently, by a different protocol, Moses and Waarts [31] tightened their number of rounds to be $t + 1$ for $t < n/8$. This is optimal for their choice of t, since Fischer and Lynch [20] proved that t rounds are required by any deterministic protocol if t faults may occur in its execution.

In light of the lower bound mentioned, all hope for faster agreement is entrusted to probabilism. Indeed, since the pioneering work of Ben-Or [5], randomization has been extensively used for reaching agreement. In particular, Rabin's notion of a *common coin* [34] has emerged as the right version of probabilism for this setting. A network with a common coin can be described as a network in which a random bit becomes available to all processors at each round but is unpredictable before then. The interest of this notion is due to a reduction of Rabin showing that as long as $t < n/4$, Byzantine agreement can be reached in expected constant number of rounds with the help of a common coin. Of course, common coins are not a standard feature of a point-to-point network; thus this reduction raises a natural and important question: *Are there efficient Byzantine agreement protocols implementable "within the network" and in the worst fault model?*

Prior to our work, no efficient within-the-network Byzantine agreement protocol was known for the worst fault model. Rabin [34] devised a cryptographic Byzantine agreement protocol running in an expected constant number of rounds but relying on a incorruptible party external to the network.[6] Bracha [6] exhibited Byzantine agreement protocols that do not require trusted parties, but his protocols are slowers (they run in expected $O(\log n)$ rounds) and are not explicitly constructed (their existence is proved by counting arguments). Chor and Coan [8] exhibit an explicit and within-the-network Byzantine agreement protocol, but their solution, though attractively simple, is much slower (their protocol tolerates any $t < n/3$ faults but runs in $O(t/\log n)$ rounds; thus it requires expected $O(n/\log n)$ rounds in the worst fault model). Feldman and Micali [18] explicitly exhibited a cryptographic within-the-network protocol that, after a preprocessing step consisting of a single Byzantine agreement (on some specially generated keys), allows any subsequent agreement to be reached in an expected constant number of rounds.[7] While their protocol is actually very practical after the first agreement has been reached, the first agreement may

[6] Rabin's algorithm uses digital signatures—which implies that dishonest processors are bound to polynomial-time computation—and a *trusted party*—i.e., an incorruptible processor outside the network. In his solution, if the trusted party distributes k pieces of reliable information to the processors in the networks in preparation, then these processors can, subsequently and without any external help, compute k common coins. Thus the number of reachable agreements is bounded by the amount of information distributed by the trusted party in the preprocessing stage. A cryptographic Byzantine agreement protocol with a trusted party but without the latter limitation was later found by the authors in [18], in addition to other results mentioned later on.

[7] Thus their protocol does not require any preprocessing if a trusted party distributes the right keys beforehand. The present result can thus be viewed as removing cryptography and preprocessing from their protocol.

very well be the most important one (i.e., whether or not to hold a meeting).

To complete the picture, let us mention that Dwork, Shmoys, and Stockmeyer [16] found a beautiful Byzantine agreement protocol running in expected constant round but not in the worst fault model. (Their algorithm tolerates only $O(n/\log n)$ faults.)

2.3. Our solution. The main theorem in this paper can be informally stated as follows:

There exists an explicit protocol $\mathcal{P}$ reaching Byzantine agreement in the worst fault model and running in in an expected constant number of rounds. Protocol $\mathcal{P}$ actually tolerates any number of faults less than one third of the total number of processors.

Our protocol is probabilistic in the "best possible way": it is *always correct* and *probably fast;* that is, an unlucky sequence of coin tosses may cause our protocol to run longer, but when it halts both consensus and validity are guaranteed to hold. Our algorithm not only exhibits optimal (within a constant) round complexity, but it also achieves optimal fault tolerance. In fact, Karlin and Yao [28] have extended the earlier deterministic lower bound of [32] by showing that even probabilistically Byzantine agreement is unreachable if $t = n/3$ faults may occur.[8]

3. Model of computation. As of today, unfortunately, no reasonable treatment of the notion of probabilistic computation in a malicious fault model can be conveniently pulled off the shelf. (A comprehensive effort in this direction—in the more general context of *secure computation.*—*was* made in [30], but this paper has not yet appeared in print.[9]) Thus we have found it *necessary* to devote a few pages to discuss—though only at a semiformal level— of definitions in what we intended to be a purely algorithmic paper.

The definitions below, presented only at a semiformal level, focus solely on what we immediately need to discuss our Byzantine agreement protocol, purposely ignoring many other subtle issues (addressed in the quoted paper [30]). We only wish to clarify what it means that, in the execution of an n-party protocol, t of the processors may make errors (i.e., deviate from their prescribed instructions) in a most malicious way and that the protocol tolerates these faults.

Basic notation. Below we assume that a proper encoding scheme is adopted. Thus. we can treat a string or a set of strings over an arbitrary alphabet as a binary string, we may consider algorithms that output (the encoding of) another algorithm, etc.

We assume that each finite set mentioned in our paper is ordered. If $S_1, \ldots, S_k$ are finite sets, we let the instruction $\forall x_1 \in S_1 \ldots \forall x_k \in S_k\ Alg(x_1, \ldots, x_k)$ stand for the program consisting of running algorithm ***Alg*** first on input the first element of $S = S_1 \times \ldots \times S_k$, then ("from scratch," i.e., in a memoryless fashion) on input the second element of S, and so on.

[8] This remains true, as proved by Dolev and Dwork [14], even if one abandons the worst fault model so as to include cryptographic protocols (against faulty processors with polynomially bounded resources). Thus the optimality of our algorithm is retained in this setting as well.

[9] Byzantine agreement aims only at guaranteeing *correctness* in the presence of an adversary (about what was decided upon) but, not at keeping secret the original single-bit inputs of the players. A secure protocol must instead simultaneously ensure that a given computation (on inputs some of which are secret) is both correct and *private*, that. is, roughly, not revealing the initially secret individual inputs more than is implicitly done by the desired output of the computation. This is much more difficult both to handle and to formalize.

The symbol ":=" denotes the *assignment* instruction. The symbol "$\circ$" denotes the concatenation operator. If a is a string and τ is a prefix of a, we denote by the expression "σ/τ" the string p such that $a = \tau \circ p$.

If *Alg* is a probabilistic algorithm, I is a string, and R is a infinite sequence of bits, by running *running Alg on input I and coins R* we mean the process of executing *Alg* on input I so that, whenever *Alg* flips a coin and $R = bR'$, the bit b is returned as the result of the coin toss and $R := R'$.

Protocols. To avoid any issue of nonconstructiveness, we insist that protocols be uniform.

DEFINITION 1. *Let n be an integer greater than 1. An* n-party protocol *is an n-tuple of probabilistic algorithms, $P_1, \ldots, P_n$, where each P_i (which is intended to be run by player i) satisfies the following property. On any input (usually representing player i's previous history in an execution), P_i halts with probability 1 computing either an n-tuple of binary strings (possibly empty, representing i's messages to the other players for the next round) or a triple consisting of an n-tuple of strings (with the same interpretation as before), the special character TERMINATE, and value v (as its output).*

Notice that each time that P_i is run, one also obtains as "side products" the sequence of coin tosses actually made by P_i and the sequence of its "future" coin tosses.

A protocol *is a probabilistic algorithm that, for all integers* n > 1, *on input the unary representation of* n, *outputs (the encoding of) an n-party protocol.*

In this paper, the expression *round* denotes a natural number; in the context of an n-party protocol, the expression *player* denotes an integer in the closed interval $[1, n]$.

Executing protocols without adversaries. Let us first describe the notion of executing a protocol when all players are honest. Intuitively, each party runs his own component of the protocol. The only coordination with other parties is via messages exchanged in an organized fashion. Namely, there is a common clock accessible by all players, messages are sent at each clock tick along private communication channels, and they are received by the next tick. The interval of time between two consecutive ticks is called a *round.* At the beginning of a round, a player reads the messages sent to him in the previous round, and then runs (his component of) the protocol to compute the messages he sends in response. These outgoing messages are computed by a player by running the protocol on the just-received incoming messages and its own past "history," (i.e., an encoding of all that has happened to the player during the execution of the protocol up to the last round). We now describe this intuitive scenario a bit more precisely, though not totally formally. In so doing, parties, hardware, private communication channels, and clocks will disappear. However, they will remain in our terminology for convenience of discourse.

DEFINITION 2. *Let n be an integer > 1, $P = (P_1, \ldots, P_n)$ be an n-party protocol, $p_1, \ldots, p_n$ be finite strings, and $R_1, \ldots,$ R, be infinite binary sequences. Then by* executing *P* on private inputs $p_1, \ldots, p_n$ and coins $R_1, \ldots, R_n$, *we mean the process of generating, for each player i and round r, the quantities*

- H_i^r, *a string called the* history of player i at round r *(a triple consisting of (1) i's history prior to round r,* **(2)** *the messages received by i in round r, and* **(3)** *the coin tosses of i in round r),*
- $M_{i\rightarrow}^r$, *the* messages sent by player i in round r *(an n-tuple of strings whose jth entry, $M_{i\rightarrow}^r[j]$, is called* the message sent by i to j in round r),
- $M_{\rightarrow i}^r$, the messages received by player i at round r *(an n-tuple of strings, whose*

jth entry, $M^r_{\rightarrow i}[j]$, is called the message received by i from j in round r),

- C^r_i, the coin tosses of i in round r *(a substring of R_i), and*
- R^r_i, the coin tosses of i after round r *(a substring of R_i)*

by executing the following instructions:

(Start) *Set* $C^0_i = \text{E}$, $R^0_i = R_i$, $M^0_{i\rightarrow} = M^0_{\rightarrow i} = (\text{E}, \ldots, \epsilon)$, *and* $H^0_i = (p_i, M^0_{\rightarrow i}, C^0_i)$.

"Only the individual input is available at the start of an execution: no message has yet been sent or received: and no coin has been flipped."

(Halt) *Say that* player i halts at round r *(and his* output is a*) if r is the minimum round s for which P_i, on input H^{s-1}_i and coins R_i, computes a triple whose second entry is the special character* TERMINATE *(and whose third entry is o.) If i halts in round r, then $\forall s > r$, $M^s_{i\rightarrow} := (\epsilon, \ldots, \epsilon)$, $M^s_{\rightarrow i} := (M^s_{1\rightarrow}[i], \ldots, M^s_{n\rightarrow}[i])$, $C^s_i := \epsilon$, $R^s_i := R^r_i$, and $H^s_i := (H^{s-1}_i, M^s_{\rightarrow i}, \text{E})$.*

(Continue) *If i has not halted in a round $< r$, run P_i on input H^{r-1}_i and coins R^{r-1}_i so as to compute either* (a) *an n-tuple of string M or* (b) *a triple $(M,$*TERMINATE*$, v$ where* M *is an n-tuple of strings,* TERMINATE *is a special character, and v is a string. If C is actually the entire sequence of coin tosses that P_i has made in this computation—and thus C is a prefix of R^{r-1}_i—then $C^r_i := C$, $R^r_i := R^{r-1}_i/C$, $M^r_{i\rightarrow} :=$* Ad*, $M^r_{\rightarrow i} := (M^r_{1\rightarrow}[i], \ldots, M^r_{n\rightarrow}[i])$, and $H^r_i := (H^{r-1}_i, M^r_{\rightarrow i}, C^r_i)$.*

For simplicity's sake (since each P_i, on any input, halts with probability 1), above we have neglected dealing with protocol "divergence." Also for simplicity, we let a player, at each round, run his own version of the protocol, P_i, on the just-received messages and on the entire history of his execution of the protocol. This is certainly wasteful. In most practical examples, in fact,, it suffices to remember very little of the past history. Also notice that the current round number is not an available input to P_i, but it can be easily derived from the current history. In our protocols, however, we make players very much aware of the round number. In fact, we actually spell out what each P_i should do separately for each round. Notice also that the strings R_i need not to be given "in full." It suffices that a mechanism is provided that "retrieves and deletes" R_i's first bit.

Adversaries. We now allow malicious errors to occur in the execution of a protocol. A processor that has made an error is called *faulty* or *bad.* To formalize the idea that faulty processors may coordinate their strategies in an optimal way, we envisage a single external entity, *the adversary,* that chooses which processors to corrupt and sends messages on behalf of the corrupted processors. Since we wish our adversary to be as strong as possible, we allow it to be a nonuniform probabilistic algorithm. (In fact, in our protocol, we might as well assume that an adversary is an arbitrary probabilistic noncomputable function.)

DEFINITION 3. *Let n be an integer. greater than* 1. *An* n-party adversary *is a probabilistic algorithm that: on. any input (usually representing A's previous activity in an execution) halts with probability 1 and outputs either an integer in the range $[1,n]$ (the identity of a newly corrupted player) or a sequence of pairs (j,M), where j is an integer between* 1 *and* n *(the identity of a corrupted player) and M is an n-tuple of strings (the messages sent by* j *in the current round). An* adversary, A, *is* a *sequence of n-party adversaries: $A = \{A(n) : n = 2,3,\ldots\}$.*

Executing protocols uiith an, adversary. We now define what it means for an n-party protocol P to be executed with an n-party adversary A. A enters the execution with an *initial adversarial history,* a string denoted symbolicallg by H^0_A, and an *initially bad set,* $\text{BAD}^0 \subset [1,n]$. String H^0_A may contain some a priori knowledge about the inputs of the players, the result of previous protocols, and so on. Set BAD^0 represents the players corrupted at round 0, that is, before the protocol starts. (In

other words, if **P** were the first protocol "ever to be executed," BAD^0 would be empty. If, as we shall see, **P** were called **as** a subprotocol, BAD^0 would comprise all the players that have been corrupted prior to calling **P.)** Adversary *A* may, at any round, *corrupt* an additional processor, **j.** When this happens, all of j's history becomes available to A;[10] as for all corrupted processors, all future messages sent to **j** will be read by *A;* and ***A*** will also compute all of the messages that **j** will be sending. Essentially, **j** becomes an extension of ***A.*** Thus if $k \in \text{BAD}^0$, *k*'s private input is what becomes available to ***A*** at the start of **P,** and *A* will totally control player k for the entire execution of **P.**

Since we want to prepare for the worst, we let the adversary be even more powerful by allowing *rushing;* that is, we let the message delivery (which is not simultaneous) be as adversarial as possible. At the beginning of each round, all currently good players read the messages sent to them in the previous round and compute the ones that they wish to send in the present round. We pessimistically assume that the messages addressed to the currently corrupted processors are always delivered immediately, and if based on this information the adversary decides to corrupt an additional processor j, we pessimistically assume that it succeeds in doing so before **j** has sent any messages to the currently good players, thus giving A a chance to change these messages. Further, we consistently "iterate this pessimism" within the same round. That is, once **j** is corrupted in round r, we assume that the messages addressed to **j** by the currently good processors are immediately delivered, while **j** has not yet sent any messages to the remaining good players. This way *A* may decide whom to corrupt next in the same round, and so on, until ***A*** does not wish to corrupt anyone else in round r. At this point, ***A*** computes all messages sent by the corrupted processors in round r. These "bad" messages will be read by the good processors (of course, each processor receives the messages addressed to him), together with all "good" messages, at the beginning of round $r + 1$.

The privacy of the communication channels of a concrete network is captured in the formulation below by the fact that messages exchanged between uncorrupted processors are never an available input to the adversary algorithm.

The history of a bad player is essentially frozen at the moment in which he is corrupted because ***A*** has essentially subsurned him from that point on.

DEFINITION 4. *Let n be an integer* > 1, $H_A^0, p_1, \ldots, p_n$ *be finite strings,* $R_A, R_1, \ldots, R_n$ *infinite binary sequences,* BAD^0 *be a subset of* $[1,n]$ *and* GOOD^0 *be its complement,* $P = (P_1, \ldots, P_n)$ *be an n–party protocol, and A be an n-party adversary. Then by* executing *P* with ***A*** on initial adveraarial history H_A^0, inputs $p_1, \ldots, p_n$, initially bad set BAD^0, and coins R_A and $R_1, \ldots, R_n$, *we mean the process of* (i) *generating, for all players i and rounds r, the quantities*

- H_i^r, $M_{i\rightarrow}^r$, $M_{\rightarrow i}^r$, C_i^r, *and* R_i^r (*whose interpretation, as well as their setting for* $r = 0$, *is the same as in Definition* **2)**

and the new quantities

- H_A^r *(a string called the* history of the adversary at round *r),*
- C_A^r *(a binary string called* the coin tosses of the adversary at round *r),*
- R_A^r *(an infinite subsequence of* R_A *called* the coin tosses of ***A*** after round r**),** *and*
- BAD^r *and* GOOD^r *(two sets of players called, respectively, the* bad players at round *r and the* good players at round *r, such that* $\forall r$, $\text{GOOD}^r = [1,n] - \text{BAD}^r$)

[10] This is a clean but pessimistic approach (which makes our result stronger). In practice, though *j* may wish to fully collaborate with ***A*** by sharing all information he has. he may still have trouble in remembering—say—all previously received messages or all previously made coin tosses.

by setting $C_A^0 = \epsilon$ *and* $R_A^0 = R_A$ *and* (ii) *executing the following instructions for* $r = 1, 2, \ldots$:

0. $\text{TEMPH}_A^r := H_A^{r-1}$; $\text{TEMPR}_A^r := R_A^{r-1}$; $\text{TEMPGOOD}^r := \text{GOOD}^{r-1}$; $\text{TEMPBAD}^r := \text{BAD}^{r-1}$.
   "Because A's history, future coin tosses, and sets of good and bad players dynamically change within a round, we shall keep track of these changes in temporary variables. However, their final values within round r, respectively, H_A^r, R_A^r, GOOD^r, *and* BAD', *are unambiguously defined."*
1. *"Just as when all processors are honest,"* $\forall g \in \text{GOOD}^{r-1}$, *generate* $M_{g\rightarrow}^r$, *"the messages that g wishes to send in this round (which may be reset if g is corrupted in this round),"* C_g^r, *and* R_g^r *by running* P_g *on input* H_g^{r-1} *and coins* R_g^{r-1}.
2. $\forall g \in \text{GOOD}^{r-1}$ and $\forall b \in \text{BAD}^{r-1}$, $\text{TEMPH}_A^r := (\text{TEMPH}_A^r, g, b, M_{g\rightarrow}^r[b])$.
3. *Run A on input* TEMPH_A^r *and coins* TEMPR_A^r.
   If in this execution of step **3** *A has output* $j \in \text{TEMPGOOD}^r$ *and made the sequence of coin tosses C, then*
   - $\text{TEMPBAD}^r := \text{TEMPBAD}^r \cup \{j\}$, $\text{TEMPGOOD}^r := \text{TEMPGOOD}^r - \{j\}$,
   - $\text{TEMPH}_A^r := (\text{TEMPH}_A^r, H_j^{r-1}, C_j^r, C)$ *"so that from* H_j^{r-1} *and* C_j^r *A can reconstruct, all of the messages that j wished to send in round r, and from* TEMPH_A^r *and C she can reconstruct why she has corrupted j,"*
   - $\text{TEMPR}_A^r := \text{TEMPR}_A^r/C$ *"adjust A's future coin tosses,"*
   - $\forall g \in \text{TEMPGOOD}^r$, $\text{TEMPH}_A^r := (\text{TEMPH}_A^r, g, j, M_{g\rightarrow}^r[j])$, *"i.e., according to rushing, A is also given the messages that the currently good players wish to send to j in this round," and*
   - *go to step* **3** *"to corrupt next processor."*

   Otherwise, if in this execution of step 3, A has output, $\forall b \in \text{TEMPBAD}^r$, *a vector* $M_b \in (\{0,1\}^*)^n$ *"as b's round-r messages"*[11] *and made the sequence of coin tosses C, then*
   - $\forall b \in \text{BAD}^r$, $M_{b\rightarrow}^r := M_b$,
   - $\text{TEMPH}_A^r := (\text{TEMPH}_A^r, C)$ *"so that she can reconstruct the bad players' messages of round r," and*
   - $\text{TEMPR}_A^r := \text{TEMPR}_A^r/C$, *and "adjust the final round-r quantities as follows."*
4. *Letting C be the sequence of coin tosses A has made since the last execution of step 2,*
   - $H_A^r := \text{TEMPH}_A^r$; $C_A^r := C$; *and* $R_A^r := \text{TEMPR}_A^r$;
   - $\text{GOOD}^r := \text{TEMPGOOD}^r$ *and* $\text{BAD}^r := \text{TEMPBAD}^r$;
   - $\forall i, j \in [1, n]$, $M_{\rightarrow i}^r[j] := M_{j\rightarrow}^r[i]$;
   - $\forall g \in \text{GOOD}'$, $H_g^r := (H_g^{r-1}, M_{\rightarrow g}^r, C_g^r)$;
   - $\forall b \in \text{BAD}^{r-1}$, $H_b^r := (H_b^{r-1}, bad)$, *and* $\forall b \in \text{BAD}^r - \text{BAD}^{r-1}$, $H_b^r := (H_b^{r-1}, C_b^r, bad)$.

Let E be the sequence (of tuples of quantities resulting from the above computation) so defined:

$$E = E_0, E_1, \ldots,$$

where

$$E_r = (H_1^r, M_{1\rightarrow}^r, M_{\rightarrow 1}^r, C_1^r, R_i^r, \ldots, H_n^r, M_{n\rightarrow}^r, M_{\rightarrow n}^r, C_n^r, R_n^r, H_A^r, C_A^r, R_A^r, \text{BAD}^r, \text{GOOD}^r).$$

[11] By convention, if A's output is not of this format, then it is assumed that $M_b = (\epsilon, \ldots, \epsilon)$ $\forall b \in \text{TEMPBAD}^r$.

We call E the execution of P with A on initial quantities H_A^0, BAD^0, and $p_1, \ldots, p_n$, and coins R_A and $R_1, \ldots, R_n$. *The value* E_r *is called* round r of E. *If* R *is a positive integer, by the expression E* up to round R, *in symbols* $E_{[0,R]}$, **we** *mean the finite subsequence* $E_0, \ldots, E_R$.

(Note: *The quantities* H_i^r, $M_{i\rightarrow}^r$, $M_{\rightarrow i}^r$, C_i^r, R_i^r, H_A^r, C_A^r, R_A^r, BAD', *and* GOOD^r *may carry an additional superscript or prefix to emphasize the protocol during the execution of which they have been generated.)*

Remark. The ability of an adversary to corrupt players at arbitrary points in time of a protocol is crucial in a randomized protocol. For a deterministic protocol, the adversary's optimal strategy may be calculated beforehand, but it may profitably change during the execution of a randomized protocol. For example, consider a probabilistic protocol for randomly selecting a "leader," that is, a processor to be put in charge of a given task. Depending on the specifics of the protocol, it may be impossible for the adversary to corrupt a few players beforehand and coordinate their actions so that one of them is guaranteed to be elected leader. It is, however, very easy for her to wait and see which processor is selected as leader and then corrupt it! (This feature models a "real-life" phenomenon: nobody is born a thief, but some may become thieves if the right circumstances arise .... To capture this realistic feature, we must allow'—and successfully deal with—adversaries that can corrupt players, during run time, in a dynamic fashion.)

Fractional adversaries. Above we have presented the mechanics of executing a protocol with an adversary exhibiting what is essentially an arbitrarily malicious behavior. To keep things meaningful, however, we wish to put a cap on the *number* of players that an adversary may corrupt without otherwise limiting its actions in any way. In fact, we assume that no n-party adversary may corrupt n players in an execution with an n-party protocol (otherwise, no meaningful property about such an execution 'could possibly be guaranteed).

We will actually be focusing on adversaries that may corrupt at most a constant fraction of the players. Let c be a constant between 0 and 1; we say that an adversary A is a *c-adversary* if for all $n > 1$ and all n-party protocols $\mathbf{P}$, in any execution with $\mathbf{P}$ on an initially bad set with $< cn$ elements, the cardinality of the bad set always remains $< m$. Whenever we consider an execution of an n-party protocol with a c-adversary, we implicitly assume that the initially bad set contains less than cn players.

We also assume that no more than one adversary is active in an execution of a protocol. Actually, because the adversary that never corrupts any processor is a special type of adversary (indeed, a c-adversary for all possible $c \in (0,1)$), we shall assume that in every execution of a protocol there is *exactly one adversary* active. Thus the expression "an execution of protocol $\mathbf{P}$ really means "an execution of protocol $\mathbf{P}$ with adversary A, for some adversary $\boldsymbol{A}$."

Initial quantities. As we have seen, to run an n-party protocol with an n-party adversary A, we need to specify, other than the coin tosses of $\boldsymbol{A}$ and the n players, the following initial quantities: **(1)** the initial adversarial history H_A^0, (2) the initially bad set BAD', and **(3)** the inputs $(p_1, \ldots, p_n)$. For the purpose of defining the mechanics of executing an n-party protocol with an n-adversary, we define $\mathcal{IQ}_n$, the set of *the initial quantities (of size n)* in a most "liberal" manner; that is, $\mathcal{IQ}_n = \{0,1\}^* \times 2^{\{1,\ldots,n\}} \times (\{0,1\}^*)^n$. In an accordingly liberal manner, we let $\mathcal{IQ} = \{\mathcal{IQ}_n : n > 1\}$ be the set of *all (possible) initial quantities.*

In general, however, it is meaningful to prove properties of protocols if the initial quantities of their executions satisfy a given constraint (e.g., reaching Byzantine

agreement on "the message" sent by a given member of a network is meaningful only if the identity of this sender is a common input to all processors in the network). We actually prefer to dismiss nonmeaningful initial quantities from consideration altogether. That is, we *define* each n-party protocol $P(n)$ together with the set of its own *proper* initial quantities, denoted by $\mathcal{IQ}_n^P$, on which—and solely on which—$P(n)$ can be run. Thus whenever we say that some specific values IQ are initial quantities for $P(n)$, it is assumed that $\mathrm{IQ} \in \mathcal{IQ}_n^P$. Also, whenever we refer to an execution of a protocol P with some specific initial quantities IQ, if $\mathrm{IQ} \in \mathcal{IQ}_n$, we actually refer to an execution of $P(n)$ on initial quantities IQ—quantities which actually belong to $\mathcal{IQ}_n^P$. (Indeed, it should be noticed that n can easily be computed from any member of $\mathcal{IQ}_n$.) In summary, all initial quantities of a protocol are deemed to be proper—and we shall use the expression "proper" only for emphasis.

Notice that by specifying the proper initial quantities of a given protocol, one could easily "cheat" by disallowing certain initial adversarial histories or initially bad sets so as to make protocol design artificially easy. In this paper, however, the proper initial quantities of a protocol will never in any way constrain the initial adversarial history or the initially bad set, except for its cardinality. Moreover, in this paper, proper initial quantities will never impose any restrictions on the inputs of the initially corrupted players. When defining a new protocol, though, we find it convenient to describe the generic element of its (proper) initial quantities by specifying (in particular) the inputs of all players, with the understanding that all constraints on the initially bad players must be dropped; that is, by saying that $(H_A^0, \mathrm{BAD}^0, (p_1, \ldots, p_n)) \in \mathcal{IQ}_n^P$, we simply mean that the private input of player i is p_i if i does not belong to BAD^0. (In other words, if we wish a more extensive notation, an element of $\mathcal{IQ}_n^P$ is of the form $(H_A^0, \mathrm{BAD}^0, \{(i, p_i) : i \notin \mathrm{BAD}^0\})$.)

Random executions and probabilities.

DEFINITION 5. *Let n be an integer > 1, P be an n-party protocol, A be an n-party adversary, and* $\mathrm{IQ} \in \mathcal{IQ}_n^P$. *By* randomly executing P with *A* on initial quantities IQ, *we mean the process consisting of generating the infinitely long bit sequences R_A, $R_1, \ldots, R_n$ by randomly and independently selecting each of their bits in $\{0,1\}$ and then executing* P *with A on initial quantities* IQ *and coins R_A, $R_1, \ldots, R_n$. We call the execution resulting from this process a* random execution *of* P *with A on initial quantities* IQ.

Thus the probability that an event *e* occurs in a random execution of P with *A* on initial quantities IQ is solely computed over the coin tosses of P and *A*. (Only if we have assumed a probability distribution on the private inputs as well—and if we explicitly say so—we may compute the probability of an event also over the random choices in selecting the private inputs.) The probabilities of events that are most important to us are those that are *intrinsic* properties of our protocols *alone;* that is, we shall prove bounds for these probabilities that are valid for any adversaries, any initial adversarial history, any initially corrupted players, and any players' inputs.

Fault tolerance. The fault tolerance of a protocol is essentially the highest fraction of faults it can tolerate.

DEFINITION 6. *Let Ψ be a property (i. e., a predicate) and* c *be a constant between* 0 *and* 1. *We say that a protocol P is a* c-fault tolerant *protocol (or a protocol* with fault tolerance *c)* with respect to Ψ *if $\Psi(E) = true$ for any execution E of P with a c-adversary.*

If the property Ψ is clear from context, we may simply say that P is a protocol *with fault tolerance* c rather than with fault tolerance *c* with respect to Ψ.

Legal shortcuts. For simplicity of discourse, we wish to "legalize" some handy

notation.

• Highlighting something. When we want to focus only on some of the quantities determining an execution, we just omit mentioning the others. For instance, the sentence "Let E be an execution of n-party protocol P with n-party adversary A on inputs $p_1, \ldots, p_n$ and initial corrupted set BAD^0" stands for "Let E be an execution of n-party protocol P with n-party adversary A on inputs $p_1, \ldots, p_n$, initially bad set BAD^0, initial adversarial history H_A^0, and coin tosses $R_1, \ldots, R_n$ and R_A, for some string H_A^0 and bit sequences $R_1, \ldots, R_n$ and R_A."

• *Matching* types. If P is an n-party protocol and we say that P is executed with an adversary A, we implicitly assume that A is an n-party adversary. Any adversary mentioned in the context of a protocol with fault tolerance c is meant to be a c-adversary. If P is a protocol and A is an adversary, by saying that n parties execute P with A, we mean that they execute $P(n)$ with adversary $A(n)$. By saying that a value IQ represents some initial quantities, we implicitly assume that $\text{IQ} \in \{0,1\}^* \times 2^{[1,n]} \times (\{0,1\}^*)^n$ for some positive integer n. By saying that a protocol P is executed with adversary A on initial quantities $\text{IQ} = (H_A^0, \text{BAD}^0, (p_1, \ldots, \text{p},))$, we mean executing $P(n)$ with $A(n)$ on initial adversarial history H_A^0, initially bad set BAD^0, and inputs $p_1, \ldots, p_n$. By an execution of protocol P with adversary A, we mean an execution of $P(n)$ with $A(n)$ for some number of players n (on some proper initial quantities).

Good, bad, and end. In an execution of a protocol, we say that processor i is good at round r if the adversary has not corrupted i at a round $\leq r$, and say that it is bad at round r otherwise. When, in an execution, the round under consideration is not specified, we say that a player i is currently good (respectively, currently bad)' to mean that it is good at round r (respectively, bad at round r) if the round under consideration is r. We say that i is eventually bad in an execution if it is corrupted at some round of it, and say that it is always good otherwise. When no confusion can arise, we may use the simpler expression good (respectively, bad) instead of currently or always good (respectively, currently or eventually bad).

We say that an execution of a protocol halts at round r if r is the smallest integer s for which every good processor has halted in a round $\leq s$. (Note: If an execution of an n-party protocol Q has not halted at round r, it continues to be considered an execution of an n-party protocol after that round, whether or not some of the good processors have halted by round r and no longer execute the protocol.) Let R be a constant and P be a protocol; we say that P is an R-round protocol if in all executions of P every good processor halts at round R. (Note: In every execution of an R-round protocol, all good processors halt "simultaneously," but if an execution of a protocol which is not R-round halts at round R, the good processors may not halt in the same round of that execution.) We say that P is a *fixed-round* protocol if it is an R-round protocol for some value R. All of our protocols, except for the last one, are fixed-round. We say that a protocol does not halt before round r if in all its executions no good processor halts at a round $\leq$ r.

Subprotocols. To facilitate the description of our Byzantine agreement protocol and to make it possible to use parts of it in other contexts, we have constructed it in a modular way. We thus need the notion of a subprotocol, that is, a protocol that is called as a subroutine by another protocol. Fortunately, in this paper, all subprotocols are fixed-round, they are called at rounds specified a priori by protocols (no execution of which halts by those rounds), and n-party protocols call only n-party subprotocols. (This simplifies our formalization somewhat; for instance, it makes it very clear when the call starts and when it ends.)

Let Q be a > r-round protocol calling an R-round protocol **P** at a prescribed round r. Then an execution of Q will be suspended once it reaches round r. At that point, the input value of each player i, p_i, is specified by either **P** itself (i.e., as when p_i is a constant) or player i's prior history, H_i^{r-1}. (If this is the case, we formally assume that there is a function $\mathcal{I}_i^P$ specified a priori, that, evaluated on H_i^{r-1}, determines p_i.) The execution of **P** on these inputs then starts. The good players execute **P as** if it were (rather than a subprotocol) "the first protocol they ever execute in their life," that is, their execution is independent of their prior histories. The adversary, on the other hand, is allowed to take advantage of what it has "learned" in the execution of Q to fine tune its strategy in the execution of P.[12] Moreover, should the adversary corrupt an additional player k during the execution of **P**, she will get, in addition to k's current history in the execution of P, its "suspended" history in Q. When **P** ends, each player appends its final "P-history" to its "suspended Q-history," and Q's computation is resumed. Processors corrupted in the execution of **P** are also considered corrupted in the resumed execution of Q.

DEFINITION **7.** *Let n be an integer > 1, A be an adversary, P be an n-party R-round protocol, Q be an n-party protocol calling P at round r, and R_A^1, R_A^P, R_A^2, $R_1^{Q1}, \ldots, R_n^{Q1}$, $R_1^P, \ldots, R_n^P$, $R_1^{Q2}, \ldots, R_n^{Q2}$ be infinite binary sequences. By executing Q with A on initial quantities IQ and coins*

$$R_A^1, R_A^P, R_A^2, R_1^{Q1}, \ldots, R_n^{Q1}, R_1^P, \ldots, R_n^P, R_1^{Q2}, \ldots, R_n^{Q2},$$

we mean the following:

1. To execute Q "a first time" with adversary A on initial quantities IQ and coins $R_A^1, R_1^{Q1}, \ldots, R_n^{Q1}$ "up to round r" so as to generate an execution up to round R, E^1, and thus quantities $H_A^{r,Q}$, $\text{BAD}^{r,Q}$, and $H_i^{r,Q}$ for each player i.

*2. (For each player i, we let p_i be the input specified by $H_i^{r,Q}$.) To generate an execution up to round R, E^2, by running **P** with A on initial inputs $p_1, \ldots, p_n$, initial adversarial history $H_A^{r,Q}$, initially bad set $\text{BAD}^{r,Q}$, and coins $R_A^P, R_1^P, \ldots, R_n^P$ as usual except for the following. If A corrupts a new player j at a round x, it receives as an input not only the history of player j in the present execution, $H_j^{x,P}$, but also $H_j^{r,Q}$ ("j's suspended history in Q").*

3. *To generate an execution E^3 by running Q with A on initial adversarial history $H_A^{R,P}$, initially corrupted set $\text{BAD}^{R,P}$, and inputs $(H_1^{r,Q}, H_1^{R,P}), \ldots, (H_n^{r,Q}, H_n^{R,P})$.*

We let the corresponding execution *(of calling protocol Q with A on the above initial quantities and coins) consist of the sequence E whose first r elements are the elements of E^1, next R elements are those of E^2, and remaining elements are those of E^3. (In other words, the execution of a protocol* Q *that calls an R-round subprotocol P at round r is obtained by identifying, for each $\rho \in [1, R]$, round ρ of P with round $r + \rho$ of Q.)*

The notion of randomly executing a protocol and that of a random execution of a protocols are extended in the natural way to a protocol that calls a subprotocol at a prescribed round.

The notion of a subprotocol is immediately generalized to allow *nesting* of subprotocols, that is, to allow Q itself to be a subprotocol. Assume that for $1 < x <$ k, protocol Q_x has called fixed-round protocol Q_{x+1} at round r_x. Then if protocol

[12] For instance, assume that a player j has been corrupted by **A** during the execution of Q before **P was** called. Then it is conceivable that from the Q-history of player j at the time of the call, the adversary may infer the Q-history at the time of the call of a good player **i** well enough to predict i's input to **P**. (That is, Q may induce some correlation among the inputs of subprotocol **P** that need not to be there if **P** were executed "from scratch.")

Q = Q_k calls P at round r, all of the mechanics for calling P, executing P, and including the result of P's execution in the Q-histories remain the same, except that if a new processor i is corrupted during the execution of P, the "suspended history" of player i learned by the adversary, rather than simply being $H_i^{r,Q}$, is actually $(H_i^{r,Q_1}, \ldots, H_i^{r_k Q_k})$.

Concurrent protocols. Since it is the goal of this paper to squeeze as much computation as possible into a few rounds, we need to introduce the notion of concurrently executing more protocols, each protocol on its own inputs.

DEFINITION. Let R be a positive integer, n be an integer > 1, and $\mathcal{L}$ be a finite set (of labels). Then an (n-party R-round) concurrent protocol *is a mapping from $\mathcal{L}$ into the set of n-party R-round protocols.*

For each $x \in \mathcal{L}$, we denote by P^x the image of x under P and by P_i^x the program of player i within P^x, that is, $P^x = (P_1^x, \ldots, P_n^x)$.

In an execution of a concurrent protocol P, the good players execute each of the P^x's *independently* of the others. This restriction does not apply to the adversary, who can make use of the information learned in the execution of one of the protocols to choose her actions in the execution of another. Moreover, if the adversary corrupts player i at round ρ of the execution of a protocol P^x, then i becomes corrupted in the execution of every other protocol in $\mathbf{P}$, but the total number of bad player increases only by one. Let us now be more precise.

DEFINITION 8. Let R be a positive integer, n be an integer > 1, $\mathcal{L}$ be a finite set, $P : x \in \mathcal{L} \to P^x$ be an n-party R-round concurrent protocol, A be an n-party adversary, H_A^0 be a string, R_A be an infinite binary sequence, BAD^0 *be a subset of* $[1,n]$, $p_1^x, \ldots, p_n^x$ *strings, and* $R_1^x, \ldots, R_n^x$ *be infinite binary sequences. By* executing P *(or, equivalently, by* concurrently executing $\forall x \in \mathcal{L}\ P^x$*)* with adversary A on initial adversarial history H_A^0, initially bad set BAD^0, inputs $p_1^x, \ldots, p_n^x$, and coins R_A and $R_1^x, \ldots, R_n^x$, *we mean performing the following instructions for each* player $i \in [1,n]$ *and each round* $r = 0, 1, \ldots$:

(a) $\forall i \in \text{GOOD}^{r-1}$ *and* $\forall x \in \mathcal{L}$, *compute* $M_{\to i}^{r,P^x}$, $M_{i\to}^{r,P^x}$, C_i^{r,P^x}, *and* H_i^{r,P^x} *from* $M_{\to i}^{r-1,P^x}$, $M_{i\to}^{r-1,P^x}$, *and* H_i^{r-1,P^x} *by running* P_i^x *so that the kth coin toss of* P_i^x *is the kth bit of* R_i^x.

(b) *Execute* $H_A^r := H_A^{r-1}$, *GOOD'* := GOOD^{r-1}, *BAD'* := BAD^{r-1}, *and* $\forall g \in$ GOOD^r, $\forall b \in$ *BAD',* $\forall x \in \mathcal{L}$, $H_A^r := (H_A^r, M_{g\to}^{r,P^x}[b])$.

(c) *Run* A *on input* H_A^r *so that A's kth coin toss is the k-bit of* R_A. *If* C *is the sequence of coin tosses made by A in this execution of step* **3**, *then* $H_A^r := (H_A^r, C)$. *If A outputs* $j \in$ *GOOD' in this execution of step* **3**, *then* BAD' := BAD' $\cup \{j\}$, **GOOD"** := GOOD$^r - \{j\}$, *and* $\forall g \in$ *GOOD',* $\forall x \in \mathcal{L}$, $H_A^r := (H_A^r, M_{g\to}^{r,P^x}[j])$, *and go to step* (c). *Else "A has output for each bad player b and label x an n-message vector M_b^x."*

(d) *Letting* C be *the sequence of coin tosses made by A since the last execution of step* (b), *set* $C_A^r = C$ *and* $\forall b \in$ BAD^r, $\forall x \in \mathcal{L}$, $M_{b\to}^r = M_b^x$.

The execution corresponding to the above process is $\mathrm{E} = E_1, \ldots, E_R$, *where*

$$E_r = (H_1^r, M_{1\to}^r, M_{\to 1}^r, C_1^r, \ldots, H_n^r, M_{n\to}^r, M_{\to n}^r, C_n^r, H_A^r, C_A^r, \text{BAD}^r, \text{GOOD}^r),$$

where $H_i^r = \{(x, H_i^{r,x}) : x \in \mathcal{L}\}$, $M_{i\to}^r = \{(x, M_{i\to}^{r,x}) : x \in \mathcal{L}\}$, $M_{\to i}^r = \{(x, M_{\to i}^{r,x}) : x \in \mathcal{L}\}$, *and* $C_i^r = \{(x, C_i^{r,x}) : x \in C\}$. *That is, each quantity relative to protocol* P^x *is labeled with x.*

The notions of randomly executing a concurrent protocol and that of a random execution of a concurrent protocol are obtained in the natural way.

A concurrent protocol $P : x \in \mathcal{L} \to P^x$ can be called at a prescribed round r by another protocol Q very much like an ordinary subprotocol. (In this case, the players

Q-histories at round r *must* specify, for each x $\in \mathcal{L}$, the inputs $p_1^x, \ldots, p_n^x$ on which to run protocol P^x; that is, $\forall x \in \mathcal{L}$, each Q_i^r specifies p_i^x.)

Sequenced protocols. We will also need the notion of a sequenced protocol. This consists of a pair of protocols (P, Q), where Q is run after P and on the histories of P. In this paper we actually need to consider only the case of sequenced protocols (P,Q), where P is R-round. Thus after an execution E of P, for all player i, i's input to Q consists of i's round-R history in E.

Like any other protocol, a sequenced protocol may be called as a subprotocol. Note that if (P, Q) is a sequenced protocol, then P and Q can never be executed concurrently. However, if $(P_1, Q_1), \ldots, (P_k, Q_k)$ are sequenced protocols, then it might be possible to concurrently execute $P_1, \ldots, P_k$ and then concurrently execute subprotocols $Q_1, \ldots, Q_n$, running each Q_i on the history of the execution of P_i.

Message bounds. As we have seen, at each round in the execution of a protocol $\mathbf{P} = (P_1, \ldots, P_n)$, the adversary sends a message to each currently good player g, which then feeds it to P_g (among other inputs). Thus by sending g arbitrarily long messages, the adversary could arbitrarily increase the amount of g's local computation. To meaningfully discuss complexity issues, we thus need to modify the mechanics of protocol execution by introducing message bounds.

The message bound is a variable internal to each processor that at each round evaluates to a positive integer or to $+\infty$ a special value greater than all positive integers. If at round r the message bound of a currently good player g is set to a positive integer k, then g is allowed to compute the k-bit prefix of any incoming message at round r in k computational steps; only after this truncation will a message become part of the input to P_g.

A simple and flexible way to specify the message bounds of a player at every round is to give him a special input, the message-bound input: in any execution in which the value of this input is v, a player sets to v the message bound of every round. (With an eye to complexity, these special inputs will be presented in unary; in fact, because we charge v steps for extracting the v-bit prefix of a string, we do not wish this operation to be exponential in the message-bound input.) Alternatively, a protocol can specify the message bound of round r within the code of round r itself—and thus may set it to a lower value when shorter messages are expected (from the good players). In either way, if it wishes to keep its own running time under control, a protocol must set the message bounds of each round to finite values. (If it fails to do so in even a single round, it will be in that round that the adversary will send extremely long messages.)

Let us now see what happens to message bounds if a protocol P calls a subprotocol Q. If Q has message-bound inputs, then P calls Q, specifying the values of these inputs, as for all other inputs of Q. If Q sets its own message bounds as part of its code at each round, then it is enough for P to call Q. In either case, throughout the execution of Q, Q's message bounds are to be enforced; only after the call is over and the execution of P is resumed will P's message bounds become effective again.

Complexity *measures.* We now wish to discuss the two notions of round complexity and local complexity. In so doing, we focus directly on the two cases that are really relevant to this paper; that is, constant round complexity and polynomially bounded local computation. We leave to the reader—if she so desires—the task of generalizing these notions in meaningful ways.

DEFINITION 9. Let P be a protocol *with* fault tolerance ϕ. *We* say that P runs in an expected constant number of rounds if there exists a positive constant d *such* that for all numbers of players n, for *all* ϕ*-adversaries* **A**, and for all proper initial

quantities IQ, *the expected number of rounds for a random execution of* $P(n)$ *with* $A(n)$ *on IQ to halt is d.*

In measuring the amount of local computation in an execution of a protocol P with an adversary A, we count only the steps taken by the currently good players. (The adversary attacking the protocol can, of course, compute as much as it wants, but its steps do not contribute to the local computation of the protocol.) Recall that we have defined protocols to be uniform programs. Thus before running a protocol P in an n-size network, the players must first run P on input n so as to compute the exact n-tuple of programs, $P(n)$, that they should execute. (Player i will, in fact, execute the ith component of $P(n)$.) We thus also count as P's local computation the steps necessary for the players of an n-size network to compute $P(n)$.

Following the current tradition, we identify efficiency with polynomial-time computation, and we insist that our polynomial-time bounds hold for any possible adversary attacking the protocol.

DEFINITION 10. *Let* P *be a protocol with fault tolerance* ϕ. *We say that* P runs in (expected) polynomial time *if there exists a polynomial* Q *such that the following hold:*

1. For all sufficiently large n, *the (expected) number of steps for protocol* P *to output* $P(n) = (P_1, \ldots, P_n)$ *on input* n *is less than* $Q(n)$.

2. *For all integers n, for all 4-adversaries* A, *and for all initial quantities* IQ, *if L is the sum of the lengths of the inputs of the players outside the initially bad set, the (expected) number of protocol steps for a random execution of* $P(n)$ *with* A *on* IQ *to halt is less than* $Q(n + L)$.

(Here by "protocol step" we mean any step executed by P_g *for any currently good player g.)* [13]

Notice that to establish the local complexity of a given protocol P, we regard as an input the size n of the network in which P is run. This is indeed necessary because we consider the steps used to compute $P(n)$ as local computation, but it is also reasonable with respect to the rest of P's local computation. Indeed, even for protocols that have no inputs (and thus $L = 0$, as in the case of our protocol OC of section 7), we expect that when they are "really" executed among n players, at least n messages will be sent, which entails that the local computation is at least $O(n)$.[14]

Notice also that although we have not demanded that a protocol set its message bounds to finite values for the purpose of defining its local complexity, the amount of local computation of a protocol P can be small—or just bounded, for that matter—only if P sets proper message bounds.

4. Presentation and organization. We have chosen to build our Byzantine agreement algorithm in a modular way. We first introduce graded broadcast, a simple primitive weakly simulating the capability of broadcasting. We then use this primitive to build another one: graded verifiable secret sharing. Both primitives are of independent interest. Next: we present a technical construction from graded verifiable secret

[13] Note that the notion of polynomial time is convenient in that we should not worry too much about fine tuning the balance between the effort of computing $P(n)$ and that necessary to run the protocol, nor should we worry about whether the polynomial Q should be evaluated on n + L or—say—the maximum between n and L, or n times the maximum length of the inputs of the initially good players. These specific choices would instead be crucial for defining that a protocol runs in a—say—quadratic amount of time. Similarly, a round complexity that is constant (and thus independent of all possible quantities affecting the computation) is "more or less uncontroversially defined;" however, the same cannot be said if the round complexity of a protocol were--say—quadratic.

[14] In any case, in a Byzantine agreement protocol each player has a single-bit input, and thus $L = O(n)$ in a network of size n.

sharing to a special protocol for collectively generating a special coin flip, that is, a bit that is both sufficiently random and sufficiently often visible by all good players. Finally, we show that Byzantine agreement is reducible to this special coin-flipping protocol.

Let us now discuss the additional choices we have made in presenting our protocols.

Proofs. Everything important becomes easy with time, and we believe that this will be the fate of adversarial computation. However, at this stage of its development, it is so easy to make mistakes that we have chosen to expand our proofs more than is legitimate and bearable in a more familiar setting. (Proofs are, after all, social processes and ought only to be convincing to a given set of researchers at a given point in time.) We have, however, consistently broken our proofs up into shorter claims so as to enable the reader to skip what she personally considers obvious.

Steps. As usual, we conceptually organize the computation of our protocols into *steps.* The primary reason for grouping certain instructions in a step is clarity of exposition. As a result, one step may require many rounds to be implemented, while another may require only one round.

In this paper, we adopt the convention of treating each step *as* a *subprotocol* in itself; that is, executing a step composed of certain instructions means calling a protocol consisting of those instructions. In view of our mechanism for subprotocol calling, a consequence of our convention is that each step starts being executed at a. "new" round; that is, a step requires at least one round to be implemented.

The advantage of this convention is that we gain a more immediate correspondence between steps and rounds. For instance, the number of rounds of a protocol simply becomes the sum of the number of rounds of its steps; for another example, in our proofs, it will be quite easy upon encountering the expression "round r" to realize which is its corresponding step.

A (superficial) disadvantage of our convention is that our protocols "seem longer" since one round may be artificially added for each step. In fact, whenever the last round of a given step consists solely of internal computations of the processors, it. can be merged in any practical implementation with the first. round of the following step. This is no great loss, however, since we are not interested in claiming $O(1)$ improvements in the running times of our protocols.

Random selections. As we have seen, by saying that a player i flips a coin, we mean that he reads the next unread bit, of a string R_i. The use of the expression "flips a. coin" is justified by the fact that we will be focusing on random executions of our protocols, in which case, since each bit of R_i is independently and unifornily selected, all coin tosses of i are "genuine" and independent. In describing our protocols, however, we make use of additional suggestive language. By saying that i "randomly selects element e in a set S of cardinality k," we mean that the elements of the set are put in one-to-one correspondence with the integer interval $[0, k-1]$ and that the player i keeps on reading $\lceil \log k \rceil$ corisecutive unread bits from string R_i until the "name" of an element in S is found.[15] Thus when executing an instruction of the type "$\forall y \in$ T randomly select, $e_y \in S$"—where both T and S are finite sets—all of the resulting selections will be *random* (since no portion of R_i is skipped) and *independent* (since

[15] Thus the possibility that an execution *diverges* exists here, though we do not "protect" ourselves against such an event for two reasons. First, handling divergence properly would have translated into much heavier definitions and notations without adding much to the specific content of this paper. Second, we focus on random executions, and the probability of divergence in a random execution is 0.

no overlapping portions of R_i are ever used). This notation holds for adversaries as well.

Hiding message bounds. Only one of our protocols, Gradecast, makes use of message-bound inputs; all others specify their message bounds at each round in their codes. To lighten these codes, however, we omit making the message bounds explicit at any round in which they can be simply computed. For instance, if round $r-1$ consists, for all players, of the instruction

> "if predicate $\mathcal{P}$ is true, then send your name to all players; else send them the empty word ε,"

then for any good player, the message bound for round r is $\lceil \log n \rceil$, the maximum length of a player's name (assuming that the name of a player i is encoded by the binary representation of integer i).[16]

Sending and receiving. When processor j is instructed to send a value v to the processor i, we let v^* denote the value actually received by i since it can be different from v in case j is bad. It may happen that such a value v* must itself be sent to other processors. In this case, we may write v^{**} for $(v^*)^*$.

It is implicitly understood that whenever a message easily recognizable as not being of the proper form is sent to a good processor, this interprets it as ε, the empty string. Any predicate of ε is defined to evaluate to false.

For any string a, we let distribute a denote the instruction of sending σ to every processor.

For every nonempty string a, the expression $tally(\sigma)$, which occurs in round $r+1$ of the code for player i of a given protocol, denotes the number of players that sent a to i in round r. If $\mathcal{P}$ is a predicate, the notation $\mathcal{P}(tally(x))$ is shorthand for "there exists a nonempty string x such that $\mathcal{P}(tally(x))$."

Self and others. Variables internal to processor i will sometimes carry the subscript i to facilitate the comparison of internal variables of different processors.

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. A distinguished processor follows the code for all players in addition to his special code.

Math. We are concerned only with integral intervals. Thus if x and y are integers, the expression [z,y] stands for the set of integers $\{i : x \leq i \leq y\}$.

If S is a set, we let S^2 stand for the Cartesian product of S and itself and 2^S stand for the set of subsets of S.

All logarithms in this paper are in base 2 (but we still use the natural base e for other purposes).

Genders. We will refer to a player as a "he" and an adversary as a "she."[17]

Protocols. To describe a protocol P, we just describe $P(n)$, leaving it to the reader to check that this code can be uniformly generated on input n.

Comments. We interleave the code of our protocols with clearly labeled comments. Often, we label short comments by writing them within quotation marks. In fact, in our protocols, all words within quotation marks are comments, not instructions.

Numberings. Definitions and claims that appear within the proof of a lemma or theorem are not expected to have interest--or even "meaning"--outside of their local

---

[16] Of course, our "English" protocols can be implemented in polynomial time only if a proper encoding is used. For instance, if for sending the name of a player i we chose to send a string consisting of 2^i 1's, some of our English protocols would not have a polynomial-time implementation. However, any "reasonable" encoding would do.

[17] This gender assignment has been made at random. (Moreover, any additional motive is no longer valid.)

5.2. A graded broadcast protocol.

PROTOCOL *Gradecast*(n)
Input for every player i: h, the identity of the sender, and k, the message bound.
Additional input for sender h: a, the sender's message.
1. (for sender h): Distribute σ.
2. (for every player i): Distribute a*.
3. (for every player i): If $tally(z) \geq 2n/3$, distribute z; otherwise, send no messages.
4. (for every player i):
4.1. If $tally(x) \geq 2n/3$, output (x,2) and halt. Else:
4.2. If $tally(x) \geq n/3$, output (x,1) and halt. Else:
4.3. Output $(\varepsilon, 0)$ and halt.

THEOREM 1. Gradecast is a four-round, polynomial-time, graded broadcast protocol with fault tolerance 1/3.

Proof. That protocol Gradecast is four-round and polynomial-time is obvious. Before proving the remaining properties, let us establish the following simple claim.

CLAIM T1-1. In any execution of Gradecast with message bound k, no good player sends a message longer than k.

Proof. A good player may send messages only at steps 1, 2, and 3. If he sends a message at step 1, then he is the dealer, and the only message he sends in this step is his input string a, which is guaranteed to be no longer than k. As for steps 2 and 3, what a good player distributes is a message that he has received at the start of the same step; thus, due to the message-bound mechanism, such a message is at most k-bit long. ■

Let us now show that Gradecast is a graded broadcast protocol with fault tolerance 1/3. First, consider property 1. Let i be a player that is good throughout the entire execution of the protocol, and assume that $grade_i > 0$. Then because of message bounding, there must exist a nonempty string X, whose length is at most k, such that i computes $tally(X) \geq n/3$ in step 4. Thus i must receive X from at least a good player g at the start of round 4. Because X is at most k bits long and because of Claim T1-1, this implies that g had actually distributed X in round 3. In turn, this implies that g had received X as the round-2 message from at least $2n/3$ players. Letting w (w $< n/3$) of these players be bad, because of Claim T1-1, at least $2n/3 - w$ good players had thus distributed X in round 2. Therefore, at most $n/3$ good players could have distributed any value other than X in round 2. Thus for any k-bit string Y, Y $\neq$ X, at most $n/3$ good players and hence $< 2n/3$ players overall could have sent Y to a good player in round 2. Thus no good player may have distributed Y in round 3, nor—because of Claim T1-1—may a good player have distributed a string Y' longer than k whose k-bit prefix coincides with Y. Hence for all good players, $tally(Y) < n/3$ at round 4 (which, in particular, implies that $value_i$ is uniquely determined). Now let j be another player, good until the end of the protocol, whose output has a positive grade component. Since this implies that there exists a string Z, whose length is at most k, such that j has received Z from at least $n/3$ players in step 4, and since we have just proved that Z cannot be different from X, it must be that in step 4 $tally(X) \geq n/3$ also for player j; that is, also $value_j = X$, which proves property 1.

Property 2 follows from the fact that if a good player i sets $grade_i = 2$, then he has received a k-bit string X from at least $2n/3$ players in round 4. Therefore, by Claim T1-1, at least $n/3$ good players distributed X in step 3; thus all good players must have received X at the start of round 4 from at least $n/3$ players, and thus all good players must decide according to 4.1 or 4.2.

Property 3 is easily verified since if h is good, all good players receive and distribute a in rounds 2 and 3. □

Remarks.

• Protocol Gradecast is still a graded broadcast protocol with fault tolerance 1/3 when it is run on a network whose communication lines are not private (i.e., if the adversary can monitor the messages exchanged by the good players).

• If all message bounds were dropped from Gradecast, the resulting protocol would still satisfy properties 1, 2, and 3 of graded broadcast but would no longer be polynomial-time.

Theorem 1 guarantees that certain relationships hold among internal variables of good processors whenever protocol Gradecast is executed with a 113-adversary. These variables, being internal, are not observable by the adversary. The following simple lemma, however, guarantees that the adversary can infer them from her history—actually, from just a portion of her history, something that will be useful much later in this paper.

LEMMA 1. For any given adversary A, any execution of Gradecast with A is computable from *A's* initial history and coin tosses after round 0 if the sender is bad at the start of the protocol, and from A's initial history and coin tosses after round 0 and the sender's message otherwise.

Proof. As for any determinist protocol, an execution of Gradecast with an adversary is solely determined by (1) the inputs of the initially good players and (2) the adversary's history and coin tosses after round 0. Now, for protocol Gradecast, quantity (1) coincides with the sender's message if the sender is initially good, and is empty otherwise. □

The use of protocol Gradecast in our paper is so extensive that it is worth establishing a convenient notation.

Notation. After an execution of Gmdecast in which player i is the sender, we use the following terminology:

• If a player j outputs a pair (v,2), we say that j accepts i's gradecast of v, or accepts v from i. If we do not wish to emphasize the value v, we may simply say that j accepts i's gradecast.

• If j outputs (v,x), for $x \geq 1$, we say that j hears i's gradecast of v, or hears v from i. If we do not want to emphasize the value v, we simply say that j hears i's gradecast.

• If j outputs (v,0) for some value v, we say that j rejects i's gradecast.

In what follows, we shall make extensive use of Gradecast as a subprotocol. It will thus be convenient to specify a call to Gradecast at step z of an n-party protocol in a compact way. In particular, since message bounds are necessary only for guaranteeing the polynomiality of Gradecast, it will be convenient to keep them in the background as much as possible. For instance, if we are guaranteed that, when executing Gradecast at step z of a given protocol, the sender's message is a single bit, we avoid explicitly specifying that Gradecast is called with message bound $k = 1$. More generally:

• If, given the possible choices for string a, k is the least upper bound to the length of a, then

z: (for player i): gradecast σ

means that step z consists of executing *Gradecast*(n) with sender i, sender's message a, and message bound k.

• If, given the possible choices for the strings σ_i, k is the least upper bound to

their length, then

$$z\text{: (for every player i): gradecast } \sigma_i$$

means that step z consists of executing Cradecast concurrently n times, one for each label $i \in [1, n]$, so that in execution i the sender is i, his message is σ_i, and the message bound is k.

• If, given the possible choices for the strings a_x, k is the least upper bound to their length, then

$$z\text{: (for player i): } \forall x \in S, \text{ gradecast } a_x$$

means that step z consists of executing Gradecast concurrently, once for each label $x \in S$, so that in execution x, the sender is i, his message is a_x, and the message bound is k.

• If, given the possible choices for the strings σ_{xi}, k is the least upper bound to their length, then

$$z\text{: (for every player i): } \forall x \in S, \text{ gradecast } \sigma_{xi}$$

means that step z consists of executing Gradecast concurrently, once for each label xi, where $x \in S$ and $i \in [1, n]$, so that in execution xi, the sender is i, his message is σ_{xi}, and the message bound is k.

Any of the above calls can be made dependent on whether a given property $\mathcal{P}$ holds: with the understanding that if $\mathcal{P}$ is not true, then the gradecast still takes place, but the sender's message is the empty string. For instance, if, given the possible choices for the strings a_x, k is the least upper bound to their length, then

$$z\text{: (for player i): if } \mathcal{P}, \forall x \in S, \text{ gradecast } \sigma_x$$

means that step z consists of executing *Gradecast* concurrently, once for each label $x \in S$, so that in execution x, the sender is i, the message bound is k, and the sender's message is a_x if $\mathcal{P}$ evaluates to TRUE (in general, on x and i's current history) and ε otherwise.

Therefore, step z always consists of four rounds. Indeed, though a bit wasteful, the above convention is convenient to keep our protocols and subprotocols fixed-round.[20]

6. Graded verifiable secret sharing. We now need to adapt the earlier and powerful notion of verifiable secret sharing, developed for a different communication model, to the present scenario.

6.1. Verifiable secret sharing and collective coin flipping. The somewhat paradoxical concept of *verifiable* secret sharing (VSS for short) was introduced by Chor et al. [9], who also provided its first cryptographic implementation (tolerating $O(\log n)$ faults). Informally, a VSS protocol consists of two stages. In the first stage, a dealer "secretly commits" to a value of its choice. In the second stage, this value is recovered. The value is secret at the end of stage 1 in the sense that no subset of players of suitably small size can guess it better than at random, even if they exchange all of the information in their possession thus far (which good players never do in the first stage). The value is committed in stage 1 in the sense that a good player can verify that there exists a unique (and unknown) value x such that whenever stage 2

[20] By adopting more complex mechanics for subprotocol calling, we may interpret the above (conditioned) steps differently, and occasionally save rounds and messages.

is performed, with or without the help of the dealer and *no matter what the current or future bad players might do,* all of the good players will recover x. Moreover, this unique but unknown x is the value originally chosen by the dealer.

Verifiable secret sharing has by now found very sophisticated applications, [21] but we will be interested in the simpler, original application of [9]: enabling a group of players, a minority of which may be faulty, to generate a common and random bit. Informally, VSS allows such players to "collectively flip a coin" as follows. Each player privately selects his own random bit and secretly commits to it in stage 1 of a VSS protocol. When all have done so, all of these committed bits are recovered in stage 2 of the corresponding VSS protocol and the common, random bit is set to be the sum modulo 2 of all the decommitted bits.

Since we have already mentioned that the problem of Byzantine agreement is reducible to that of generating a common random bit, the possibility exists of using VSS for reaching Byzantine agreement. Indeed, as we shall see, we will use a special version of VSS (graded VSS) and a much more special version of the above coin-flipping algorithm (oblivious common coin) so as to produce a bit that is "common enough" and "random enough" to reach Byzantine agreement in constant expected time. Why don't we use ordinary VSS to collectively flip a coin in the straightforward way? The reason is simple: we want to use collective coin flipping for reaching fast Byzantine agreement in our point-to-point communication networks, but all implementations of VSS prior to our work either made use of *broadcasting* (an unavailable primitive in our networks!) or Byzantine agreement (which for us is a goal and not a tool!).

6.2. The notion of graded verifiable secret sharing. We now introduce a weaker version of VSS that is more easily obtainable on our networks without broadcasting. We call it *graded verifiable secret sharing* (graded VSS for short). Informally, this is a sequenced protocol with two components: *Graded Share-Verify,* which roughly corresponds to stage 1 of a VSS protocol, and *Graded Recover,* which roughly corresponds to stage 2. To properly define graded VSS, we need the notion of a an event becoming "fixed" at some .point of the execution of a protocol.

DEFINITION 12. *Let X be an event that may occur only after round r in an execution of a protocol P, and let E be an ezecution of P. We say that X is* fixed at round r in E *if* X *occurs in every execution E' coinciding with E up to round r.*

DEFINITION 13. *Let P be a sequenced protocol, P = (Graded Share-Verify, Graded Recover), in which*

- *all players have a common input consisting of the identity of a distinguished processor, the* dealer, *and (the encoding of) a set of integers, called the* candidate-secret set;
- *the dealer has an additional input, called the* secret, *consisting of an element of the candidate-secret set; and*
- *each processor x is instructed to output a value verification, $\in \{0, 1, 2)$ at the end of Graded Share-Verify and an element of the candidate-secret set at the end of Graded Recover (if this latter component is ever executed on the history of the first one).*

We say that P is a graded verifiable secret sharing protocol with fault tolerance c *if the following four properties hold:*

1. Semiunanimity. *For all initial quantities (IQ), for all c-adversaries A, and for all executions of Graded Share-Verify with A on IQ, if a good player i outputs*

[21] For instance? since [26], it has become the crucial subroutine of all subsequent completeness theorems for protocols with honest majority, most notably those in [2], [3], [10], [21], and [35].

verification$_i$ = 2, then verification$_j$ > 0 for all good players j.

2. Acceptance of good secrets. *For all* IQ *c-adversaries A and for all executions of Graded Share-Verify with A on IQ, if the dealer is always good, then verification$_i$ = 2 for all good players i.*

3. Verifiability. *For all on IQ c-adversaries A and for all executions E of Graded Share-Verify with A on IQ, if verification$_i$ > 0 for a good player i, then there exists a value a in, the candidate-secret set such that the event that all good players output a when executing Graded Recover (on their histories in E) is fixed at the end of E. Moreover, if the dealer is always good in E, a = the secret.*

4. Unpredictability. *For all c-adversaries A, for all players h, for all integer m, and for all cardinality-m set S, if*

- *s is randomly chosen in S,*
- *Graded Share-Verify is randomly executed with A, dealer h, candidate-secret set S, and secret s, and*
- *dealer h is good throughout this execution, and the adversary outputs a value a ∈ S (as her "guess" for the secret) at its end,*

then Prob$(a = s) = 1/m$.

Here the probability is taken not only over the coin tosses of P and A but also over the choice of s. [22]

Remarks.

- Notice that simply saying—in the verifiability condition—"all good players output a in *Graded Recover*" is not sufficient for our purposes. In fact, although the adversary cannot prevent the good players from outputting the same value, this formulation still allows her to decide what the value of *a* should be while executing *Graded Recover*. (An example of this has been constructed by the second author.) Thus *Graded Share-Verify* would not model a secret commitment as discussed above. For this we need the value of a to be *fixed* at the end of *Graded Share-Verify* (when *a* coincides with the dealer's secret and is totally unpredictable to the adversary if the dealer is currently good).
- A definition of VSS can be obtained from the above definition of graded VSS by replacing throughout "*verification$_i$ = 2*" by "*verification$_i$ > 0*." (The definition of VSS obtained in this way is actually, in our opinion, the most general and satisfactory one in the literature to date.) Similarly, jumping ahead, from our protocol *Graded-VSS*, one can easily derive a verifiable secret sharing protocol with broadcasting by essentially replacing all gradecast instructions with broadcast instructions. (It is the transformation of a verifiable secret sharing protocol with broadcasting to a graded

[22] An equivalent formulation of Unpredictability that does not require that the secret be chosen at random in the candidate secret set can be informally described as follows.

Let $PS(A, h, H_A^0, H_{-\{h\}}^0, S, s)$ denote the probability space over the final histories of *A* obtained by first randomly executing *Graded Share-Verify* with adversary *A*, dealer *h*, initial adversarial history H_A^0, initial histories (in a suitable encoding, of all initially good players other than dealer *h*) $H_{-\{h\}}^0$, candidate-secret set S, and secret *s* and then outputting the final adversarial history if dealer h has not been corrupted. Then unpredictability can be reformulated as follows:

4'. For all c-adversaries A, Vh, $\forall H_A^0$, $\forall H_{-\{h\}}$, $\forall S$, and $\forall s_1, s_2 \in S$,

$$PS(A, h, H_A^0, H_{-\{h\}}^0, S, s_1) = PS(A, h, H_A^0, H_{-\{h\}}^0, S, s_2).$$

(The reason for including the histories of all players except the dealer is that we want to maintain unpredictability even when *Graded Share-Verify* is called as a subprotocol. In which case, though the prior histories of the players do not affect the execution of *Graded Share-Verify*, they will appear—for the corrupted players—in the final history of *A.)*

Personally, we find the above formulation (after properly "cleaning it up") generally preferable, but the one in the main text is in a more convenient form for the purposes of this paper.

VSS protocol without broadcasting that proves to be trickier.)

• Let P be a graded verifiable secret sharing protocol with fault tolerance c. P = (*GSV*, GR). Then if there are too few players (i.e., if $\lfloor n \cdot c \rfloor = 0$), even a single player (over than the dealer) may at the end of an execution of GSV possess sufficient information to predict with probability 1 the dealer's secret. However, this does not contradict Unpredictability. Indeed, this property demands that no adversary can predict a good dealer's secret better than at. random, and whenever $\lfloor n \cdot c \rfloor = 0$, she cannot corrupt. any player. (The reader who perceives this phenomenon as awkward may prefer to define graded verifiable secret sharing protocols only when there are sufficiently many players. Personally, we prefer to define protocols so that any number of players greater than 1 is admissible, and we find it awkward to make exceptions for c-fault-tolerant protocols.)

6.3. A graded verifiable secret sharing protocol. This subsection is devoted to constructing the first graded VSS protocol. The basis of our construction was provided by an ingenious VSS protocol developed by Ben-Or, Goldwasser, and Wigderson [3]. Their protocol runs in $O(n)$ rounds—when there may be $O(n)$ faults—in a special type of communication network: the standard-plus-broadcast network. This is a network in which not only each pair of users, communicate via their own private line, but all processors also share a broadcast channel.[23] We have adapted their protocol to our needs in two phases:

1. First, we have improved their result by providing a VSS protocol for standard-plus-broadcast networks, Fast VSS, that (1) runs in a constant number of rounds and (2) is conceptually simpler.[24] (Ben-Or, Goldwasser, and Wigderson have told us that they have independently found a constant-round version of their result, but it is more complicated than ours.)

2. Second, we have transformed Fast VSS (a protocol for standard-plus-broadcast networks) into *Graded VSS*, a constant-round graded VSS rotocol for *standard* networks (i.e., without any broadcasting facilities).

For the sake of conciseness, since the focus of this paper is on stand rd networks, we forgo providing an explicit description of Fast VSS. (Below we present just the basic intuition behind it since this can effectively be used for *Graded VSS* as well.) Indeed, in our protocol *Graded* VSS, we have merged the above two steps into a single one. (The reader can, however, easily reconstruct the code of Fast VSS from that of GradedVSS.) We will, however, provide separate intuition for each of the above two phases.

Phase 1: Fast VSS. In Fast *VSS*, the dealer encodes his own secret in a special and redundant way. Namely, if the adversary can corrupt at most t players, the dealer selects a bivariate polynomial $f(x, y)$ of degree t in each variable such that $f(0.0)$ equals his secret. He then privately gives to player i the polynomials $P_i(y) = f(i, y)$ and $Q_i(x) = f(x, i)$ as his shares of the secret, an x-share and a y-share. As we shall see, the shares of any $\leq$ t players do not betray the secret at. all. On the other hand, as expressed by the following lemma, any $t + 1$ genuine z-shares determine the secret (and the same is true for the y-shares).

[23] Actually, they share a bit more powerful means of communication: each recipient of a message m traveling along this special shared channel is guaranteed not only that all processors receive the same string m that he does but. also that all processors know who the sender of m is.

[24] The protocol of [3] made use of Reed–Solomon codes. while our *Fast VSS* does not rely on any error-correcting codes—or at least it succeeds in hiding them awav while remaining self-contained in a very simple manner.

a

Our choice of encoding for the dealer's secret does not guarantee verifiability per se. In fact, a good player cannot check whether his received x-share is genuine or—say—a random polynomial of degree t. It is here that the y-shares come into play. In fact, *FastVSS* performs several checks centered around the following simple property: if two players i and j both hold genuine shares, then it should be that $P_i(j) = Q_j(i)$.

Unpredictability is guaranteed since *FastVSS* is constructed so that, in every check, the information about the secret of a good dealer obtainable by the adversary can be computed from the shares in her possession—which we have already claimed to be insufficient to predict the dealer's secret.

Phase 2: From VSS to graded VSS. Our transformation of *FastVSS* into *GradedVSS* possesses a somewhat general flavor: it appears to provide a compiler-type algorithm that, on input any *known* VSS protocol for standard-plus-broadcast networks, outputs a graded VSS protocol running in a standard network, with the same fault tolerance of the input protocol and with essentially the same time and number of rounds.[25] This simple transformation is thus potentially useful: one may be able to turn more efficient VSS protocols developed in the future into more efficient graded VSS protocols.

Quite intuitively, the essence of our transformation consists of replacing the broadcast instructions of the input VSS protocol by gradecast instructions and then of properly branching on the grade produced by each gradecast. As the expression "properly" indicates, however, some care is needed in deciding how to branch. Though some degree of freedom is available, it is crucial to exploit the fact that grades are *3-valued.* It should be noticed that after a gradecast instruction of *GradedVSS*, we sometimes branch based on whether the gradecast is accepted or not (i.e., on whether the resulting grade is **2** or less than 2) and other times based on whether the gradecast is heard or not (i.e., on whether the resulting grade is ≥ 1 or equal to 0). Now, although some of these "accepted-or-not" branchings could be replaced by "heard-or-not" branchings (and vice versa), it can be shown that adopting only a single type of branching does not work. Carrying VSS from standard-plus-broadcast networks to standard ones without losing too much meaning is indeed the very reason that we have introduced our 3-valued graded broadcasting primitive.

(If going from VSS protocols to graded VSS protocols requires a minimum of attention, the "reverse" transformation is instead quite straightforward. Protocol *FastVSS* is in fact immediately obtainable from protocol *GradedVSS.)*

Before presenting protocol *GradedVSS*, let us state and prove a variant of the classic Lagrange interpolation theorem.

LEMMA **2.** *Let p be a prime, t be a nonnegative integer, $x_1, \ldots, x_{t+1}$ be distinct elements in Z_p, and $Q_1(y), \ldots, Q_{t+1}(y)$ be polynomials* mod p *of degree t. Then there exists a unique bivariate polynomial $F(x,y)$ of degree t (in each of the variables x and y) such that*

$$(*) \qquad F(x_i, y) = Q_i(y) \quad \text{for } i = 1, \ldots, t+1.$$

Proof. Define (Lagrange interpolation)

$$F(x,y) = \sum_{i=1}^{t+1} Q_i(y) \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)}.$$

[25] While our transformation works for all known VSS protocols, it is still conceivable that it cannot be applied to some future "bizarre" one.

Then $F(x,y)$ has degree t and satisfies $(*)$. We now argue that this polynomial is unique. Now assume that there exist two different t-degree bivariate polynomials $F_1(x,y)$ and $F_2(x,y)$ that satisfy $(*)$. We will prove that the polynomial

$$R(x,y) = F_1(x,y) - F_2(x,y) = \sum_{i,j} r_{ij} x^i y^j$$

is identically 0. For each $k = 1,\ldots,t+1$, we have

$$\sum_{i,j=0}^{t} r_{ij} x_k^i y^i = R(x_k,y) \stackrel{\text{def.}}{=} F_1(x_k,y) - F_2(x_k,y) \stackrel{(*)}{=} Q_k(y) - Q_k(y) = 0.$$

That is, for each $k = 1,\ldots,t+1$, the polynomial in y $\sum_{j=0}^{t}(\sum_{i=0}^{t} r_{ij} x_k^i) y^j$ is identically 0. Thus for each fixed $\bar{j}$, $\sum_{i=0}^{t} r_{i\bar{j}} x_k^i = 0$ for $k = 1,\ldots,t+1$, that is, the polynomial $\sum_{i=0}^{t} r_{i\bar{j}} x^i$ evaluates to 0 at the $t+1$ points $x_1,\ldots,x_{t+1}$. This implies that $r_{i\bar{j}} = 0$ for all $i = 1,\ldots,t+1$. Thus $R(x,y)$ is identically 0, which proves the uniqueness of $F(x,y)$. □

Notation for protocol *GradedSV*. In most of the scientific literature, the upper bound on the number of corruptable processors, denoted by t, is integral and explicitly given as an input to a fault-tolerant protocol. Having t as an input to each protocol would, however, be a bit cumbersome in our case: we have quite a few subprotocol calls and thus we would need to continuously specify the value of t for each call. Also, we are primarily interested in the highest possible value of t (i.e., $t = \lfloor (n-1)/3 \rfloor$) and our protocols—with the singular exception of *GradedVSS*—do not become more efficient for smaller values of t. However, to allow the reader to appreciate how the efficiency of Graded VSS decreases with t, we set $t = \lfloor (n-1)/3 \rfloor$ at its start (rather than making t an input to the protocol).

THEOREM 2. *GradedVSS* is a graded *verifiable* secret sharing protocol with fault tolerance 1/3 that runs in expected *polynomial time*; GradedSV is a 25-round protocol, and *GradedR* is a two-round protocol.

Proof. The claims about the number of rounds of GradedSV and *GradedR* are trivially verified. Equally simple to verify is the claim about the running time. (Recall that our choice of notation allows us to hide our message bounds.) The only difficulty that perhaps arises is about the computation of the prime p in step 1 of *GradedSV*(n). Actually, this prime can be found in deterministic polynomial time. In fact, for all sufficiently large k, there is a prime in the interval $[k, 2k]$. Thus, letting k be the maximum between n and m, we can in poly(k) time (and thus in time polynomial in n plus the total length of our inputs since $h < n$ and m is presented in unary) consider all integers in [k,2k] in increasing order until one is found that is proved prime by exaustive search of its divisors.

Let us now address the other claims.

Semiunanimify. If any good player G outputs $\textit{verification}_G = 2$, he has received recoverable from at least $2t+1$ players, of which at least $t+1$ are good. Thus every other good player g has received recoverable from at least $t+1$ players, so he outputs $\textit{verification}_g \geq 1$.

Acceptance of good secrets. Let us first show that if the dealer is good (throughout GradedSV), then no good player gradecasts *badshare* in step 6. Since in this step a good player G can gradecast *badshare* only in three cases, let us show that none of them can occur.

Case 1. Assume that G has accepted the gradecast of *disagree*(j) from a player k in step 4. Then because of property 2 of any gradecast protocol, the dealer has heard

PROTOCOL *GradedSV(n)*

Input for every player i: h. the identity of the dealer, and m. a unary string encoding the candidate-secret interval $[0, m-1]$.

1. (for every player i): Compute p, the smallest prime greater than n and m, and set t = $\lfloor (n-1)/3 \rfloor$.

   Comment. All computations are done modulo p.

3. (for dealer h): Randomly select a t-degree bivariate polynomial f(x,y) such that $f(0,0) = s$. "In other words, set $a_{00} = s$, for all (i.j) $\in [1,t]^2 - \{(0,0)\}$, randomly select a_{ij} in [0,p − 1], and set f(x.y) = $\sum_{i,j} a_{ij} x^i y^j$." For all i, privately send (P_i, Q_i) to player i, where $P_i = P_i(y) = f$(i,y) and $Q_i = Q_i(x) = f(x,i)$.

   Comment. $f(0,0) = s$, your secret. $P_i(j) = Q_j(i)$ for all i and j.

3. (for every player i): For all j, if the dealer has not sent you a pair of t-degree polynomials mod p, send ε to player j; else, privately send j the value $Q_i^*(j)$.

4. (for every player i): For all j, if $P_i^*(j) \# (Q_j^*(i))^*$, gradecast disagree (j).

   Comment. Either j or the dealer is bad or both are bad.

5. (for dealer h): For all (i,j) $\in [1,n]^2$, if you heard disagree (j) from player i in step 4, gradecast $(i, j, Q_j(i))$.

   Comment. i or j is bad or both are bad: what you reveal is already known to the adversary.

6. (for every player i): For all (k,j) $\in [1,n]^2$, if you accepted disagree (j) from player k in step 4 and
   - in the previous step you did not accept from the dealer exactly one value of the form (k, j, V), where $V \in [0, p-1]$; or
   - you accepted such a value and k = i, "i.e., you are player k," but $V \# P_i^*(j)$; or
   - you accepted such a value and j = i, "i.e., you are player j," but V ≠ Qf(k),

   gradecast badshare. "The dealer is bad."

7. (for dealer h): For all i, if you heard *badshare* from player i in step 6, gradecast $(i, P_i(y), Q_i(x))$.

   Comment. i is bad: what you reveal is already known to the adversary.

8. (for every player i): If
   (a) you gradecasted *badshare* in step 6: or
   (b) you accepted *badshare* from more than t players in step 6; or
   (c) for each player j whose gradecast of *badshare* in step 6 you have accepted, a step ago you did not accept from the dealer the gradecast of a value (j, U, V), where U and V are t-degree polynomials, or you accepted such a value but Qf(j) ≠ $U(i)$ or P_i^*(j) ≠ $V(i)$,

   distribute *badshare*. "The dealer is bad."

9. (for each player i): If $tally(badshare) \leq t$, distribute recoverable.

10. (for each player i):

    If $tally(recoverable) > 2t$, output verification, = 2.

    Comment. The secret is recoverable and all good players know it.

    Else: If $tally(recoverable) >$ t, output $verification_i = 1$.

    Comment. You know that the secret is recoverable, but other good players may not know it.

    Else: Output $verification_i = 0$.

    Comment. The secret may or may not be recoverable.

PROTOCOL *GradedR*(n)

1. (for every player i): Distribute P_i^* and Q_i^*.
2. (for every player i):

For each player j, set $P_j^i(y) = P_j^{**}$ and $Q_j^i(y) = Q_j^{**}$. If you have accepted *badshare* from j in step 6 of GradedSV and you have heard $(j, U(y), V(x))$ from the dealer in step 7, reset $P_j^i(y) = U(y)$ and $Q_j^i(x) = V(x)$.

Comment. $P_j^i(y)$ and $Q_j^i(z)$ are your own view of player j's *final* shares.

Let $count_i(j)$ consist of the number of players k for which $P_j^i(k) = Q_k^i(j)$.

Comment. If a good player has output verification > 0 in GradedSV, all good players are given count $> 2t + 1$. However, a bad player may be given count $> 2t + 1$ by some good player and a low count by another good player.

If possible, select a set of $t + 1$ players k such that $count_i(k) \geq 2t + 1$. Let $k_1, \ldots, k_{t+1}$ be the members of this set.

Comment. If a good player has output verification > 0 in GradedSV, there will be such a set. In this case, although different good players may select different sets and some of these sets may contain bad players, each set determines the same bivariate polynomial

Compute the unique bivariate polynomial $\mathcal{P}(x, y)$ such that $\mathcal{P}(k_j, y) = P_{k_j}^i(y)$ $j \in [1, t + 1]$ and output $\mathcal{P}(0, 0)$ mod m as the dealer's secret.

k's gradecast and has thus responded in step 5 by gradecasting $(k, j, Q_j(k))$. Since the dealer's gradecast is proper, it is necessarily accepted by G.

Case 2. If G gradecasts *disagree*(j) in step 4, the dealer accepts this proper gradecast and thus properly responds by gradecasting $(G, j, Q_j(G))$, and $Q_j(G)$ coincides with G's x-share, $P_G^*(y)$, evaluated at point j since for a good dealer $P_G^*(j) = P_G(j) = f(G, j) = Q_j(G)$.

Case 3. If G has accepted *disagree*(G) from k in step 4, the dealer has at least heard this value and has thus responded by properly gradecasting $(k, G, Q_G(k))$. This value is thus accepted by G; moreover, since the dealer is good, $Q_G^*(x) = Q_G(x)$ and thus $Q_G^*(k) = Q_G(k)$. Thus if the dealer is good, in no case does a good player G gradecast *badshare* in step 6. This implies that no good player can distribute *badshare* in step 8 according to conditions 8(a) or 8(b). Moreover, as long as the dealer continues to be good in step 7, a good player G cannot distribute *badshare* because of 8(c) as well. In fact, if G has accepted *badshare* from j in step 6, the dealer has at least heard this value and responded by properly gradecasting the polynomials $P_j(y)$ and $Q_j(x)$; since G accepts all proper gradecasts, and because when the dealer is good $P_j^*(i) = P_j(i) = Q_i(j) = Q_i^*(j)$ for all i and j, all of G's checks in steps 8 will be passed. We conclude that if the dealer is good in GradedSV, only the bad players may distribute *badshare* in step 8. Thus, since they are at most t in number, all good players G will distribute *recoverable* in Step 9 and output $verification_G = 2$ in step 10.

Verifiability. Let S be an execution of GradedSV in which at most $t < n/3$ players are corrupted and a good player outputs *verification* > 0, and let R be an execution of GradedR on the histories of S. We need to show that (1) there exists a value σ_S such that the event that all good players output σ_S in $\mathcal{R}$ is fixed at the end of S and (2) σ_S coincides with the dealer's secret if he is always good in S.

To this end. let us establish a convenient notation and a sequence of simple claims relative to $\mathcal{S}$ and R. Recall that, since (*GradedVSS*, GradedR) is a sequenced protocol,

any player good in R (actually, in $\mathcal{R}$'s first round) is always good in S.

LOCAL DEFINITION. In an execution of Graded VSS, a player is said to be satisfied *if* he is good throughout the execution and does not distribute *badshare* in step 8.

CLAIM T2-0. In S, there is a set *of* t + 1 satisfied players.

Proof. The proof is by contradiction. Were our claim false, then since there are at least 2t + 1 good players in $\mathcal{S}$, at least t + 1 of them would have distributed *badshare* in step 8. Thus no good player would have distributed recoverable in step 9, and no good player would have output verification> 0 in step 10. ■

Due to condition 8(a), we also know that a satisfied player does not gradecast *badshare* in step 6 either; thus for all satisfied players i and j, $P_i^*(j) = Q_j^*(\mathrm{i})$. In view of Claim T2-0 and Lemma 2, we can thus present the following (local) definition.

LOCAL DEFINITION. In execution S, we let $\mathcal{G}_\mathcal{S}$ denote the lexicographically first *set* of t + 1 satisfied *players,*[26] and we let $\mathcal{F}_\mathcal{S}$ denote the unique, *bivariate,* t-degree *polynomial* associated with $\mathcal{G}_\mathcal{S}$ by Lemma 2: that is, $\mathcal{F}(i, \mathrm{y}) = P_i^*(y)\ \forall i \in \mathcal{G}_\mathcal{S}$.

CLAIM T2-1. $\forall i \in \mathcal{G}_\mathcal{S}$, $P_i^*(y) = \mathcal{F}_\mathcal{S}(i, \mathrm{y})$ and $Q_i^*(x) = \mathcal{F}_\mathcal{S}(x, i)$.

Proof. Clearly, $P_i^*(y) = \mathcal{F}_\mathcal{S}(i, \mathrm{y})\ \forall i \in \mathcal{G}_\mathcal{S}$ by construction. We now prove the second set of equalities. Let i $\in \mathcal{G}_\mathcal{S}$; since we are dealing with polynomials of degree t, it is enough to prove that $\mathcal{F}_\mathcal{S}(x, \mathrm{i})$ equals $Q_i^*(\mathrm{x})$ at t + 1 points. Indeed, for all j $\in \mathcal{G}_\mathcal{S}$, we have

$$\mathcal{F}_\mathcal{S}(j, i) \overset{\text{by construction}}{=} P_j^*(i) = Q_i^*(j),$$

where the last equality has been checked to hold by satisfied player j in step 4 since good player i sent him $Q_i^*(\mathrm{j})$ in step 3. ■

CLAIM T2-2. Let G and i be good in $\mathcal{R}$, and let i also be satisfied in S. Then $P_i^* = P_i^G$ and $Q_i^* = Q_i^G$.

Proof. Informally, we must show that G's own view of i's final shares coincides with that of i himself. Being good in $\mathcal{R}$, i sends P_i^* and Q_i^* to G in step 1 of $\mathcal{R}$. Thus G sets $P_i^G = P_i^*$ and $Q_i^G = Q_i^*$ at the beginning of step 2 of R. Finally, G does not reset these variables in the remainder of step 2. Indeed, he may reset these variables only under certain conditions, which include having accepted *badshare* from i in step 6 of *GradedSV;* however, no satisfied player gradecasts *badshare* in step 6. ■

CLAIM T2-3. Let G and g be good in R. Then $P_g^G(y) = \mathcal{F}_\mathcal{S}(g, \mathrm{y})$ and $Q_g^G(x) = \mathcal{F}_\mathcal{S}(x, g)$.

Proof. We prove only the first equality since the second one is proved similarly. To this end, it is enough to show that $\mathcal{F}_\mathcal{S}(g, \mathrm{y})$ and $P_g^G(y)$ agree on every i $\in \mathcal{G}_\mathcal{S}$. Indeed, by Claim T2-3, we have

$$\forall i \in \mathcal{G}_\mathcal{S}, \quad \mathcal{F}_\mathcal{S}(g, i) = Q_i^*(g).$$

We now prove that

$$\forall i \in \mathcal{G}_\mathcal{S}, \quad Q_i^*(g) = P_g^G(i).$$

We break the proof of the above statement into two cases:

(a) G does not reset $P_g^G(y)$ (i.e., $P_g^G(y)$ coincides with the polynomial in y that g sent to G in step 1 of R);

(b) G resets $P_g^G(y)$ (i.e., $P_g^G(y)$ is the polynomial in y gradecasted by the dealer in step 7 of S).

[26] Any uniquely specified set of t + 1 satisfied players in S would do.

Case (a). In this case. $P_g^G(y) = P_g^*(y)$. We must now argue that if $i \in \mathcal{G}_S$, then $Q_i^*(g) = P_g^*(i)$. To begin with, notice that since i privately sent value Qf (g) to g in step 3 of $\mathcal{S}$, player g could compare these two values. We now show that $P_g^*(\mathrm{i}) \neq Q_i^*(\mathrm{g})$ leads to a contradiction. In fact, if the latter inequality holds, g gradecasts *disagree*(i) in step 4 of S. Since i accepts this gradecast, to remain satisfied he must accept $(g, \mathrm{i}, Q_i^*(\mathrm{g}))$ from the dealer in step 5 of S. This causes player g to gradecast *badshare* in step 6 of S—either because he does not accept the dealer's answer or because he accepts exactly the same answer that i does by property 1 of gradecast and thus we still have $Q_i^*(g) \neq P_g^*(i)$. Since i must accept g's gradecast of badshare, to keep him satisfied, the dealer must reply by gradecasting g's x-share and y-share in step 7 of S in order to have these values accepted by i. Thus these shares are at least heard by G, who thus resets P_g^G and Q_g^G. This contradicts the assumption that we are in Case (a).

Case (b). In this case, G must have first accepted the gradecast of *badshare* from g in step 6. Since g is good, this gradecast must have been accepted by good player i as well. Since i is satisfied, he must also have accepted the value replied by the dealer, $(\mathrm{g}, U(y), V(x))$. By property 1 of gradecast, U and V are the same values heard by G and are thus G's own view of g's final shares; that is, $U(y) = P_g^G(y)$ and $V(x) = Q_g^G(x)$. The reason that now Qf(g) $= P_g^G(i)$ is that i satisfactorily checked in step 8 that $Q_i^*(\mathrm{g}) = U(\mathrm{i})$. ■

CLAIM T2-4. For all G good in R and *for* all k, $count_G(k) \geq 2\mathrm{t} + 1 \Rightarrow P_k^G(y) = \mathcal{F}_S(k, y)$.

Proof. Since $P_k^G(y)$ and $\mathcal{F}_S$(k,y) are t-degree univariate polynomials, it is sufficient'to show that they agree on t + 1 points. Indeed, if $count_G(k) \geq 2\mathrm{t} + 1$ and the bad players are at most t, there must exist t + 1 good players g such that

$$P_k^G(g) \overset{\text{def. of } count_G}{=} Q_g^G(k) \overset{\text{Claim T2-3}}{=} \mathcal{F}_S(k, g). \quad \blacksquare$$

We are now ready to prove that in step 2 of $\mathcal{R}$, every good player G computes the bivariate polynomial $\mathcal{F}_S$ and thus outputs $\mathcal{F}_S(0,0) \bmod m$. To this end, first notice that there will be at least t + 1 players k such that $count_G(k) \geq 2\mathrm{t} + 1$. In fact, for all good players g_1 and g_2, we have

$$P_{g_1}^G(g_2) \overset{\text{Claim T2-3}}{=} \mathcal{F}_S(g_1, g_2) \overset{\text{Claim T2-3}}{=} Q_{g_2}^G(g_1).$$

Thus $count_G(g_1)$ will he at least 2t + 1, that is, at least the number of good players g_2; since there are at least 2t + 1 such players g_1, it will be possible for G to select t + 1 players for which $count_G \geq 2\mathrm{t} + 1$. Let. $k_1, \ldots, k_{t+1}$ be the ones he actually selects—some of them possibly bad. Then G outputs the polynomial $\mathcal{P}(x, \mathrm{y})$ such that

$$\forall l \in [1, \ldots, t+1], \quad \mathcal{P}(k_l, y) = P_{k_l}^G(y) \overset{\text{Claim T2-4}}{=} \mathcal{F}_S(k_l, y).$$

Thus by Lemma 2, $\mathcal{P}$ and $\mathcal{F}_S$ must be equal, and all good players output $\mathcal{F}_S(0,0) \bmod m$ at the end of R. Because R was just any execution of *GradedR* on the histories of S, because $\mathcal{F}_S$ is determined by execution S. and because $\mathcal{F}_S(0,0) \bmod m$ is guaranteed to belong to the candidate-secret interval $[0, m-1]$, this shows that the event that, all good players in an execution of *GradedR* on the histories of S output $\sigma_S = \mathcal{F}_S(0,0) \bmod m$ is fixed at the end of S.

Let us now show that if the dealer is good throughout S. $\mathcal{F}_S$ actually coincides with the polynomial $f(x, \mathrm{y})$ originally selected by the dealer in step 2 of S. By

Lemma 2, it is enough to prove that there are t + 1 distinct values v such that $\mathcal{F}_S(v, y) = f(\mathrm{v}.y)$. This is our case. In fact, letting G be a fixed good player, for any good player g, we have

$$\mathcal{F}_S(g, y) \overset{\text{Claim T2-3}}{=} P_g^G(y) \overset{\text{Claim T2-2}}{=} P_g^*(y) \overset{\text{good dealer}}{=} P_g(y) \overset{\text{by def.}}{=} f(g, y).$$

Because a good dealer chooses f (x,*y)* so that $f(0,0)$ coincides with his secret, which belongs to the candidate-secret interval $[0, \mathrm{m} - 1]$, whenever the dealer is good, the value output by the good players in $\mathcal{R}$, $\sigma_S = \mathcal{F}_S(0,0)$ mod m, fixed at the end of S, coincides with the dealer's secret.

Because S was just any execution of GradedSV in which $verification_i > 0$ for some good player i, this completes the proof that Verifiability holds for *GradedVSS*.

Unpredictubility. An appealing, rigorous, and general proof of Unpredictability is obtainable utilizing the notion of "secure (or zero-knowledge) computation" [30]. As we have remarked, however, such a notion has not yet been published, and it is too difficult to be quickly summarized here. Therefore. we shall use an ad hoc and "quick-and-dirty" argument.

Our proof consists of showing that, in an execution of GradedSV in which the dealer is never corrupted. there exists a special piece of information (a string), independent of the secret, from which the adversary's history can be deterministically computed. Because the adversary cannot but predict the secret on the basis of her history (and of her coin tosses, which are clearly independent of the secret), this proves that the adversary cannot predict the secret better than at random.

The existence of such a special piece of information follows in part from the general mechanics of (any) protocol execution, and in part from the specific characteristics of our *GradedSV* protocol. We find it useful to present separately the following two parts of our argument: we present the first part in claims T2-5 and T2-6 and the second part in claim T2-8. Let us first establish some convenient notation.

Local definitions. Let P be a protocol, *A* be an adversary, E be an execution of P with A, and r be a round in E. Then:

- we say that a player is eventually bad in E if he is corrupted at some point of it and always good otherwise.
- we denote by $M_r^{AG \to EB}$ the set of messages (labeled with their senders and receivers) sent by the always good players to the eventually bad ones in round r.
- We refer to the quantities
  1. BAD^r
  2. the round-r history of A,
  3. the round-r histories of the eventually bad players.
  4. the coin tosses of A after round r, and
  5. the coin tosses of the eventually bad players after round r

  as the final quantities of round r. (Thus, the final quantities of round 0 include A's initial history.) If x is a step of P and r is x's last round, we refer to round r's final quantities as the *final* quantities of step x.

CLAIM T2-5. Given a protocol P and an adversary A in an execution of P with A, the final quantities of a round r (r > 0) are computable from the final quantities of round $r - 1$ and $M_r^{AG \to EB}$.

Proof. The proof consists of recalling Definition 4 (i.e., how a protocol is executed with an adversary) and verifying that one can execute instructions 0–4 of Definition 4 to the extent necessary to compute the desired final quantities.

(In essence, A's history and coin tosses after each point of the execution of round r are computable given A's history and coin tosses after round $r-1$ (available by hypothesis), the history of each newly corrupted processor at the end of round $r-1$ (also available by hypothesis), and the message that each currently good processor g wishes to send to each currently bad processor b in round r. Now, if g is always good, such a message is among the available inputs, and if g is eventually bad, it can be computed by running P_g on g's history and g's future coin tosses at the end of round $r-1$, both of which belong to the available inputs. The history and coin tosses of each eventually bad player i after round r can be computed from the corresponding and available quantities after round $r-1$ by running P,.) ■

Repeated application of Claim T2-5 immediately yields the following claim.

CLAIM T2-6. Given a protocol P and an adversary **A,** in an execution of P with A, A's final history is computable from the final quantities of round 0 and $\{M_r^{AG\to EB} : r = 1, 2, \ldots\}$.

Properties stronger than those of Claim T2-6 hold for our Gradecast and *GradedSV* protocols. Indeed. Lemma 1 implies that the final quantities of round 0 and only $M_0^{AG\to EB}$ suffice for reconstructing an entire execution of Gradecast. (Notice that in fact, in an execution of Gradecast, $M_0^{AG\to EB}$ coincides with the sender's message whenever the sender is always good.) As we show below for protocol GradedSV, the final quantities of round 0 and $M_1^{AG\to EB}$ suffice for reconstructing the final history of the adversary. (Notice that in an execution of GradedSV, $M_1^{AG\to EB}$ coincides with the x- and y-shares of the eventually bad players whenever the dealer is always good.)

CLAIM T2-7. Given an adversary **A,** *in* an execution *of* GradedSV with A in which the dealer is good, A's final history is computable from the *final* quantities of round *0* and the x- and y-shares of the eventually bad players.

Proof. Let us show how we compute (one by one) the final quantities of each step of *GradedSV* from the quantities available by hypothesis. For the reader's convenience, we recall (emphasizing it and omitting our comments) each step of GradedSV.

1. (For every player i): Compute p, the smallest prime greater than n and m, and set $t = \lfloor (n-1)/3 \rfloor$.

This step consists of a single round in which no good player sends any message (i.e., denoting by r the single round of this step, $M_r^{AG\to EB}$ is empty). Thus, as per Claim T2-5, we compute the final quantities of step 1 from the final quantities of round 0 alone.

2. (For dealer h): Randomly select a degree-t bivariate polynomial $f(x,y)$ such that $f(0,0) = s$. For all i, privately send (P_i, Q_i) to player i, where $P_i = P_i(y) = f(i,y)$ *and* $Q_i = Q_i(x) = f(x,i)$.

This step consists of a single round. Denoting it by r, $M_r^{AG\to EB}$ coincides with the x- and y-shares of the eventually bad players (which are available by hypothesis). Thus, as per Claim T2-5, we can compute the final quantities of step 2 from the final quantities of step 1 and $M_r^{AG\to EB}$.

3. (For every *player* i): For all j, if the dealer *has* not *sent* you a *pair* of t-degree *polynomials mod* p, *send* ε to player j; else, privately send j the value $Q_i^*(j)$.

This step also consists of a single round. Denote it by r, and let i be an always good player and j be an eventually bad one. Because the dealer is good, the message sent by i to j is not ε but $Q_i^*(j) = Q_i(j)$. Although we do not know Q_i, we compute $Q_i^*(j)$ by evaluating polynomial P_j (which is available as the x-share of player j) at point i. Doing so for each always good player and each eventually bad one, we compute the entire $M_r^{AG\to EB}$. We compute the final quantities of step 3 from $M_r^{AG\to EB}$ and the final quantities of step 2. as per Claim T2-5.

4. (For every player i): *For all* j, *if* $P_i^*(j) \neq (Q_j^*(i))^*$, *yradecast* disagree(j).

Step 4 consists of four rounds in which n^2 simultaneous executions of *Gradecast*, properly labeled. take place.[27] Let us now show that for each execution of *Gradecast* in which the sender is (currently) good, we readily compute the sender's message. Let i j be the label of such an execution (and thus i its good sender). There are two mutually exclusive cases to consider: (a) j is a currently bad player and (b) j is currently good. In case (a) the message $(Q_j^*(i))^*$ sent by j to i in step 3 is contained in A's history of the previous round, and thus in the computed final quantities of step 3. In addition, because both the dealer and i are good, we know that $P_i^*(j) = P_i(j) = Q_j(i)$. Thus we compute $Q_j(i)$ by evaluating polynomial Q_j (which is available as the y-share of eventually bad player j) on input i. Consequently, we compute whether i's message in this execution of *Gradecast* is *disagree (j).* In case (b) we know a priori that j has sent the proper quantity to i, and thus i will not gradecast *disagree* (j). In either case? therefore, whenever the sender is good we compute the sender's message. Consequently, as per Lemma 1, we compute all executions of *gradecast* of step *4.*

5. (For dealer h): For all $(i, j) \in [1, n]^2$, if *you heard* disagree (j) *from player* i *in step* 4, *gradecast* $(i, j, Q_j(i))$.

Because we have computed all executions of *Gradecast* of step 4, in particular we have computed, for each label (i, jj, whether the dealer has heard *disagree* (j) from player i in execution (i,j). That is, we have computed whether execution (i,j) of *Gradecast* occurs. Moreover, whenever this is the case we can also compute the sender's message of execution (i,j). (The proof of this fact is similar to the corresponding fact of step 4; namely. if both i and j are good, then we know that no execution of *Gradecast* labeled i j occurs. Else, if i is good and j is bad, we compute the sender's message, $(i, j, Q_j(i))$, by evaluating polynomial Q_j—available as the y-share of bad player j—on input i.) Thus, as per Lemma 1, from these sender's messages and from the final quantities of step 4, we compute the final quantities of step 5 as well as the grades and values output by the good players in step 5.

6. (For every player i): *For all* $(k, j) \in [1, n]^2$, if *you accepted disagree* (j) *from player* k *in Step* 4 *and*

- *in the previous step you did not accept from the dealer exactly one value of the form* (k, j, V), *where* $V \in [0, p-1]$; *or*
- *you accepted such a value and* $k = i$ *but* $V \neq P_i^*(j)$; *or*
- *you accepted such a value and* $j = i$ *but* $V \neq Q_i^*(k)$,

gradecast badshare.

Because we have reconstructed the grades and values output by the good players in steps 4 and 5, we easily determine whether a good player gradecasts *badshare* in step *6.* Thus, as per Lemma 1 and per the computed final quantities of step 5, we readily compute the final quantities of step 6 as well as all grades and values output in it by the currently good players.

7. (For dealer h): For all i, if *you heard* badshare *from player* i *in step* 6, *gradecast* $(i, P_i(y), Q_i(x))$.

Given the final quantities of step 6 and the grades and values output by the good players in step 6, we compute per Lemma 1 the final quantities of step 7 as well as the grades and values output by the good players of step 7.

8. (For every player i): *If*

(a) *you gradecasted* badshare in *step* 6: *or*

(b) *you accepted* badshare *from more than t players in step 6: or*

[27] Recall the notation established at the end of section 6.

(c) for *each* player j whose gradecast of badshare in step 6 you have accepted, *if* a step ago you did not accept from the dealer the gradecast of a value (j, U, V)—where U and V are t-degree polynomials—or, if you accepted such a value but $Q_i^*(j) \# U(i)$ or $P_i^*(j) \neq V(i)$,
distribute badshare.

This step consists of a single round which we now denote by r. We then compute $M_r^{AG \to EB}$ by computing which good players distribute bcdshare. This determination is easily made from the computed senders' messages, grades, and outputs of step 6 and from the following three facts: (a) If both i and j are good, then i does not distribute badshare; (b) because the dealer is good, $Q_i^*(j) = Q_i(j)$ and $P_i^*(j) = P_i(j)$; and (c) if i is good and j is bad, then both $Q_i(j)$ and $P_i(j)$ are computable from the available x- and y-shares of bad player j. Thus, as per Claim T2-5 and per the final quantities of step 7, we compute the final quantities of step 8.

9. (For each *player* i): If $tally(badshare) \leq$ t distribute recoverable.

Note that we have just computed in step 8 which good players distribute badshare. Moreover, whether or not in that step a bad player sends *badshare* to a good player appears in A's history of step 8 (computed as part of the final quantities of step 8). Therefore, we compute *tally(badshare)* for each currently good player and thus determine which currently good players wish to distribute recoverable in step 9. Thus, as per Claim T2-5 and the final quantities of step 8, we compute the final quantities of step 9.

10. (For each player i):
If $tally(recoverable) > 2t$, output $verification_i = 2$.
Else, *if* $tally(recoverable) >$ t, *output* $verification_i = 1$.
Else, Output $verification_i = 0$.

Since in this last step no good player sends any message, given just the final quantities of step 9, we compute, as per Claim T2-5, the final quantities of step 10.

Because the final history of the adversary is part of the final quantities of step 10, we have established our claim. ■

We can now easily finish the proof of Unpredictability. In Claim T2-6 we saw that A's final history in *GradedSV* depends solely on (1) the final quantities of round 0 and (2) the x- and y-shares of the eventually bad players. Now, in a random execution of GradedSV in which the dealer is always good and the secret s is randomly selected in $[0, m-1]$, the value of the secret is clearly independent of quantities (1). Thus, to prove that no strategy exists for the adversary to guess this secret with probability greater than $1/m$, it is sufficient to show that the dealer's secret is also independent of the x- and y-shares of the $t' \leq t < n/3$ players corrupted by A. Since we can modify any adversary so that she corrupts an additional $t - t'$ players just prior to finishing her last round of *GradedSV*, we can actually limit ourselves to prove our claim for the case $t' = t$. (In fact: if the adversary is such that the x- and y-shares of the first t' corrupted players are not independent of the dealer's secret, then by adding the shares of $t - t'$ other players we cannot obtain shares that are independent of the secret.) Thus, we now want to prove that for any choice of t eventually bad players, $b_1, \ldots, b_t$, any choice of 2t t-degree polynomials $P_{b_1}(y), Q_{b_1}(x), \ldots, P_{b_t}(y), Q_{b_t}(x)$, and any choice of secret s in $[0, m-1]$, there exists a unique bivariate polynomial $F(x, y)$ such that

(A) $F(b_i, y) = P_{b_i}(y) \ \forall i \in [1, t]$,
(B) $F(x, b_i) = Q_{b_i}(x) \ \forall i \in [1, t]$, and
(C) $F(0, 0) = s$.

We first show F's existence. Set $b_0 = 0$ and let $P_{b_0}(y)$ be the univariate, t-degree polynomial passing through the t + 1 points (0, s) and $(b_i, Q_{b_i}(0))$, $i \in [1, t]$. Then, by Lemma 2, there exists a unique bivariate polynomial F satisfying $F(b_i, y) = P_{b_i}(y)$ $\forall i \in [0,t]$. We now show that F enjoys three of the above required properties:

(A) By construction.

(B) Fix a $\in [1, t]$. We prove that $F(x, b_a) = Q_{b_a}(x)$ by showing that these two t-degree polynomials are equal at $t + 1$ points. In fact, by construction we have $\forall j \in [1, \ldots, t]$ $F(b_j, b_a) = P_{b_j}(b_a) = Q_{b_a}(b_j)$, and $F(0, b_a) = P_0(b_a) = Q_{b_a}(0)$.

(C) By construction, $F(0, 0) = P_0(0) = s$.

The uniqueness of F is thus a consequence of the uniqueness of $P_0(y)$. This proves that "unpredictability" holds for GradedSV and thus completes the proof of Theorem 2. ■

Remark. We have chosen the VSS protocol of [3] as the basis of our *GradedSV* protocol because it relied solely on private and broadcast channels but not on cryptography. (Several beautiful cryptographic VSS protocols are available, but our transformation would have yielded a cryptographic graded VSS protocol, and thus a Byzantine agreement algorithm tolerating only computationally bounded adversaries.) Another ingenious VSS protocol for standard networks that, in addition, possess broadcast channels, was found by Chaum, Crépeau, and Damgård [10]. We could have also adapted their protocol in our setting, but at the expenses of some additional complications since their protocol allows a—controllable but positive—probability of error.

Theorem 2 guarantees that whenever protocol GradedSV is executed with a 1/3-adversary, certain properties hold for the verification values of the good players. These values, however, are internal to the good players and not directly "observable" by the adversary at that point. The following simple lemma, however, shows that these values can be inferred from (a portion of) the adversarial history. We will make use of this result in the next section.

LEMMA **3.** ***At*** the end of any execution of GradedSV, the verification value output by each good player is computable from the final history of the adversary.

Proof. If, in an execution of *GradedSV*, all players are good, then the dealer must have been good throughout the protocol. Thus, due to property 2 of verifiable secret sharing (acceptance of good secrets), we do know that every good player must output 2 as his own verification value. Now assume that one or more players are bad—including, possibly, the dealer. Then the verification value output by a good player i at the end of GradedSV is determined by the number of players that sent him recoverable in step 9. Now, the number of bad players that sent recoverable to i is immediately evident from the messages sent from bad players to good players (messages that are part of the final history of the adversary). Moreover, the number of good players that have sent *recoverable* to i is immediately computable from the messages sent from the good players to the bad players (messages that are also part of the final history of the adversary); in fact, a good player g sends *recoverable* to i if and only if he distributes it to all players, including the bad ones. □

7. Oblivious common coins. In this section, we want to show that processors of a network with private channels can exchange messages so that, in the presence of any 113-adversary, the outcome of a reasonably unpredictable coin toss becomes available to all good players. We start by defining what this means.

7.1. The notion of an oblivious common coin.

DEFINITION 14. *Let P be a fixed-round protocol in which each processor x has no input and is instructed to output a bit r_x. We say that P is an* oblivious common coin protocol (with fairness p and fault tolerance c) *iff for all bits b, for all c-adversaries A, and for all initial quantities IQ, in a random execution of P with A on IQ,*

$$\text{Prob}(\forall \text{ good players } i, r_i = b) \geq p.$$

We will refer to an execution of P as a coin; *by saying that this coin is* unanimously *b, we mean that $r_i = b$ for every good processor i.*

Remarks.

- Our notion of an oblivious coin is a strengthening of Dwork, Shmoys, and Stockmeyer's *persuasive coin* [16], which they implemented for at most $O(n/\log n)$ faults.
- We chose the term *oblivious* to emphasize that, at the end of the protocol, the good processors are "unaware" of whether the outcome of the reasonably unpredictable coin toss is "common." That is, by following the protocol, each good processor computes a bit, but it does not know whether the other good processors compute the same bit. We shall see how to successfully cope with this ambiguity in section 8, but let us first exhibit an oblivious common coin protocol with fault tolerance *1/3*.

7.2. An oblivious common coin protocol.

LEMMA 4. *At the end of every execution of steps 1–3 of OC with a 1/3 adversary, for every good player i and every player j, whether $SUM_{ij} = bad$ can be computed from the final history of the adversary.*

Proof. The adversary's history at the end of step *3* of OC includes the adversary's history at the end of step *1*. Thus, as per lemma *3*, from the latter history we compute the value $verification_i^{hj}$ for all good players i and players j.

Consequently, as per Lemma *1*, we compute each entire execution of *Gradecast* of step *2* of OC. From this information we readily compute, for each good player i and player j, whether conditions (3.*a*), (3.*b*), and (3.*c*) apply, and thus whether $player_{ij}$ (and consequently SUM_{ij}) equals *bad*. □

LEMMA 5. *In every execution of OC with a 113-adversary, for all good players g and* G, $SUM_{gG} \neq bad$.

Proof. This is so because of the following:

(a) By property *3* of graded broadcast, y accepts G's gradecast of G's confidence list.

(b) By the semiunanimity property of *GradedSV*, $|verification_g^{hj} - verification2| \leq 1$ for all labels hj.

(c) By the acceptance-of-good-secrets property of *GradedSV*, $verification_G^{hG} = 2$ for each of the $n - t$ good players h. □

LEMMA 6. *For all $n > 1$, for all executions of $OC(n)$ with a 113-adversary, and for all players j, there exists an integer $sum_j \in [0, n-1]$ such that for all good g, either $SUM_{gj} = sum_j$ or $SUM_{gj} = bad$.*

Proof. We distinguish three mutually exclusive cases.

Case 1: Player j looks bad to all good players. In this case, setting $sum_j = 1$ trivially satisfies our claim.

Case **2:** *j looks okay to a single good player g and bad to all other good players.* In this case, choosing $sum_j = SUM_{gj}$ satisfies our claim for the following reasons. First, notice that SUM_{gj} is a well-defined integer value belonging to the interval

PROTOCOL $OC(n)$
Input for every player i: None.

1. (for every player i): For $\mathbf{j} = 1 \ldots \mathrm{n}$, randomly and independently choose a value $s_{ij} \in [0,\mathrm{n}-1]$. "We will refer to s_{ij} as the ith secret assigned to **j**, or the secret assigned to **j** by i."
   Concurrently run GradedSV n^2 times, one for each label hj, $1 \le h, \mathbf{j} \le \mathrm{n}$. In execution hj, the candidate-secret set is $[0,\mathrm{n}-1]$ "and thus the number of possible secrets equals the number of players," the dealer is h, and the secret is s_{hj}, "i.e., the dealer chooses s_{hj} to be his secret whenever he is good."
   Let $verification_i^{hj}$ be your output of execution hj, "that is, your own opinion about the existence/recoverability of s_{hj}."
2. (for every player i): Gradecast the value $(verification_i^{1i}, \ldots, verification_i^{ni})$. "This is your confidence list, that is, your own opinion about the existence/recoverability of each secret assigned to you."
3. (for every player i): for all j, if
   (a) in the last step, you have accepted j's gradecast of a vector $\vec{e}_j \in \{0,1,2\}^n$, "i.e., j's own confidence list—thus if j's is good, $\vec{e}_j = (verification_j^{1j}, \ldots, verification_j^{nj})$";
   (b) for all h, $|verification_i^{hj} - \vec{e}_j[h]| \le 1$, "that is, your opinion about the recoverability of every secret assigned to **j** differs by at most 1 from the opinion that **j** has gradecasted to you"; and
   (c) $\vec{e}_j[\mathrm{h}] = 2$ for at least $\mathrm{n} - t$ values of h,
   set $\mathrm{player}_{\mathrm{ij}}$ = ok, "meaning that **j** looks okay'to you;" otherwise, set $\mathrm{player}_{\mathrm{ij}}$ = bad, "in which case **j** looks bad to you and he is bad."
4. (for every player i): "Recover all possible secrets:"
   Concurrently run *GradedR* on the n^2 histories of GradedSV that you generated in step 1, and denote by $value_i^{hj}$ your output for execution hj.
   If $\mathrm{player}_{\mathrm{ij}} = \mathrm{bad}$, set $SUM_{ij} = \mathrm{bad}$. Else: Set

$$SUM_{ij} = \left(\sum_{\substack{h \text{ such that} \\ \vec{e}_j[h]=2}} value_i^{hj} \right) \bmod n.$$

   "That is, if player **j** looks okay to i, SUM_{ij} equals the sum modulo n of all those secrets assigned to **j** that **j** himself thinks are optimally verified."
   If for some player **j**, $SUM_{ij} = 0$, output $r_i = 0$; otherwise, output $r_i = 1$.

$[0, n-1]$. This is so because each of the addenda contributing to SUM_{gj} is a well-defined integer (and thus taking the sum of these addenda mod n necessarily yields a value in $[0, n-1]$). Indeed, if $value_g^{hj}$ is an addendum of SUM_{gj}, then in the confidence list of **j** accepted by g, $\vec{e}_j[h] = 2$. Moreover, since **j** looks okay to g, step 3(b) ensures that verification? > 0. In turn, by the verifiability property of *GradedSV*, this guarantees that the corresponding secret is "well shared," that is, that the value output by g running GradedR, $value_g^{hj}$, belongs to the candidate-secret set $[0,\mathrm{n}-1]$, as we wished to prove.

Case **3:** Player **j** *looks* okay to more than one good player. Let g and G be any two such good players—thus $SUM_{gj} \neq \mathrm{bad} \neq SUM_{Gj}$. To begin with, notice that the

value $\vec{e}_j$ is the same for both g and G due to property 1 of graded broadcast. We now show that $SUM_{gj} = SUM_{Gj}$. Indeed, we have

$$SUM_{Gj} = \left(\sum_{\substack{h \text{ such that} \\ \vec{e}_j[h]=2}} value_G^{hj} \right) \bmod n$$

and

$$SUM_{gj} = \left(\sum_{\substack{h \text{ such that} \\ \vec{e}_j[h]=2}} value_g^{hj} \right) \bmod n.$$

First, notice that the set of values h for which $\vec{e}_j[h] = 2$ are the same for both G and g—in fact, both g and G accepted j's gradecast of his own confidence list in step 2 and by virtue of property 1 of any graded broadcast protocol, their accepted lists are equal.. Moreover, corresponding addenda in the two summations are equal. In fact, since j looks okay to G, $\vec{e}_j[h] = 2$ implies that *verification?* > 0, which in turn, due to the verifiability property of *GradedVSS*, implies that all good players will recover the same value as the secret of execution h j of *GradedSV*. □

LEMMA 7.[28] *Let $n > 1$ and let S and $\mathcal{G}$ be subsets of $[1, n]$. Let the set*

$$\mathcal{O} = \{\mathcal{O}_{gj} \in \{ok, bad\} : g \in \mathcal{G},\ j \in [1, n]\}$$

be such that for all $j \in S$, there exists g $\in \mathcal{G}$ such that $\mathcal{O}_{gj} = ok$. Then for all 1/3-adversaries A, in a random execution of $OC(n)$ with $A(n)$ in which $\mathcal{G}$ is the set of always good players and $\forall g \in \mathcal{G}\ \forall j \in [1, n]$ $player_{gi} = \mathcal{O}_{gj}$, the values $\{sum_j : j \in S\}$ are uniformly and independently distributed in $[0, n-1]$.

Before proving Lemma 7, let us consider a simpler but naïve argument. We have three good reasons for doing so: to use this naive argument as an introduction to our subsequent proof; to reassure the reader that our subsequent proof, though admittedly somewhat tedious, at least does not possess any obvious shortcuts; and to bring to light a subtle point that, unless it becomes known, may become a common as well as "fatal" logical trap in similar cryptographic contexts. For simplicity, let us state our naïve argument in a particularly simple case, that is, when S's cardinality equals 1, $S = \{1\}$. In this case, all we have to prove is that the unique sum_j is uniformly distributed in $[0, n-1]$.

Naive argument: If $SUM_{gj} \neq$ bad, then $sum_j = SUM_{gj} = \alpha + \beta \bmod n$, where

$$\alpha = \left(\sum_{\substack{h \text{ such that } h \text{ is good.} \\ verification_j^{hj}=2}} value_g^{hj} \right) \bmod n$$

[28] We condition the uniform and independent distribution of the sum_j's on a rather rich set of events. This is so because Lemma 7 will be invoked in rather diverse contexts, each with its own "conditioning." and we wish to make it very easy to see that it applies properly.

and

$$\beta = \left(\sum_{\substack{\text{h such that h is bad,} \\ \textit{verification}'_j{}'' = 2}} value_g^{hj} \right) \bmod n.$$

(These two values are well defined at the end of *GradedSV*, though no good player knows them because he does not know who else is good.) Value α is uniformly distributed in [0. n−1] and is, in addition, unpredictable to the adversary at the end of step 1. Value β is controllable by the adversary (since each secret that contributes to it might have been chosen by the adversary via a processor h corrupted sufficiently early in step 1). Nonetheless, since at the end of *GradedSV* each of the secrets contributing to SUM_{gj} is fixed, so is β; that is, β is fixed at a point in which α is unpredictable. "Thus" no matter how much the adversary can control β, $\alpha+\beta$ is uniformly distributed in $[0, n-1]$.

Why is this naïve? The flaw in the above argument is that, in principle, the unpredictability of α may be consistent with the fact that, say, $\alpha + \beta$ always equals 0. Indeed, in principle, the adversary may be capable of guarateeing that $\beta = -\alpha \bmod n$ without knowing α nor (necessarily!) β.[29] This "magic correlation," though possible in principle, is actually impossible to achieve in our protocol due to many of its ***specificities,*** which have been ignored by the above reasoning. For instance, our protocol is such that the value β is actually ***"known"*** to the adversary at the end of step 1. This and other specificities are indeed an integral part of the following simulation-based argument, which properly corrects and formalizes the above naïve argument. The reader who, at this point, finds it obvious can proceed to Theorem 3.

Proof of Lemma-7. The proof is by induction on k, the cardinality of the set S. For $k = 0$, our statement is vacuously true. We now prove the inductive step by contradiction. Assume that our statement holds for $k-1$ but not for k. Then a simple averaging argument implies the following proposition.

PROPOSITION P1. ***There exist an integer $n > 1$, a subset $\mathcal{G} \subset [1, n]$, a set of values $\mathcal{O} = \{\mathcal{O}_{gi} \in \{ok, bad\} : g \in \mathcal{G}\ i \in [1, n])$, a subset $S' \subset [1, n]$, whose cardinality*** is ***$k-1$, an additional player $j \notin S'$ such that $\forall i \in S' \cup \{j\} \exists g \in \mathcal{G}\ \mathcal{O}_{gi} = ok$, a set of $k-1$ values $\{v_i \in [0, n-1] : i \in S')$, an additional value $v \in [0, n-1]$, a distinguished***

[29] Mutatis mutandis, consider the following simpler scenario (simpler because it envisages computationally bounded players and thus the possibility of successfully using uniquely decodable encryptions) in which this is indeed the case. Two players desire to compute a common and random bit in the following manner. First, player 1 chooses a random bit b_1 and announces its encryption $E_1(b_1)$. Then player 2 chooses a random bit b_2 and announces its encryption $E_2(b_2)$. Then player 1 releases his own decryption key, d_1, and, finally, player 2 releases his own decryption key, d_2. This will enable both players to compute bits b_1 and b_2 and thus b, their sum modulo 2. Is such a b a random bit if, say, player 2 is bad? The answer is no. Player 2 may force bit b to be 0. Although he cannot predict b_1, he may exploit the fact that player 1 announces his encrypted bit first. (Recall that in our scenario, simultaneity is not guaranteed! Messages arrive by the next clock tick, but the adversary is allowed "rushing.") The strategy of player 2 is as follows. First, he announces the same ciphertext that player 1 does. Then he announces the same decryption key that player 1 does. This is a quite serious problem and does not have easy solutions. Simply requiring that the second player announce a different value than the one announced by the first is not a solution. However, discussion of this point is beyond the scope of this paper. (Let us just say that Micali devised a cryptographic protocol that enables two mutually distrusting people to announce independent values—but the protocol and its proof are not at all straightforward. Dolev, Dwork, and Naor [15] have provided a new type of public-key cryptosystem that would make easy to solve this and similar problems. Neither method, however, can be applied to the context of this paper. where the adversary is not restricted in the amount of computation she can perform and thus could break any public-key cryptosystem.)

player $G \in \mathcal{G}$ *such that* $\mathcal{O}_{Gj} = ok$, *a constant* $\varepsilon > 0$, *a* 1/3-*adversary* A, *a string* H, *and a subset* $B \subset [1, n]$. *whose intersection with* $\mathcal{G}$ *is empty, such that, in a random execution of* OC(n) *with* $A(n)$ *on initial adversarial history* H *and initially bad set* B, *letting "*$\mathcal{G} = AG$*" denote the event that the set of always good players coincides with* $\mathcal{G}$ *and defining*

$$X = (\mathcal{G} = AG) \wedge (\forall i \in [1, n]\ \forall g \in \mathcal{G},\ player_{gi} = \mathcal{O}_{gi}),$$

then

(a) Prob($\mathcal{X}$) > ε;

(b) Prob($\forall i \in S^t$, $sum_i = v_i \mid X$) = $(1/n)^{k-1}$; *and*

(c) Prob($\forall i \in S^t$, $sum_i = v_i \wedge sum_j = v \mid X$) > $(1/n)^k$.[30]

CLAIM L7-0. *Let* n, S^t, $\mathcal{G}$, $\mathcal{O}$, j, v, G, ϵ, A, H, B, *and* X *be as in Proposition* P1, *and let* $\mathcal{Y}$ *be the event defined as follows:*

$$Y = X \wedge (\forall i \in S^t,\ sum_i = v_i).$$

Then in a random execution of steps 1–3 *of* $OC(n)$ *with* A *on initial adversarial history* H *and initially bad set* B *in which* G *is not corrupted! the following holds:*

1. $\mathcal{Y}$ *occurs with positive probability.*

2. *Whether* $\mathcal{Y}$ *occurs is computable on the following inputs:* (2.1) *the set of the always good players,* (2.2) *the vectors* $\vec{e}_x \in \{0,1,2\}^n$ *accepted by at least one good player in step 2, and* (2.3) *for all good players g and for all players* $x \neq j$, *g's history of execution gx of GradedSV.*

3. **If** $\mathcal{Y}$ *occurs (and thus* G $\in \mathcal{G}$ *is good), the secret* s_{Gj} *(i.e., the secret—randomly selected in* $[0, n-1]$*—of execution* G j *of GradedSV, where good G is the dealer) is predictable with probability* $> 1/n$ *on inputs* (2.1), (2.2), *and* (2.3) *above.*

Proof of Claim L7-0.1. First, notice that, because G $\in \mathcal{G}$, G is not corrupted whenever X occurs. 'Thus Prob($\mathcal{Y}$ | G good) = Prob($\mathcal{Y}$) = Prob($\mathcal{Y}$ | X) · Prob($\mathcal{X}$). Now Prob($\mathcal{X}$) > ε by Proposition P1(a), and Prob($\mathcal{Y}$ | X) = $(1/n)^{k-1}$ by Proposition P1(b).

Proof of Claim L7-0.2. Inputs (2.1) and (2.2) are by definition sufficient to determine whether X holds: and if this is the case, $\forall i \in S^t$, $sum_i \neq bad$, and thus $sum_i = sum_{gi}$ for some good player g who has accepted i's gradecast of a vector $\vec{e}_i$ in step 3. Also, inputs (2.3) and (2.4) are more than sufficient to compute which actual value in $[0, n-1]$ sum_{gi} takes because this value depends only on $\vec{e}_i$ and g's history of execution hi of *GradedSV*, $h = 1, \ldots, n$, none of which coincides with execution G j since $i \in S^t$ and $S^t \not\ni j$.

Proof of Claim L7-0.3. If $\mathcal{Y}$ occurs, $SUM_{Gj} \neq bad$ and, on inputs (2.1), (2.2), and (2.3), one can compute all addenda that contribute to SUM_{Gj}, with the singular exception of $value_G^{Gj}$. Indeed. for each $h \neq$ G such that $\vec{e}_j[h] = 2$, the occurrence of Y implies that G is good, that $player_{Gj} = ok$, that $|verification_G^{hj} - \vec{e}_j[h]| \leq 1$, and thus that $verification_G^{hj} > 0$. In turn, $verification_G^{hj} > 0$ implies that the secret of each such execution hj is recoverable no matter what the currently bad players (and those which may become bad while running *GradedR)* may do. In particular, the secret of each such execution hj is recoverable if no more players are corrupted during *GradedR* and the bad players do not send any messages during *GradedR*. Thus when one has the histories of the currently good players (i.e., those in $\mathcal{G}$) at the end of each such

[30] Statement *(c)* is equivalent to the following statement:

(c̃) Prob($\forall i \in S$, $sum_i = v_i \wedge sum_j = v \mid \mathcal{X}$) $\neq (1/n)^k$.

In fact, (c) clearly implies (c̃) and the converse can again be established by averaging.

execution hj of *GradedSV*, one can run *GradedR* so as to reconstruct $value_G^{hj}$ for each of the above labels hj. Having done this, one can trivially compute the sum modulo n of these values; that is, one can compute

$$w = \left(\sum_{\substack{h \text{ such that } h \neq G \\ \vec{e}_j[h]=2}} value_G^{hj} \right) \bmod n$$

and output $v - w$ as a prediction for $value_G^{Gj} = s_{Gj}$. We now show that $\text{Prob}(v - w = value_G^{Gj}) > 1/n$. Indeed, given the above notation, in Proposition P1 we can rewrite inequality (c) as follows:

(c) $\text{Prob}(sum_j = v \mid \mathcal{Y}) \cdot \text{Prob}(\mathcal{Y} \mid X) > (1/n)^k$.

Thus, since $\text{Prob}(\mathcal{Y} \mid X) = (1/n)^{k-1}$, Proposition P1(b) implies that

$$\text{Prob}(sum_j = v \mid \mathcal{Y}) > 1/n.$$

Now, whenever $\mathcal{Y}$ occurs, we have, in particular, $SUM_{Gj} \neq$ bad, and thus $sum_j = value_G^{Gj} + \omega \bmod n$. Therefore, as we wanted,

$$\text{Prob}(value_G^{Gj} = v - \omega \mid \mathcal{Y}) > 1/n. \quad \blacksquare$$

Notice that Claim L7-0 is not (yet!) a violation of the unpredictability of *GradedSV*.[31] To reach such a contradiction, we now show that (letting n, S', $\mathcal{G}$, $\mathcal{O}$, G, j, v, ϵ, A, H, B, and X be as in Claim L7-0 and Proposition P1) Claim L7-0 implies the existence of a 1/3-adversary for GradedSV, A' $(= A'_{n,S',\mathcal{G},\mathcal{O},j,v,G,\epsilon,A,H,B})$, that in a random execution with $GradedSV(n)$ succeeds in achieving the following two goals. First, her random execution with $GradedSV(n)$ coincides with execution G j of GradedSV in a random execution of the first three steps of $OC(n)$ with $A(n)$. Second, she possesses all inputs (2.1), (2.2), and (2.3) relative to said execution of $OC(n)$ with

Informal description of A' $(= A'_{n,S',\mathcal{G},\mathcal{O},G,j,\epsilon,A,H,B})$. Although adversaries A' and A are different and attack different protocols, they both act on networks with n players. Thus for any given player $i \in [1, n]$, we must specify at all points whether he is a player executing $GradedSV(n)$ with A' or a player executing $OC(n)$ with A. We find it convenient to do so by writing i' in the first case and i in the second.

Let us now describe the behavior of A' in a random execution, E', with GradedSV when the dealer is G' and both the adversarial history and the initially bad set are empty. During E', A' orchestrates and monitors portions of a "*virtual*" execution, E, of $OC(n)$ with adversary A. (We thus think of adversary A' as acting in the actual network N'—where $GradedSV(n)$ is executed—and of A as acting in the virtual network N where $OC(u)$ is executed.)

Since we shall only consider n-party executions of protocols GradedSV and OC (where n is as in Proposition P1) in the proof of our lemma, we may more simply write GradedSV and OC instead of, respectively, $GradedSV(n)$ and $OC(n)$.

Adversary A' causes E to start by letting the adversarial history of A be H, the initially bad set be B (where H and B are as in Proposition P1), and the initial

[31] Indeed, for this to be the case, it is necessary that a 1/3-adversary succeed in predicting better than at random the random secret of a good dealer in a random execution between this adversary and *GradedSV*, that is, without assuming that such a random execution is embedded into an execution of *OC* for which certain key quantities are an available inputs.

histories of players 1...., n be those of a random execution of OC. As usual, the first 25 rounds of E consist, of the concurrent execution n^2 times of the 25-round protocol *GradedSV:* one execution for each label h.j (where h and j are player names and h is the dealer of execution hj).

For the first 25 rounds, adversary A' keeps E' in *lockstep* with E, *identifying* execution G j (where G and j are as in Proposition P1) with E'. By "lockstep," we mean that, for each round $p > 0$, the round-p quantities of E' and E depend on and are generated after the round-$(p-1)$ quantities of both E' and E. By "identifying" E', with execution G j of *GradedSV* in E, we mean that A' corrupts processors in N' while interfering with the delivery of messages in N in the following way. Adversary A' corrupts player j' in network N' at round p if and only if A corrupts j in network N at round p. (Since the computation of A starts with initially bad set B, adversary A' corrupts j' in network N' at round 0 for all $j \in B$.) Let us now discuss how A' interferes with the delivery of messages in network N. At every round $p = 1, \ldots, 25$, after A has ended her corruption process and computed the messages from each bad player to each good one for all executions *xy* of *GradedSV,* A' acts *as* follows:

- For each execution $xy \neq Gj$, she delivers the proper messages to the proper recipients in network N.
- For execution Gj, if A outputs m as the message from bad player b to good player g, then A' has b' send m to g' in network N'; vice versa, if good player g' sends a message m' to bad player b' in execution E', then A' delivers m' as the message from g to b in round p of execution Gj.

As we shall prove, after making this description a bit more precise, the virtual execution E thusly generated is actually a "genuine" random execution of the first three steps of $OC(n)$ with A. Moreover, A' will be capable of computing inputs (2.1), (2.2), and (2.3) specified in Claim L7-0. This will enable her to contradict the unpredictability property of *GradedSV*. .

More formal description of A' $(= A'_{n,S',\mathcal{G},\mathcal{O},G,j,\epsilon,A,H,B})$. Let us now describe a bit more precisely the way A' acts in a random execution of $GradedSV(n)$ in network N', where the dealer is G', the candidate-secret set is [0,n -- 1], the dealer's secret is randomly chosen in said candidate-secret set, the initial adversarial history equals the empty string, and the initially' bad set is empty. We have already specified each player's version of *GradedSV* and the mechanics of an execution of an n-party protocol with an adversary. Thus, choosing the players' and adversary's coin tosses at random, our description of A' specifies the values taken by all possible players' quantities $H^r_{i'}$, $M^r_{i'\rightarrow}$, $M^r_{\rightarrow i'}$, $C^r_{i'}$, and $R^r_{i'}$ as well as the values taken by the adversarial quantities $H^r_{A'}$, $C^r_{A'}$, and $R^r_{A'}$ and by the sets BAD'^r and GOOD'^r.

In order to determine her actions in network N', adversary A' will construct only in part the quantities $H^{r,xy}_i$, $M^{r,xy}_{i\rightarrow}$, $M^{r,xy}_{\rightarrow i}$, $C^{r,xy}_i$, and $R^{r,xy}_i$, where $i \in [1,n]$ and $xy \in [1,n]^2$, but she will construct all of the possible quantities H^r_A, C^r_A, and R^r_A and the sets BAD^r and GOOD^r.

She generates these quantities with the same mechanics of a random execution of the first three steps of protocol $OC(n)$ with adversary A. initially bad set B, and initial adversarial history H. The quantities generated thusly by A': however, fall short of constituting such a random execution because they are *incomplete.* Indeed, they miss some Gj-labeled quantities—e.g., $H^{r,Gj}_i$ whenever $i \in \text{GOOD}^r$.[32]

[32] Since the quantities reconstructed by A' relative to network N do not quite constitute an execution of $OC(n)$ with A, and since it can be recognized that these quantities can be integrated so as to yield a virtual execution only after the entire behavior of A' has been described. it. would be improper during our description to use suggestive expressions—such as "good at round r"—that,

It will be clear from our description. however. that if there is no r for which BAD^r contains G, then if one were to integrate these missing quantities with the corresponding quantities of E' (i.e., the execution of *GradedSV*(n) with A' in network N'), one would obtain a random execution of *OC*(n) with **A,** on initially bad set B and adversarial history H, in which G is not corrupted.

Notice that A' is active in network N' for 25 rounds because *GradedSV*(n) is a 25-round protocol. Notice also that the number of rounds in step 1 of *OC*(n) is also 25 if one imagines **(as** we do) that in each execution xy of *GradedSV*(n), the dealer randomly chooses the secret in round 0. Thus for r $= 1\ldots,25$ and each label $xy \in [1,n]^2$, the ith round of step 1 is the ith round of an execution of *GradedSV*. Indeed, for $r = 0,\ldots,25$, adversary A' decides her action at round r in network N' "simultaneously" with her generation of round-r quantities in virtual network N (i.e., after having generated round-(r $-$ 1) quantities in network N and before generating round-(r_1) quantities in network N).

To facilitate seeing that the round-r quantities generated by A' for virtual network N follow the mechanics of an execution of *OC*(n) with A on initially bad set B and adversarial history H. we break the instructions for this generation into instructions 1∗–4∗, thus matching the instructions 1–4 that we used in section **3** to describe how a protocol is executed with an adversary.

LOCAL DEFINITION. Let $b_1,\ldots,b_k$ be the *elements* of subset B; denote by $\mathcal{L}$ the set of all execution labels of *GradedSV* in step 1 (*i.e.,* $\mathcal{L} = \{hj : 1 \leq h, j \leq n)\}$ and by $\mathcal{L}^-$ the set $\mathcal{L} - \{Gj)$.

Instructions for round $r = 0$.

(In virtual network N):

Set $H_A^0 =$ H, BAD^0 = B, GOOD' $= [1,n] -$ BAD^0, and $C_A^0 =$ s. Then construct a binary string R_A by selecting randomly and independently each of its coins, and set $R_A^0 = R_A$.[33]

For all xy $\in \mathcal{L}^-$, randomly and independently select S$_{\text{,}}$ in [0, n $-$ 1], and let C_{xy}. denote the sequence of random bits used for this selection.

For all xy $\in \mathcal{L}^-$ and for all i $\in [1,n]$, construct an infinite bit string R_i^{xy} by choosing each of its bits randomly and independently. [34] Then reset $R_x^{xy} := C_{xy} \circ R_x^{xy}$.

Finally. for all $xy \in \mathcal{L}^-$ and for all $i \in [1,n]$, set $C_i^{0,xy} = \epsilon$, $R_i^{0,xy} = R_i^{xy}$, and $M_{i\rightarrow}^{0,xy} = M_{\rightarrow i}^{0,xy} =$ (s,...,s). and, if i $\neq x$, $H_i^{0,xy} = ((z,n), M_{\rightarrow i}^{0,xy}, C_i^{0,xy})$—otherwise (i.e., $i =$ x), set $H_i^{0,xy} = ((x,n,S_{xy}), M_{\rightarrow i}^{0,xy}, C_i^{0,xy})$.

"In an execution of *GradedSV*(n) with dealer x, the input for any player other than x is (x,m), that is, the name of the dealer and an encoding of the candidate-secret set, $[0.m-1]$. The private input for x is instead (x,m,s), that is, x is given his secret **as** an additional input, unrelated to his sequence of future coin tosses. (Therefore, should the dealer be corrupted by the adversary at round 1, she would discover his input secret but not the random choices made to come up with that secret, even if it were randomly selected.) In any execution xy of *GradedSV*(n) **as** a subprotocol of *OC*(n), there are two peculiarities. First, m $= n$ (which is easily reflected in the initial histories of the players in execution xy). Second, the dealer x of execution xy is not given his secret S_{xy} **as** an outside input; rather, he randomly chooses it in $[0, n-1]$

---

though very useful in building up intuition, presuppose that we are already dealing with a genuine execution. Notice, in fact, that all quantities relative to the virtual network N are constructed using only a syntactic description. Only in our comments do we use suggestive language.

[33] This is expressed thusly for convenience. In reality, each R_A will be constructed on an "as-needed basis."

[34] Again, in reality. each R_i^{xy} will be constructed on an "as-needed basis."

prior to calling $GradedSV(n)$. Therefore, should the adversary corrupt player x at round 1 of her execution with $OC(n)$, then, she should be able to discover not only x's random secret relative to each execution xy but also the coin tosses that led x to choose S_{xy}. This is exactly what is accomplished by the above steps (which also accomplish giving these secrets suitable names—i.e., S_{xy}—and making it evident that all of them (for xy $\neq Gj$) are known to A')."

(In network N'):

Corrupt processors $b'_1, \ldots, b'_k$,

Instructions for $r = 1, \ldots, 25$.

0* (in virtual network N):

$\text{TEMPH}_A^r := H_A^{r-1}$; $\text{TEMPR}_A^r := R_A^{r-1}$; $\text{TEMPGOOD}^r := \text{GOOD}^{r-1}$; $\text{TEMPBAD}^r := \text{BAD}^{r-1}$.

1* (in virtual network N):

For all g $\in \text{GOOD}^{r-1}$ and for all xy $\in \mathcal{L}^-$, generate $M_{g\rightarrow}^{r,xy}$, "the messages g wishes to send in this round (which may be reset if g is corrupted in this round)," $C_g^{r,xy}$, and $R_g^{r,xy}$ by running $GradedSV(n)_g$ on input $H_g^{r-1,xy}$ and coins $R_g^{r-1,xy}$.

2* (in virtual network N):

For all $xy \in \mathcal{L}^-$, for all g $\in \text{GOOD}^{r-1}$, for all b $\in \text{BAD}^{r-1}$, $\text{TEMPH}_A^r := (\text{TEMPH}_A^r, g, b, xy, M_{g\rightarrow}^r[b])$.

"In other words; for each message m from a good player g to a bad player b computed by running $GradedSV$ relative to label xy $\in \mathcal{L}^-$, deliver m to A as usual (i.e., specifying the name of the sender, the recipient, and the execution label). Then":

For each message m' received by a bad player b' from a good player g' in network N' at round r, $\text{TEMPH}_A^r := (\text{TEMPH}_A^r, \text{g}, b, Gj, \text{m}')$.

3* (in virtual network N):

Run A on input TEMPH_A^r and coins TEMPR_A^r.

(In network N' and in virtual network N):

If in this execution of step **3** A has output **j** "as the next player to corrupt," then HALT—"both the virtual execution in network N and the real execution in network N' are terminated"—and output a random value in $[0, \text{n} - 1]$ as your guess for the secret of dealer G'. "You will be correct with probability $1/n$. Else":

(In virtual network N):

If A has output q $\in \text{TEMPGOOD}^r$ and made the sequence of coin tosses C, then

$\text{TEMPBAD}^r := \text{TEMPBAD}^r \cup \{q\}$, $\text{TEMPGOOD}^r := \text{TEMPGOOD}^r - \{q\}$,

$\text{TEMPH}_A^r := (\text{TEMPH}_A^r, xy, H_q^{r-1,xy}, C_q^{r,xy}, \text{C})$,

$\text{TEMPR}_A^r := \text{TEMPR}_A^r / C$,

$\forall xy \in \mathcal{L}^-$, $\forall g \in \text{TEMPGOOD}^r$, $\text{TEMPH}_A^r := (\text{TEMPH}_A^r, g, \text{q}, xy, M_{g\rightarrow}^{r,xy}[q])$.

(In network N'):

Corrupt q' in network N', thereby learning his history $H_{q'}^{(r-1)}$ and coin tosses $C_{q'}^{(r-1)}$, as well as $M_{\rightarrow q'}^r[g']$ for all currently good players g',

"i.e., as well as each message sent by a currently good player g' to q'. Note that a player g' (respectively, b') is currently good (respectively, bad) in network N' if g $\in \text{TEMPGOOD}^r$ (respectively: b $\in \text{TEMPBAD}^r$) in the virtual network N."

(In virtual network N):

$\text{TEMPH}_A^r := (\text{TEMPH}_A^r, xy, H_{q'}^{r-1}, C_{q'}^{r-1}, C)$,

$\forall g \in \text{TEMPGOOD}^r$, Vb $\in \text{TEMPBAD}^r$, $M_{g\rightarrow}^{r,Gj}[b] := M_{g'\rightarrow}^r[b']$.

Go to step 3* "to corrupt next processor."

If "A no longer wishes in this round to corrupt. additional players," if in this execution of step **3** A has output, for all $xy \in \mathcal{L}$ and for all $b \in \text{TEMPBAD}^r$, a vector

$M_b^{xy} \in (\{0,1\}^*)^n$ "as b's round-r messages" and made the sequence of coin tosses C, then:

(In virtual network N):

$\forall xy \in \mathcal{L}^-$, $\forall b \in \text{BAD}^r$, $M_{b\rightarrow}^{r,xy} := M_b^{xy}$,

$\text{TEMPH}_A^r := (\text{TEMPH}_A^r, \text{C})$ "so that she can reconstruct the bad players' messages of round r,"

$\text{TEMPR}_A^r := \text{TEMPR}_A^r / C$;

(In network N'):

$\forall b \in \text{BAD}^r$, $M_{b'\rightarrow}^r := M_b^{r,Gj}$.

"In other words, for each message m from a bad player b to a currently good player g in network N relative to execution Gj, have b' send m to g' as his round-r message in network N'."

4* (In virtual network N):

"Adjust the final round-r quantities as follows." Letting $\mathcal{C}$ be the sequence of coin tosses that A has made since the last execution of step 2,

$H_A^r := \text{TEMPH}_A^r$; $C_A^r := \mathcal{C}$; and $R_A^r := \text{TEMPR}_A^r$;

$\text{GOOD}' := \text{TEMPGOOD}^r$ and $\text{BAD}^r := \text{TEMPBAD}^r$;

$\forall xy \in \mathcal{L}^-$, $\forall k, \text{i} \in [1, \text{n}]$, $M_{\rightarrow i}^{r,xy}[k] := M_{k\rightarrow}^{r,xy}[i]$; and

$\forall k \in [1,\text{n}]$, $\forall i \in \text{BAD}'$, $M_{\rightarrow i}^{r,Gj}[k] := M_{k\rightarrow}^{r,Gj}[\text{i}]$;

"Adversary A' does not know the messages exchanged among good processors in network N' and thus does not construct the corresponding messages in execution Gj."

$\forall xy \in \mathcal{L}^-$, $\forall g \in \text{GOOD}^r$, $H_g^{r,xy} := (H_g^{r-1,xy}, M_{\rightarrow g}^{r,xy}, C_g^{r,xy})$;

"A' does not know the histories of the good players in N' and thus does not construct $H_g^{r,Gj}$."

$\forall xy \in \mathcal{L}$, $\forall b \in \text{BAD}^{r-1}$, $H_b^{r-1,xy} := (H_b^{r-1,xy}, \text{bad})$, and $\forall b \in \text{BAD}^r - \text{BAD}^{r-1}$, $H_b^{r,xy} := (H_b^{r-1,xy}, C_b^{r,xy}, \text{bad})$.

"Thus far, each time that a new round was added to the partial virtual execution of OC with A, the execution of $GradedSV$ with A' also progressed one round. At this point, however, the rounds added to the partial virtual execution only allow A' to make additional internal computations and, possibly, additional corruptions, but her execution with $GradedSV$ remains at round 25.

At the start of the execution of step 2 of $OC(n)$, the prior history of a good player consists of an n^2-vector, $\{H_g^{25,xy} : xy \in \mathcal{L}\}$. But at this point of the computation in the virtual network N, there is no quantity $H_g^{25,Gj}$. However, there is a quantity $H_g^{25,xy}$ whenever $g \in \text{GOOD}^{25}$ and $xy \in \mathcal{L}^-$. Such a quantity $H_g^{25,xy}$ specifies the quantity $verification_i^{xy}$ (via some proper input function $\mathcal{I}_g^{steps2-3}$). Thus for all $xy \in \mathcal{L}^-$ and for all $g \in \text{GOOD}^{25}$, $verification_i^{xy}$ is computable by A' because she has already computed $H_g^{25,xy}$."

Additional instructions for round 25.

LOCAL DEFINITION. We denote by steps 2–3 the protocol consisting of steps 2 and 3 of $OC(n)$.

For all $g \in \text{GOOD}^{25}$, set $verification_g^{Gj}$ to be the verification value output at round 25 by player g' in the execution of $GradedSV(n)$ with you in network N'.

"Although these values are internal to good processors of network N' and thus invisible to you, you can compute them—by virtue of Lemma 3—from your knowledge of the sets of good and bad players and the messages exchanged between good and bad players."

For all g $\in$ GOOD^{25}, set $H_g^{25.Gj} = rc_g^{Gj}$, where rc_g^{Gj} is a. reserved character that (via the input function $\mathcal{I}_g^{steps2-3}$) specifies *verification?* .

Execute subprotocol *steps* 2–3 with A (with the initial adversarial history being the computed quantity H_A^{25}, the initially bad set being the computed quantity BAD^{25}, and the prior history of each good player g being the ("thusly completed") vector $\{H_g^{25,xy} : xy \in \mathcal{L}\}$). handling corruptions as follows.

"In this execution of *Gradecast* as a subprotocol, you know the sender, the sender's message, the set of initially bad players, the active adversary and the initial adversarial history. Thus you do not need to know the players' prior histories exactly in order to exactly reconstruct. all messages exchanged up to the next corruption. Indeed, the good players do not rely on their prior histories (more than is needed to figure out which message to gradecast). Once a corruption occurs, however, in order to update the adversarial history in a proper manner, you need the corrupted player's prior history."

Whenever A corrupts an additional player k, corrupt k' in network N' so as to find his current. history, H'_k. In the current *steps* 2–3 history of k, replace the reserved character rc_k^{Gj} by the string H'_k and deliver the thusly updated history to A (in the syntactically proper manner).

Instructions for predicting the secret.

"If you have not already predicted the secret of G' at random, do the following":

Detect whether event $\mathcal{Y}$ "of Claim L7-0" occurs.

"You can do that by virtue of Claim L7-0 because you may compute all inputs (2.1), (2.2), and (2.3) envisaged in that claim."

If $\mathcal{Y}$ has not occurred, then predict the secret of G' by outputting a random number in $[0,n-1]$.

Else output the value v – w mod n as your prediction of the secret of G'. "v is the value of Proposition P1 and Claim L7-0 and

$$\omega = \left(\sum_{\substack{h \text{ such that } h \neq G \\ verification_j^{hj} = 2}} value_G^{hj}, \right) \bmod n."$$

This ends our description of A'. Notice that Steps 2 and 3 of OC take, respectively, four rounds and one round. Thus the subprotocol consisting of Steps 1–3 of OC is a 30-round protocol.

Let us now more precisely claim (without proof) that if one replaces the missing quantities constructed by A' for the virtual network N with the "corresponding" quantities that arise in the execution of adversary A' with *GradedSV*(n) in network N', one obtains a random execution of Steps 1—3 of $OC(n)$ with A.

CLAIM L7-1. *Let E' be a random execution. of protocol GradedSV*(n) *with adversary A' in which the initially bud set is empty, the initial adversarial history is the empty string, the dealer is G', the candidate-secret set is* $[0,n-1]$, *and the secret is mndomly selected in* $[0,n-1]$.

- *For all* $r = 0,\ldots,25$. *for all* $xy \in \mathcal{L}^-$. *and for all* $i \in [1,n]$, *let* $H_i^{r,xy}$, $M_{i-}^{r,xy}[j]$, $M_{-i}^{r,xy}[j]$, $C_i^{r,xy}$, GOOD^r, BAD', H_A^r, *and* C_A^r *be the network-N quantities generated by A' during* E'.
- *For all* $r = 1.\ldots.25$ *and for all* $g \in$ GOOD', *let* $H_{g'}^r$, $M_{g'-}^r$, $M_{-g'}^r$, *and* $C_{g'}^r$, *be, respectively, the round-r history. messages sent, messages received, and coin tosses of player g' in* E'.

- *For all* $r = 0,\ldots,25$ *and for all* $g \in \text{GOOD}^r$, *set* $H_g^{r,Gj} = H_{g'}^r$, $M_{g\rightarrow}^{r,Gj} = M_{g'\rightarrow}^r$, $M_{\rightarrow g}^{r,Gj} = M_{\rightarrow g'}^r$, *and* $C_g^{r,Gj} = C_{g'}^r$.[35]
- *For all* $r = 1,\ldots,25$ *and for all* $xy \in \mathcal{L}$, *set* $H_i^r = \{H_i^{r,xy} : xy \in \mathcal{L}\}$, $M_{i\rightarrow}^r = \{M_{i\rightarrow}^{r,xy} : xy \in \mathcal{L}\}$, $M_{\rightarrow i}^r = \{M_{\rightarrow i}^{r,xy} : xy \in \mathcal{L}\}$, *and* $C_i^r = \{C_i^{r,xy} : xy \in \mathcal{L}\}$.
- *For all* $r = 26,\ldots,30$ *and for* all $i \in [1, n]$ *let* H_i^r, $M_{i\rightarrow}^r$, $M_{\rightarrow i}^r$, C_i^r, GOOD^r, BAD^r, H_A^r, C_A^r, *and* R_A^r *be the network-N quantities generated by* A' *when executing protocol steps* 2–3 *with* A *during* E'.

Then the sequence of tuples

$$E = E_0, \ldots, E_{30},$$

where

$$E_r = (H_1^r,\ M_{1\rightarrow}^r,\ M_{\rightarrow 1}^r,\ C_1^r,\ R_i^r, \ldots,$$
$$H_n^r,\ M_{n\rightarrow}^r,\ M_{\rightarrow n}^r,\ C_n^r,\ R_n^r,\ H_A^r,\ C_A^r,\ R_A^r,\ \text{BAD}^r, \text{GOOD}^r),$$

is a random execution (up to round 30) *of protocol* $OC(n)$ *with adversary* A, *on initially bad set* B *and initial adversarial history* H, *in which player* G *is not corrupted.*

The above claim follows from our description of A' and by the observation that all n^2 secrets of E have been randomly and independently selected in $[0, n-1]$: secrets s_{xy} for $xy \in \mathcal{L}$ by construction, and secret s_{Gj} (i.e., the secret of G' in execution E' of $GradedSV(n)$ with A') by hypothesis.

Notice that A' is a 113-adversary (because A is a 113-adversary and because A' corrupts a player in E' if and only if A corrupts the corresponding player'in E), that A' may compute all inputs (2.1), (2.2), and (2.3) envisaged in Claim L7-0, and that A' never corrupts dealer G in an execution with $GradedSV(n)$ when the initially bad set is empty and the initial aclversarial history is the empty string.

Let us now show that A' can predict the secret of G' with probability $> 1/n$. Indeed, whenever A wishes to corrupt G, A' halts, outputting a random number in $[0, n-1]$ as her prediction of the secret of G'. Thus she will be correct with probability $1/n$ in these cases. If A does not corrupt G in E, but neither does event $\mathcal{Y}$ (which A' detects), then A' again guesses the secret of G' at random and is right with probability $1/n$. However, whenever $\mathcal{Y}$ occurs, which by virtue of Claims L7-0 and L7-1 is with positive probability, then A' (per Claim L7-0) correctly guesses the secret of G' with probability $> 1/n$. Finally, because the event that A does not corrupt G occurs whenever $\mathcal{Y}$ occurs, we have that A' correctly guesses the secret of G' with probability $> 1/n$. This contradiction of the unpredictability property of $GradedSV$ establishes Lemma 7. □

THEOREM 3. OC *is an expected-polynomial-time 32-round oblivious common coin protocol with fairness* $> .35$ *and fault tolerance* $1/3$.

Proof. The claims regarding round complexity and running time are easy to verify. (Recall that—though in a "hidden" way—we do make use of message bounds.) Let us thus prove the other claims. We start with some convenient notation.

Let A be a 1/3-adversary and $\mathcal{IQ} \in \mathcal{IQ}_n^{OC}$ be proper initial quantities for OC. Then in an execution of $OC(n)$ with a 113-adversary on $\mathcal{IQ}$, let C_0 denote the event that the coin is unanimously 0, C_0^{good} denote the event that sum, $= 0$ for some good player g (i.e., the coin is unanimously 0 "thanks to a good player"), and C_1 denote the event that the coin is unanimously 1. Correspondingly, let P_0, P_0^{good}, and P_1 denote

[35] Notice that some of these quantities might have already been computed by 4' for virtual network N, in which case they would be reset to the same value.

the probabilities of C_0, C_0^{good}, and C_1, respectively, in a random execution of $OC(n)$ with $A(n)$ on $\mathcal{IQ}$. (Notice that $C_0 \neq \neg C_1$, where $\neg E$ denotes the complement of an event E.)

We are now ready to lower-bound both P_0 and P_1.

Lower-bounding P_0. Since $P_0 \geq P_0^{good}$, we lower-bound P_0 by lower-bounding P_0^{good}.[36] If S is a subset of $[1,n]$, let "$AG = S$" denote the event that the set of the always good players coincides with S. Notice that if $\text{Prob}(AG = \text{S}) > 0$ then $|S| > 2n/3$ and that $\sum_{|S|>2n/3} \text{Prob}(AG = S) = 1$. (In fact, in any execution with a 113-adversary, there must be at least $2n/3$ always good players.) Also notice that (because for all good players G and g, $SUM_{Gg} \neq$ bad by Lemma 5) Lemma 7 implies that whenever $AG = S$ occurs, the values {sum, : g $\in S$} are independently and uniformly distributed in $[0, n-1]$. Thus, because it is sufficient that any such value equals 0 for the coin to be unanimously 0, and because of S's cardinality, we have

$$\text{Prob}(\neg C_0^{good} \mid AG = S) < (1 - 1/n)^{2n/3} < e^{-2/3}.$$

Hence

$$\text{Prob}(C_0^{good} \mid AG = S) > 1 - e^{-2/3}.$$

Thus

$$\begin{aligned}\text{Prob}(C_0^{good}) &= \sum_{|S|>2n/3} \text{Prob}(C_0^{good} \mid AG = S) \cdot \text{Prob}(AG = S) \\ &> \sum_{|S|>2n/3} (1 - e^{-2/3}) \,\text{Prob}(AG = S) \\ &= (1 - e^{-2/3}) \sum_{|S|>2n/3} \text{Prob}(AG = S) = 1 - e^{-2/3}.\end{aligned}$$

Lower-bounding P_1. If S $\subset [1,n]$, let bad $\neq S$ denote the following event: S = {j $\in [1,n]$: $\forall$ good g, $SUM_{gj} \neq$ bad). Notice that if $\text{Prob}(bad \neq \text{S}) > 0$, then S's cardinality is greater than $2n/3$. (Indeed, $SUM_{gG} \neq$ bad for all good players g and G.) Also notice that $\sum_{|S|>2n/3} \text{Prob}(bad \neq \text{S}) = 1$. (Indeed, in any of our random executions; there must be more than $2n/3$ good players.)

We now lower-bound P_1 as follows.

$$\begin{aligned}P_1 &= \sum_{|S|>2n/3} \text{Prob}(\forall j \in S\ sum_j \neq 0 | bad \neq S) \cdot \text{Prob}(bad \neq S) \\ &\overset{\text{Lemma 7}}{=} \sum_{|S|>2n/3} (1 - 1/n)^{|S|} \cdot \text{Prob}(bad \neq S) \\ &\geq \sum_{|S|>2n/3} (1 - 1/n)^n \cdot \text{Prob}(bad \neq S) > e^{-1} \cdot \sum_{|S|>2n/3} \text{Prob}(bad \neq S) = e^{-1}\end{aligned}$$

Since our lower bounds on P_0 and P_1 do not depend on n, A, or IQ, we have proved that the fairness of protocol OC is $\min(P_0, P_1) = \min(1 - e^{-2/3}, e^{-1}) = e^{-1} > .35$. □

[36] The bad players may also contribute to raising the probability of the coin being unanimously 0. For instance, it is enough that, for some bad player j, the adversary acts so that for all good g, $SUM_{Gj} \neq$ bad. By Lemma 7, sum_j then has a $1/n$ chance of being equal to 0. Our lower bound, however, must hold for all possible adversaries; thus we have to disregard this probability from our computation since it may be 0 for some adversaries. Instead, we must consider and guard against such possible behavior of the adversary when lower-bounding the probability of the coin being unanimously 1.

8. Byzantine agreement from oblivious common coins.

8.1. The notion of Byzantine agreement. When Byzantine agreement is needed, the values to be agreed upon may have arbitrary length. Without loss of generality, however, we restrict our attention to the case where every initial value is a single bit: in fact: in [13] and [37], it is proved that general Byzantine agreement is reducible to the binary case in a constant number of rounds.

DEFINITION 15. *We say that a protocol P is a* Byzantine agreement protocol (with fault-tolerance *c) if, for all c-adversaries A, any string H_A^0, any number of players n, and any bits $b_1, \ldots, b_n$, in any execution of $P(n)$ with adversary A on initial adversarial history H_A^0 and inputs $b_1, \ldots, b_n$, there exists a bit d such that the following two properties hold:*

1. Consistency: *Every good player that halts outputs d.*
2. Validity: If *there exists a bit b such that, for all initially good player i, $b_i = b$, then $d = b$.*

Notice that the above definition does not require that a Byzantine agreement protocol ever terminate. A Byzantine agreement protocol is most interesting, however, only if it terminates with positive probability, has high fault tolerance? and requires only a "moderate computational effort" from the good players.

8.2. An optimal Byzantine agreement protocol. We are finally ready to construct our Byzantine agreement protocol from our discussed primitives. It consists of three basic subprotocols: $\mathcal{P}_r$, $\mathcal{P}_1$, and $\mathcal{P}_0$. Subprotocol $\mathcal{P}_r$ includes instructions for "randomly flipping" an oblivious common coin. Protocols $\mathcal{P}_0$ and $\mathcal{P}_1$ actually consist of protocol $\mathcal{P}_r$, where the outcome of the coin flip is *forced* to be, respectively, 0 and 1. Thus: although the coin flips of subprotocols $\mathcal{P}_0$ and $\mathcal{P}_1$ are predictable, they have the advantage that all good processors are "aware" of the result, and this result is always the same for all good processors.

The goal of protocol $\mathcal{P}_r$ is to give the network a chance of reaching an "oblivious agreement" (i.e., with positive probability. all players adopt the same bit without knowing that this has happened). The goal of protocols $\mathcal{P}_0$ and $\mathcal{P}_1$ is to provide a *proof*, if all players are in oblivious agreement, that agreement has indeed been reached, so as to allow everyone to terminate. More precisely, if the good players obliviously agree on 0 (respectively, 1), an execution of $\mathcal{P}_0$ (respectively. $\mathcal{P}_1$) makes them aware that they are in agreement on 0 (respectively, 1) and terminate.

Our Byzantine agreement protocol is not a fixed-round protocol. Rather, each good player i keeps on executing, in order, subprotocols $\mathcal{P}_r$, $\mathcal{P}_0$, and $\mathcal{P}_1$ until he *individually* terminates. It thus may happen that different good processors terminate at different rounds. Nonetheless, in a random execution with a 113-adversary, our protocol terminates with probability 1, and when that happens, the outputs of all good players, though produced at different times, will always satisfy the consistency and validity requirements.

THEOREM 4. *BA is a Byzantine agreement protocol with fault tolerance $1/3$ and runs in expected polynomial time and in an expected constant number of rounds.*

(More precisely, there exists a polynomial Q and constant c such that, for any number of players n, any 113-adversary A, any initial quantities IQ. and any positive integer k. the probability that. in randomly executing $BA(n)$ with A on IQ, the protocol does not halt within $Q(n)k$ BA-steps and ck rounds is less than 2^{-k}.)

Proof. Let us start by establishing a convenient notation.

LOCAL DEFINITIONS. *In an execution of BA, we call a good player* dead if *he has already terminated and* alive *otherwise.*

PROTOCOL $BA(n)$

Input for player i: b_i, a bit. "We actually consider b_i as a variable of player i whose initial value coincides with i's input bit."

Code for every P layer i.

0: For all players j, set $B_j = 0$. "B_j represents the last one-bit message received from player j."

(Subprotocol $\mathcal{P}_r$.)

1: Distribute b_i.

2: For all j, if $b_j^* \in \{0,1\}$, then reset $B_j := b_j^*$; else, reset $b_j^* := B_j$. Let $count_i = tally(1)$.

"In other words, for the purpose of computing $tally(1)$, if you did not receive a bit from player j, assume that he *virtually* sent you the same bit that he *really* sent you last."

Run $OC(n)$ and let r_i be your output. Then:

(a) If $count_i \in [0, n/3)$, then reset $b_i := 0$. Else:

(b) If $count_i \in [n/3, 2n/3)$, then reset $b_i := r_i$. Else:

(c) If $count_i \in [2n/3, n]$, then reset $b_i := 1$.

(Subprotocol $\mathcal{P}_0$.)

3: Distribute b_i.

4: For all j, if $b_j^* \in \{0,1\}$, then reset $B_j := b_j^*$; else, reset $b_j^* := B_j$. Let $count_i = tally(1)$. Then:

(a) If $count_i \in [0, n/3)$, then output 0, distribute 0 in next round, and TERMINATE

"In a round from now, you will be dead and will keep on *virtually* distributing 0. Every other good player is either dead and his output is 0, or will terminate: outputting *0.*"

(b) If $count_i \in [n/3, 2n/3)$, then reset $b_i := 0$. Else:

(c) If $count_i \in [2n/3, n]$, then reset $b_i := 1$.

(Subprotocol $\mathcal{P}_1$)

5: Distribute b_i.

6: For all j, if $b_j^* \in \{0,1\}$, then reset $B_j := b_j^*$; else, reset $b_j^* := B_j$. Let. $count_i = tally(1)$. Then:

(a) If $count_i \in [0, n/3)$, then reset $b_i := 0$. Else:

(b) If $count_i \in [n/3, 2n/3)$, then reset $b_i := 1$. Else:

(c) If $count_i \in [2n/3, n]$, then output *1,* distribute *1* in next round, and TERMINATE.

"In a round from now: you will be dead and will keep on *virtually* distributing *1.* Every other good player is either dead and his output is *1,* or will terminate outputting *1.*"

Go to step 1.

Let $\mathcal{P} \in \{\mathcal{P}_r, \mathcal{P}_1, \mathcal{P}_0\}$ and $b \in \{0,1)$. Within an execution of BA, we say that at the start (at the end) of an execution of subprotocol $\mathcal{P}$, the network is in agreement *on b if, for all good players g, either g is dead and his output is b or he is alive and the current value of variable b_g is b.*

We say that at the start (at. the end) of an, execution of $\mathcal{P}$, the network is in agreement *if there exists a. bit b such that the network is in agreement on b.*

CLAIM *T4-1. For any subprotocol $\mathcal{P} \in \{\mathcal{P}_r, \mathcal{P}_1, \mathcal{P}_0\}$, any execution of $\mathcal{P}$ with a 1/3-adversary, and any alive good players g and G, $|count_g - count_G| < n/3$.*

Proof. Only the bad processors may send different bits to different players in the same step. Thus, at any given step, the difference between the tallies (of *1)* of two good players is upper-bounded by the number of currently bad players and thus by $n/3$. ■

CLAIM T4-2. *For all* $\mathcal{P} \in \{\mathcal{P}_r, \mathcal{P}_1, \mathcal{P}_0\}$, *for all executions of* $\mathcal{P}$ *with a* 1/3-*adversary, and for all bits b, if the network is in agreement on b at the start of the execution! it is in agreement on b at its end.*

Proof. Since each execution of subprotocols $\mathcal{P}_1$ and $\mathcal{P}_0$ is in essence a special execution of $\mathcal{P}_r$, it is sufficient to prove our claim with respect to this latter protocol. Assume that at the start of $\mathcal{P}_r$, the network is in agreement on 0; that is, every good dead player outputs 0 and. for all goocl alive players g. $b_g = 0$. Then all good players ("really" the alive ones and "virtually" the dead ones) distribute 0 in step 1. This implies that only the bad players can distribute *1* in step 1; thus in step 2, for all goocl alive g, $count_g < n/3$. As a consequence. independently of his own output of subprotocol OC, at the end of step *2,* each good alive player g sets $b_g := 0$ in accordance with instruction 2(a); that is, the network is in agreement on 0 at the end of $\mathcal{P}_r$. The case in which the network is in agreement on 1 at the start of $\mathcal{P}_r$ is handled similarly. ■

CLAIM *T4-3. In any execution of BA with a 113-adversary, whenever a good player outputs a bit, the network is in ugreement on that bit.*

Proof. A good processor generates an output only during the execution of either subprotocol $\mathcal{P}_1$ or subprotocol $\mathcal{P}_0$. Let E be the first execution of either $\mathcal{P}_1$ or $\mathcal{P}_0$ in which a good player produces an output, and let g be one such player. Assume that E is an execution of $\mathcal{P}_0$; then all good players are alive during E, and g must output 0 at E's end. Thus at E's end, $count_i \in [0, n/3)$. Therefore, by Claim T4-1, for all good G, $count_G \in [0, 2n/3)$. This entails that, because of either rule (a) or rule (b), every good player G resets $b_G := 0$; that is (because there are no dead good players). the network is in agreement on 0 at the end of E (though only those good players whose counter belongs to $[0, n/3)$ are aware of this and will thus output 0 and terminate). If there are no more executions of either $\mathcal{P}_1$ or $\mathcal{P}_0$ in which a good player outputs a bit, then we are done. Otherwise, because of Claini T4-2, since the network is in agreement on 0 at E's end, it will remain in agreement on 0 thereafter. Thus, whenever a good player outputs a bit later on, this bit must be 0, in accordance with our claim. The case in which E is an execution of $\mathcal{P}_1$ is argued in the "symmetric" way. ■

CLAIM *T4-4. For any random execution of BA with a* 1/3-*adversary and any positive integer k, if the network is not in agreement at the start of the kth execution of subprotocol* $\mathcal{P}_r$, *then the probability that it will be in agreement at its end is greater than* .35.

Proof. Because of Clairn T4-3, our hypothesis implies that all good players are alive throughout our execution of $\mathcal{P}_r$. Moreover, since by Claim T4-1 we have $|count_g - count_G| < n/3$ for any good players g and G, one of the following two cases must occur:

(0) $\forall good\ i$, $count_i \in [0, 2n/3)$, or

(1) $\forall good\ i$. $count_i \in (n/3, n]$.

When case (0) occurs, if the oblivious common coin is unanimously 0, then each good player i resets $b_i := 0$ (if $count_i \in [0, n/3]$ because of rule 2(a), if $count_i \in (n/3, 2n/3)$ because of rule 2(b)) and thus the network is in agreement on 0. Similarly, when case (1) occurs, if the oblivious common coin is unanimously *1,* then every good processor i resets $b_i := 1$ arid thus—since there are no dead good players to worry about—the

network is in agreement on 1. Since either case (0) or case (1) must occur, and since OC is an oblivious coin protocol with fairness .35, the probability that the network is in agreement at the end of a random execution of $\mathcal{P}_r$ (though the good players may not be "aware" of this event.) is greater than .35. ■

CLAIM T4-5. In *any* execution of BA with a 113-adversary, *if* at the beginning *of* an execution of subprotocol $\mathcal{P}_1$ (respectively, $\mathcal{P}_0$), *the* network is in agreement on 1 (respectively, 0), then one *round* after the end of the *subprotocol* execution, BA *terminates* and *the* output of every good *player* is 1 (respectively, 0).

Proof. Assume that the network is in agreement on 1 at the beginning of an execution of $\mathcal{P}_1$ (the "0 case" is similarly handled). Then all dead good processors have output 1 prior to the present execution of $\mathcal{P}_1$, and all alive good processors distribute 1 in the first step of the execution. Thus, since their tallies of 1 belong to the interval $(n/3, n]$, all good (and alive) processors will perform instruction 4(c) throughout this execution. Therefore, each one of them outputs 1 and will terminate in the next round, unless he will get corrupted in the next round, an event that cannot: in any case, change the output of the still uncorrupted players or the termination of the protocol since. as usual, BA ends when all good players have terminated. ■

It is now easy to complete the proof of Theorem 4; we start by proving our claim about *BA*'s round complexity and fault tolerance. $\mathcal{P}_r$ is a 36-round protocol, while $\mathcal{P}_0$ and $\mathcal{P}_1$ are both two-round protocols. Protocol BA iterates the ordered execution of $\mathcal{P}_r, \mathcal{P}_0$, and $\mathcal{P}_1$ until all good players terminate. Claim T4-4 guarantees that, no matter what the initial quantities and the strategy of a 1/3-adversary may be, in a random execution of BA, the probability that an "oblivious" agreement is not reached after the $2k$th execution of $\mathcal{P}_r$ is less than $(.35)^{2k} < 2^{-k}$. Once oblivious agreement is reached at the end of an execution of $\mathcal{P}_r$, Claim T4-5 guarantees that, no matter what the actions of the 113-adversary may be, protocol BA halts—with all the good processors "aware" of having reached Byzantine agreement—within the next five rounds (i.e., at most one round after the end of $\mathcal{P}_1$ if the agreement was on 1). Thus the probability that protocol BA does not reach Byzantine agreement within $80k + 5$ rounds is less than 2^{-k}.

Let us now prove our claim about, the amount of local computation of protocol BA. Having set (hidden) message bounds, the good processors do not waste running time reading excessively long messages sent by the adversary. Moreover, except for some occasional random selections; each round of protocol BA can be performed in fixed polynomial-in-n time. As for those raridom selections, they consist of the random choices of elements in integer intervals of the form $[0, z-1]$. Now, whenever z is a power of 2, a random selection in $[0, z-1]$ can be performed by flipping $\log z$ coins—and thus in fixed (as opposed to expected) polynomial time by a probabilistic Turing machine. However, if z is not a power of 2, then the adopted strategy for this task consists of randomly selecting a $\lceil \log z \rceil$-bit string until a member of the desired interval is found. Clearly, the probability that more than T such trials are needed is less than 2^{-T}. Since in each iteration of $\mathcal{P}_r, \mathcal{P}_0$, and $\mathcal{P}_1$, at most $Q(n)$ such selections (where Q is a given polynomial) must be rnade by the good players, since the rest of the computation can be performed in fixed polynomial time, and because of our recently proven claim about the round complexity of BA, our claim about the running time of BA easily follows. □

Remarks.

• Our reduction of Byzantine agreement to (oblivious) common coins was inspired by an earlier work of Rabin [34]. His reduction is much simpler, but it assumes a common coin that not only is *externally provided* but also is not oblivious (i.e., all

players are *guaranteed* to see the same common random bit[37]), and it requires that the number of faults is $< n/4$.

• As we have seen? protocol BA enjoys the property of being always *correct* and *probably* fast; that is. our use of prohabilisrn introduces some uncertainty of how long it will take to terminate (a modest uncertainty since we prove that the expected number of rounds is constant), but no possibility of error in the correctness of the final agreement. This desirable property implies that one cannot get rid of "expected" in our round complexity. In fact, an algorithm that reaches a guaranteed agreement in a fixed number of rounds, no matter what the sequence of its coin tosses may be, is immediately transformed to a fixed-round, deterministic algorithm. Thus the result of Fischer and Lynch [20] would imply that at least $O(n)$ rounds are needed if the number of possible faults is $O(n)$.

• In general, as we have said, the input of a processor is a *private* value; that is, the adversary has no way of knowing it unless she corrupts its corresponding processor or this processor is instructed by the protocol to divulge it. Privacy of the initial inputs is also a necessary condition for certain protocols to be meaningful. This is indeed the case, for instance, with protocol *GradedVSS*—indeed, unless the input of an honest dealer is secret, there is no hope that an aclversary cannot guess it better than at random. In the case of Byzantine agreement, on the other hand, the privacy of the initial inputs plays no role in defining the problem, which in fact remains totally meaningful even if we assume that the players' initial bits are known to the adversary.[38] Indeed, it should be noted that our protocol BA instructs each good player to distribute his input bit at the very first step, and it thus works even in the case in which the adversary knows the input bits of all players in 'advance.

• As we know, protocol BA relies on subprotocol OC. One may describe this subprotocol as producing a bit that is "sufficiently random and common." Such a description would, however, be quite incomplete.. Namely, the output of OC is also sufficiently *unpredictable* at the start of each execution of the protocol. In fact, if the fairness of the coin that OC produces is positive, then we know that in any random execution both 0 and 1 have a positive probability of being output. It should be noticed that this unpredictability is used in our Byzantine agreement protocol: in protocol $\mathcal{P}_r$, "the oblivious coin" is flipped after every processor i distributes his current value b_i; thus in step 1, the adversary must choose which values the bad players distribute when the oblivious coin is still unpredictable. Actually, the unpredictability of the oblivious common coin flip is more than merely *used* in our protocol; it is actually crucial to it: should the adversary know the result of the coin flips of OC in advance, she could prevent agreement indefinitely. In fact, a bit more precisely, it can be shown that if all processors have as a common input—at the beginning of the protocol—a sequence of *truly* random *and* independent (but also: necessarily, predictable) coin tosses and use these bits instead of the outcomes of OC in subprotocol $\mathcal{P}_r$, a 1/3-adversary can easily and indefinitely prevent agreement from being reached.

The above discussion can be summarized by saying that our protocol BA relies heavily on hiding—at least temporarily—information. We will further elaborate on this crucial point in section 9.2.

• As we have indicated, the good processors need not terminate simultaneously. Indeed, the adversary can force "staggered termination" if she so desires.

• To avoid staggered termination, one may consider iterating subprotocol $\mathcal{P}_r$ a

[37] In his scenario, random coin flips are "predistributed" by a trusted party. Thus once they are "revealed," all good processors will see the same result.

[38] This is a quite plausible scenario since bad guys tend to "know" more than good ones.

prescribed number of times. If this number of times is large enough. upon termination. agreement would be reached with high probability. However, such a protocol would be unsatisfactory. First. from a theoretical point of view. it would introduce a probability of error. (In other words, there would be a chance that upon termination the good processors may not be in agreement—an event that is not allowed by our definition.) Second, from a "practical" point of view, to ensure that agreement is reached with probability $1 - 2^{-k}$, the envisaged protocol would have *always* to run $\mathcal{P}_r$ k times. By comparison, our protocol will run $\mathcal{P}_r$ k times only "very seldomly," that is, with probability 2^{-k}. (Truly, each time that our protocol runs $\mathcal{P}_r$, it also runs $\mathcal{P}_0$ and $\mathcal{P}_1$, but these latter protocols require only two rounds each and are extremely simple. The brunt of the computation is constituted by $\mathcal{P}_r$ alone, which is a quite complex 34-round protocol.)

• It should be noticed that. since some good processors may be alive and some others may be dead, in some executions of $\mathcal{P}_r$, there may not be a $2/3$ majority of good processors. In fact, the dead ones do not participate in the protocol but simply "virtually" send a bit at given times. Under these circumstances. the coin tosses of OC need not to be common or fair in any way. This is not a problem, though. As we have shown, when a good processor terminates. the processors are in agreement and agreement cannot be disrupted. Protocol $\mathcal{P}_r$ is thus executed at most once without an honest majority of players: in fact, all alive good processors will terminate one round after the next execution of either $\mathcal{P}_0$ or $\mathcal{P}_1$, whose coin toss the adversary does not control.

9. Adjustments and improvements.

9.1. The model independence of our Byzantine agreement protocol.

Pros and cons of *standurd* networks. In presenting our Byzantine agreement protocol, following a time-honored tradition, we have chosen standard networks (i.e., networks in which every pair of processors is connected by a dedicated and private communication line) as its underlying communication model. This model has notably simplified our argument and has helped us to focus on the *essential* distributed aspects of the quintessentially distributed problem at hand without getting sidetracked by a variety of important but quite different issues. (Essence, of course, is in the eyes of the beholder!) Moreover, the standard-network model is quite realistic in some contexts—for instance, in the case of computer networks whose processors are not directly controlled by humans.[39] Unfortunately, this is also the context in which, in our opinion, Byzantine agreement is less meaningful, at least for the extremely malicious fault model addressed in this paper—which, regrettably, belongs to the domain of human interactions. As a matter of fact,, people being what they are, private channels may prove to be too much of an abstraction. If a Byzantine agreenient protocol were run in the context of an adversarial negotiation conducted in a computer network, it would be remarkable that impostors would chivalrously confine themselves to purely software attacks, refraining from tampering with the network itself. Indeed, if they did, communication channels would not remain "private" for too long, no matter how much metal they could be shielded with or how deep they could be routed. We thus wish to briefly discuss what happens to our algorithm when its communication model is more... "humanized."

[39] Indeed, when such computers malfunction, they may start running algorithms that are different from their intended ones, may act—due to Murphy's law—its if the); coordinate their disruptful efforts, and so on, but they cannot gain access to the dedicated line connecting two properly working processors!

Other possible models. If the adversary may prevent messages between good processors from being delivered. Byzantine agreement would be impossible. However, we may still trust our network to be asynchronous; that is, the adversary might delay messages arbitrarily long but cannot prevent them from everitually reaching their intentled recipients. (For a discussion of this model, consult. for instance, [19].) Fortunately. our Byzantine agreement protocol has been ingeniously extended by Feldman [17] and Canetti and Rabin [7] to work on asynchronous networks as well.

If the adversary is able to change the messages exchanged between two good processors. Byzantine agreement would also be impossible since a single faulty processor could impersonate as many processors as it likes. Alternatively, the adversary may be capable of reading messages between good processors but not altering them.[40] In either case. one can still run our protocol using cryptography to simulate the privacy of such "public" lines (assuming, of course, that the adversary is computationally bounded). The basic underlying idea is that injecting or altering messages may be made infeasible by secure digital signatures that are secure in the sense of [25], while reading messages can be made infeasible by an encryption scheme that is secure in the sense of [23]. (One caveat, however: for very subtle reasons that exceed the scope of this paper, this basic strategy is surprisingly hard to implement correctly.)

If the adversary can "disconnect" two good processors, Byzantine agreement would again be impossible. However, rather than assuming that our network is complete, we may trust that it has some special. uncorruptable nodes that do not perform any computation but simply reliably route properly labeled messages. (Indeed, this may allow for quite sparse networks.) In this setting, our protocol would work essentially without any changes and with the same efficiency. Alternatively, one may consider networks with fewer communication lines but with sufficiently high connectivity. This way. for every set of faulty processors with small enough cardinality, every two good processors are still connected by a path consistirig solely of good processors. Solving the problem in this new setting would require encrypting each message and sending it to its recipient through several node-disjoint paths. This, of course, would increase the running time of our protocol by a "network-topology" factor, but, most likely, the same increase in running time would be suffered by other protocols.

9.2. Improvements of our results. Our results have been found useful in several ways.

- As we have already mentioned in subsection 9.1, our Byzantine agreement protocol has been extended by Feldman [17] to work on asynchronous networks in which each pair of players is connected by a private channel. His asynchronous protocol tolerates up to $t < n/4$ faults. Using cryptography and assuming a computationally bounded adversary. it regains optimal fault tolerance, $t < n/3$, in the asynchronous case as well. Quite recently, Canetti and Rabin [7] have exhibited (for the same networks) an asynchronous Byzantine agreement protocol running in expected constant time and possessing resiliency 1/3 against an adversary with unbounded computational capabilities—though allowing a probability of error. (Let us note in passing that the notion of "constant time" must—and can—be meaningfully formulated in the asynchronous setting.)
- Ben-Or and El-Yaniv [4] have extended our algorithm to to reach Byzantine agreement in standard networks, in an expected constant number of rounds, for an entire collection of players' initial values.

[40] This may be the case in an ordinary telephone network, whose lines can be easily eavesdropped, while the voices of its users may be hard to imitate.

• Using our results and those of [4], Micali and Rabin [29] have obtained a VSS protocol (i.e., a .'nongraded one"!) that works in standard networks (rather than standard-plus-broadcast ones), runs in polynomial time and an expected constant number of rounds. and tolerates any $n/3$ faults in the worst model. They have also exhibited a *nonoblivious* common coin protocol, with fairness 1/2 and fault tolerance 1/3, that works in standard networks and runs in expected polynomial time and an expected constant number of rounds. (Dolev, Dwork. and Yung have informed them that they have independently found these same protocols.)

• Goldreich and Petrank [27] have shown how to modify our algorithm so as to keep its expected running time and round complexity, while guaranteeing termination in the worst case (i.e., with the most unlucky sequence of coin tosses) in $t + O(\log t)$ rounds whenever the upper bound on the number of faulty players is t. (Thus termination is guaranteed in $O(n)$ rounds in the worst-fault model.)

10. Significance.

10.1. The "right" significance of Byzantine agreement. Until now: we have been advocating that Byzantine agreement is "the best one can do, in an adversarial scenario. when broadcasting is impossible." At this point,, having gained more experience with adversarial behavior, we wish to point out. that this informal saying is misleading in that it seems to imply that broadcasting is an available resource? and only when you are deprived of it should you turn to Byzantine agreement as a meaningful substitute. The truth is that, in an adversarial setting, closer scrutiny reveals broadcasting to be "almost always impossible."

Consider, for instance, a radio network. The recipient of a message in such a network cannot. tell whether a satellite has aimed its signal to his specific geographical area. or to the whole country. Moreover, since imitating (or cutting and pasting recorded pieces of) one's voice is quite possible, the recipient of .a radio message cannot have any certainty about the identity of the sender of the message. Indeed, in an adversarial setting, broadcasting is an *abstraction.* Thus a natural question arises:

> In what "reasonable" communication models can one "concretely implement" an abstract notion satisfactorily close to that of broadcasting?

It is in light of this question that Byzantine agreement achieves, in our view, its true significance: namely, it demonstrates that standard networks offer a reasonable communication model to approximate: despite the presence of adversaries, the abstract notion of a broadcasting. Better said:

> We regard Byzantine agreement as showing that the abstraction of broadcasting can be meaningfully approximated by "simpler" abstractions: strong honest majority, synchrony. and private channels (and by even simpler ones: as we have discussed in section 9.1).

10.2. The significance of our results. It is now time to ask ourselves, "What is the significance of our own result,?"

While our simplest primitives—*Gradecast* and, for small n, *GradedVSS*—are quite practical, we do not expect. our Byzantine agreement protocol to have a direct practical impact,. In fact. though it does not have any monstrous "hidden constants" and is actually quite feasible. our protocol starts outperforming prior ones when run in standard networks (or networks with "simulated standardness," as discussed in section 9.1) with a few hundred players.[41]

[41] Should standard networks of this size become feasible. our result actually opens the possibility of *artificially increasing* the number of players so as to increase the reliability of the network without

However, our results should have an *indirect* practical impact. Solving a long-standing open problem always marks a technical advance in a given field, and it is reasonable to expect that in our case as well this increased level of understanding will eventually translate into more practical protocols than ours.

More importantly. our techniques will be quite effective when dealing with much more complex problems than Byzantine agreement, that is. with those problems for which the existence of any solution is by itself a blessing and no superpractical answer can be legitimately expected.[42] In fact, it should be appreciated that our protocol solves a more difficult problem than Byzantine agreement (a fact that may perhaps excuse some of our complications): it provides a reasonably fair and fault-tolerant coin-flipping protocol in a quite unmanageable communication arid fault model.[43]

Finally. *scientists shall not live by technique alone.* and we now wish to argue that our result is more significant from a purely conceptual point of view.

Probabilism versus determinism. Can randomness speed up computation? This is one of the most intriguing and fundamental questions of complexity theory. The celebrated probabilistic algorithms for primality *testing* of Solovay and Strassen [36] and Rabin [33] (and the more recent and equally beautiful ones for primality *proving* of Goldwasser and Kilian [22] and Adleman and Huang [1]) show that efficient probabilistic solutions exist for problems for which no polynoniial-time solution is yet known. We cannot, however. prove that no deterministic, polynomial-time primality algorithm exists. Indeed, the fact that generating a sequence of coin tosses, independently from the problem at hand, may help solve our problem much faster is quite puzzling.

From this point of view, our result takes on a more serious significance. Namely, contrasting its performance with the quoted $t + 1$-round lower bound [20] for any deterministic protocol in which t malicious faults may occur, our Byzantine agreement protocol offers a dramatic example that, at least in some scenarios. probabilistic solutions *are provably vastly superior* to all deterministic ones.

Such a speedup was already demonstrated by Rabin [34]. but by making the additional assumption of a common source of randomness *external to the network:* a common coin toss magically available to all processors at every clock tick. We instead demonstrate that randomness alone (i.e., individual and independent random choices made by individual processors), without any additional assumptions, suffices to beat any deterministic Byzantine agreement protocol in a dramatic way.

Privacy versus correctness. Our probabilistic solution to the synchronous Byzantine agreement problem sprung from recent advances in the field of *zero-knowledge computation.* Roughly said. this is the science of communication protocols that need to satisfy both a *correctness* and a *privacy* requirement. (For example. following the original application of Goldwasser, Micali, and Rackoff [24], a *zero-knowledge proof* shows that a given statement indeed possesses a correct proof but does not reveal what this proof might be.)

It should be noticed. however. that while Byzantine agreement has subtle correct-

---

making the time needed to reach agreement helplessly long. (In fact, if we know that—say—10% of the players are expected to become faulty during a decade, to ensure that 2/3 of them will be working properly in such a period, we are better off having a network of hundreds of processors rather than just a dozen of them.)

[42] Indeed. the usefulness of our algorithm for solving the problems mentioned in section 10.1 provides some support for this claim, and it augurs wonderfully for future ventures.

[43] Indeed, flipping a coin with adversaries does not get much easier even in friendlier scenarios than ours.

ness requirements, it has no constraints whatsoever about privacy.[44] Nonetheless. the correctness and speed of our protocol depend in a fundamental way on *GradedVSS*, a protocol where privacy *is* the central issue. We thus wish to advocate a novel role for privacy: namely. a tool for reaching correctness. This is less puzzling than it sounds. Our intuition behind it is simple:

> *Error in, computation can. be modeled as an adversary, and if your adversary "knows little," she can do little to disrupt your computation.*

Indeed, we believe that privacy will become a fundamental ingredient in the design of fault-tolerant protocols. Are we right? Time will tell. But may our journey be enjoyable in any case.

Acknowledgments. We are particularly grateful to Michael Fischer, Rosario Gennaro, Nancy Lynch, and David Shmoys for their generous, attentive, and constructive criticism.

Special thanks go to Ray Sidney, Tal Rabin, and Philip Rogaway. As we have already mentioned, the second author has collaborated with Philip Rogaway in modeling computation in the presence of faults in more complicated scenarios than the present one. The computational model of this paper has benefitted from the insights gained during that collaboration.

We would also like to acknowledge Michael Ben-Or, Benny Chor, Cynthia. Dwork, Peter Elias, Rosario Gennaro, Oded Goldreich, Shafi Goldwasser, and hlichael Rabin for many wonderful discussions about the Byzantine agreement problem.

Thanks also to two anonymous referees for their wonderful comments. The present version of our paper corresponds to the point in which one referee lamented that formalization exceeded intuition and another that intuition outmatched formalization.

Finally, our main motivation for working on the Byzantine agreement problem came from the *beauty* and *novelty* of the ideas of those who preceded us. We have immensely enjoyed standing on such tall shoulders!

REFERENCES

[1] L. M. ADLEMAN AND M. .4. HUANG, *Recognizing primes* in *random polynomial time*, in Proc. 19th ACM Symposium on Theory of Computing, ACM, New York; **1987.** pp. 462–469.

[2] D. BEAVER, S. MICALI. AND P. ROGAWAY, *The round complexity of secure protocols,* in Proc. 22th ACM Symposium on Theory of Computing, ACM. New York, **1990.**

[3] M. BEN-OR. S. GOLDWASSER. AND A. WIGDERSON, *Completeness theorems for fault-tolerant distributed computing*, in Proc. 20th ACM Symposium on Theory of Computing, ACM, New York. **1988.** pp. 1–10.

[4] M. BEN-OR AND R. EL-YANIV, *Interactive consistency* in *constant time*. Distrib. Comput., **1991.** submitted.

[5] M. BEN-OR, *Another advantage of free choice: Completely asynchronous agreement. protocols,* in Proc. 2nd Annual Symposium on Principles of Distributed Computing, ACM, New York, **1983,** pp. **27–30.**

[6] G. BRACHA. *An "o*(log *n*)" *expected rounds randomized Byzantine generals protocol,* in Proc. **17th** ACM Symposium on Theory of Computing. ACM, New York, **1985.**

[7] **R.** CANETTI AND T. RABIN, *Fast. asynchronous agreement with optimal resilience.* in Proc. 25th ACM Symposium on Theory of Computing, ACM, New York. 1993, pp. **42–51.**

[8] B. CHOR AND B. COAN, *A simple and efficient randomized Byzantine agreement problem,* TGEE Trans. Software Engrg., SE–I **1** (1985), pp. **531--539.**

[9] B. CHOR. S. GOLDWASSER. S. MICALI. AND B. AWERBUCH, *Verifiable secret sharing and achieving simultaneity in the presence of faults.* in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science. IEEE Computer Society Press. Los Alamitos. CA, 1985, pp. 383-395.

[44] Indeed, our protocol *BA* starts by having each good processor distribute his own input value to all players.

[10] D. CHAUM, C. CREPEAU, AND I. DAMGÄRD, *Multi-party unconditionally secure protocols,* in Proc. 20th ACXI Symposiuni on Theory of Computing, ACM, New York, **1988.**

[11] B. CHOR AND C. DWORK, *Randomization in Byzantine agreement,* in Randomness and Cornputation. S. Micali, ed., JAI Press, Greenwich. CT, **1989,** pp. 433–498.

[12] D. DOLEV, M. FISCHER, R. FOWLER, N. LYNCH, AND H. STRONG, *An efficient algorithm for Byzantine agreement without tauthentication,* Inform. and Control, 52 (1982), pp. 257–274.

[13] D. DOLEV, *The Byzantine generals strike ugain,* J. Algorithms, **3** (1982), pp. **14–30.**

[14] D. DOLEV AND C. DWORK, manuscript, **1987.**

[15] D. DOLEV, C. DWORK, AND M. NAOR, *Non-malleable cryptography,* in Proc. 23rd ACM Symposium on Theory of Computing, ACM, New York, **1993,** pp. 542–552.

[16] C. DWORK, D. SHMOYS, AND L. STOCKMEYER, *Flipping persuasively in constant expected time,* SIAM J. Comput., **19** (1990), pp. 472–499.

[17] P. FELDMAN, *Optimal algorithms for Byzantine agreement,* Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, ILIA, **1988.**

[18] P. FELDMAN AND S. MICALI, *Byzantine agreement in constant ezpected time (and trusting no one),* in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alatnitos, CA, **1985,** pp. **267–276.**

[19] M. FISCHER, *The consensus problem in unreliable distributed systems (a brief survey),* in Proc. International Conference on Foundations of Computation, **1983.**

[20] M. FISCHER AND N. LYNCH. A *lower bound for the time to assure interactive consistency,* Inform. Process. Lett., 14 (1982), pp. **183–186.**

[21] Z. GALIL. S. HABER, AND M. YUNG, *Cryptographic computation: Secure falt-tolerant protocols and public-key model,* in Proc. CRYPTO **'87,** Springer-Verlag, Berlin, **1987,** pp. **135–155.**

[22] S. GOLDWASSER **AND** J. KILIAN. *Almost all primes can be quickly certified,* in Proc; 18th ACM Symposium on Theory of Computing, ACM, New York. **1986,** pp. **316–329.**

[23] S. GOLDWASSER AND S. MICALI, *Probabilistic encryption,* J. Comput. System Sci., **28** (1984), pp. **270–299.**

[24] S. GOLDWASSER, S. MICALI, AND C. RACKOFF, *The knowledge complexity of interactive proof-systems.* SIAM. J. Comput., **18** (1989), pp. **186–208.**

[25] S. GOLDWASSER, S. MICALI, AND R. RIVEST, *A digitul signature scheme secure against adaptive chosen-message attacks,* SIAM J. Comput., **17** (1988), pp. 281–308.

[26] 0 . GOLDREICH. S. MICALI, AND A. WIGDERSON, *How to play any mental game, or a completeness theorem for protocols with honest majority,* in Proc. 19th ACM Symposium on Theory of Computing, ACM, New.York, **1987,** pp. **218–229.**

[27] O. GOLDREICH AND E. PETRANK, *The best of both worlds: Guaranteeing termination in fast randomized Byzantine agreement protocols,* Inform. Process. Lett., **36** (1990), pp. 45–49.

[28] A. KARLIN AND A. YAO, manuscript, **1987.**

[29] S. MICALI AND T. RABIN, *Collective coin tossing without assumptions nor broadcasting,* in Proc. CRYPTO **'90,** Springer-Verlag, Berlin, **1990,** pp. **253–266.**

[30] S. MICALI AND P. ROGAWAY, *Secure computation,* in Proc. CRYPTO '91, Springer-Verlag, Berlin, **1992;** full paper available from authors.

[31] Y. MOSES AND 0 . WAARTS, *Coordinated travel:* $(t+1)$*-round Byzantine agreement in polynomial time,* in Proc. *29th* Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press. Los Alamitos, CA, **1988.** pp. 246–255.

[32] XI. PEASE. R. SHOSTAK, AND L. LAMPORT, *Reaching agreement in the presence of faults,* J. Assoc. Comput. Mach., **27** (1980), pp. **228–234.**

[33] M. RABIN, *Probabilistic algorithms for testing primnlity,* J. Nuniber Theory, **12** (1980), pp. 128–**138.**

[34] M. RABIN, *Randomized Byzantine generals.* in Proc. *24th* Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, **1983,** pp. 403–409.

[35] T. RABIN AND M. REX-OR, *Verifiable secret sharing and multiparty protocols with honest majority,* in Proc. **21th** ACM Symposium on Theory of Computing, ACM, New York, **1989.**

[36] R. SOLOVAY AND V. STRASSEN, A *fast Monte-Carlo test for primality.* SIAM J. Comput., **6 (1977).** pp. **84–85.**

[37] R. TURPIN AND B. COAN, *Extending binary Byzantine agreement to multivalued Byzantine agreement,* Inform. Process. Lett., **18** (1984), pp. **73–76.**