Simple Constant-Time Consensus Protocols in Realistic Failure Models

BENNY CHOR

Technion, Haifa, Israel

MICHAEL MERRITT

Massachusetts Institute of Technology, Cambridge, Massachusetts and AT&T Bell Laboratories, Murray Hill, New Jersey

AND

DAVID B. SHMOYS

Massachusetts Institute of Technology, Cambridge, Massachusetts

Abstract. Using simple protocols, it is shown how to achieve consensus in constant expected time, within a variety of fail-stop and omission failure models. Significantly, the strongest models considered are completely asynchronous. All of the results are based on distributively flipping a coin, which is usable by a significant majority of the processors. Finally, a nearly matching lower bound is also given for randomized protocols for consensus.

Categories and Subject Descriptors: C.2.2 **[Computer-Communication Network]**: Network Protocols; C.2.4 **[Computer-Communication Network]**: Distributed Systems; F.2.2 **[Analysis of Algorithms and Problem Complexity]**: Nonnumerical Algorithms and Problems

General Terms: Algorithms, Performance, Reliability, Theory

Additional Key Words and Phrases: Agreement problem, consensus problem, cryptography, fault tolerance

1. *Introduction*

Randomization has proven to be an extremely useful tool in the design of protocols for distributed agreement. In this paper we present new randomized protocols for the consensus problem in synchronous and asynchronous fail-stop and

Much of B. Chor's research was done while at MIT, supported in part by an IBM graduate fellowship. Much of M. Merritt's research was done at MIT, supported in part by the Office of Naval Research under N00014-85-K-0168, by OAR under DAAG 29-84-K-0058, by the National Science Foundation (NSF) under DCR 83-02391, and by DARPA under N0014-83-K-0125. The research of D. B. Shmoys was supported in part by the NSF under DCR 83-02385 and by the Air Force under AFOSR 86-0078.

Authors' present addresses: B. Chor, Technion, Haifa, Israel; M. Merritt, AT&T Bell Laboratories, Murray Hill, NJ 07974; D. B. Shmoys, Room 2-376, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1989 ACM 0004-5411/89/0700-0591 $01.50

failure-by-omission models. These protocols all terminate within constant expected time, and unlike previous fast protocols, are very simple and need not rely on any preprocessing. In fact, we believe that these protocols will be the method of choice in practical implementations. The major novelty of our algorithms is the notion of a weak form of a global coin, and a method for generating it.

We define the *consensus problem* as follows: processor i has a private binary value v_i; at the termination of the protocol all processors have agreed on a common value v; if all v_i were equal initially, the final value agreed upon is this common value.

We shall initially consider the following synchronous model. We are given a system of n processors that can communicate through a completely connected network. The processors act synchronously, where at each time step each processor can broadcast a message, receive all incoming messages, and perform some private computation (possibly involving coin tossing). In the absence of failure, any message sent at time i will be received at time $i + 1$. As a result, we view the computation as occurring in *rounds*, each consisting of transmission, reception, and private computation phases.

The situation for deterministic algorithms for consensus is well understood. A result of Dolev and Strong implies that in a synchronous fail-stop model, at least $t + 1$ rounds are needed, in the worst case, to achieve consensus; they also provided an algorithm that achieves this bound and transmits only a polynomial number of messages [10]. In an asynchronous model, Fischer et al. showed that no protocol exists for consensus in the fail-stop model that tolerates even a single fault [14].

Fortunately, randomization can overcome this inherent intractability. Ben-Or describes a protocol for asynchronous consensus that tolerates up to $n/2$ faults in the fail-stop model, and terminates with probability 1 [4]. Results of a similar nature were given by Bracha and Toueg [6]. However, the expected number of rounds needed to reach agreement as measured locally by every processor is exponential in the asynchronous case (and can be shown to be $O(t/\sqrt{n})$ in the synchronous one). Rabin introduced the important notion of a *global* coin flip [22], which is a coin flip whose outcome is visible to all processors. He describes a different protocol that employs such a coin, so that each processor can use the outcome of a common coin. The expected number of rounds to reach agreement is $O(T(n))$, where $T(n)$ is the time required to flip the coin in a network of n processors. In order to implement his global coin, Rabin required some predealt information to be distributed by a trusted third party. Bracha, using a beautiful "boot-strapping" construction, showed that Rabin's result could be improved so that agreement is reached in $O(T(\log n))$ expected number of rounds (that is, the time to flip many independent coins in subnetworks whose size is logarithmic in n, in the size of the original network) [5]. It has been shown how to use cryptographic techniques to implement such a coin-toss in $T(n) = O(n)$ rounds, so that overall, Bracha's procedure can be run in $O(\log n)$ expected time as shown in [3] and by (A. C. Yao, private communication). However, this scheme requires an assignment of processors to committees for which no explicit construction is known. In contrast, our protocol is completely constructive. Feldman and Micali [12] have also eliminated the nonconstructive part of Bracha's probabilistic assignment, by having the processors generate the assignment themselves. However, in the process, Feldman and Micali introduced a preprocessing phase that requires $O(t)$ rounds. Their protocol is superior to deterministic protocols in an amortized sense, since additional agreements require only $O(1)$ time. The best-known bound for a Byzantine fault model without predealt information or preprocessing is $O(\log n)$.

Since our algorithms for omission faults run in constant expected time, current results leave a log n separation between the Byzantine and omission fault models.[1]

Throughout this paper, n will be used to denote the number of processors, and t will denote an upper bound on the number of failures tolerated. We present protocols for achieving consensus in completely connected networks despite omission faults of various types. The protocols can tolerate up to a constant fraction of the processors failing: that is, for each protocol and fault type there is a constant $\beta < \frac{1}{2}$, independent of the value of n, such that the protocol can tolerate as many as $t = \beta n$ omission faults of the given type.

The algorithms that we present are based on producing a coin flip that is essentially global. (A global coin flip has a random outcome that is viewed identically by every processor.) We relax the condition that each processor's view of the coin must always be identical, and in fact, the coin may even be somewhat biased.

Definition. A coin is called *weakly global* if there exists an absolute constant $c > 0$, such that for all $v \in \{0, 1\}$, the probability that at least $\min\{\lfloor n/2 \rfloor + t + 1, n\}$ processors all see outcome v is at least c.

The intuition behind this definition is that if $\lfloor n/2 \rfloor + t + 1$ processors see the same outcome, then a majority of the processors ($\lfloor n/2 \rfloor + 1$) will use this value in the consensus protocol, and reach consensus in a few more rounds. The essence of our weakly global coin procedure is to randomly select a temporary leader, and then to use the leader's local coin flip for the given round. After showing how such a coin can be produced in a variety of omission fault models, we then indicate how to use it to achieve consensus.

The design strategy of our protocols reflects a heuristic rule prevalent in distributed protocol design: It should be possible for simpler algorithms to defeat weaker adversaries. In the search for provably good algorithms that are also useful in practice, this rule suggests that some complex protocols have simple counterparts in more realistic fault models. In the case studied here, the algorithm against the adaptive adversary is transparent in comparison to the protocol for the Byzantine case that results from the combined work in [3] and [5].

Finally, we show that these results are nearly tight, by showing that for any protocol for the synchronous fail-stop model, if t processor faults are tolerated, then the probability that all correct processors have decided after k rounds ($k \le t$) is at most $1 - \frac{1}{2} \cdot (t/(2nk))^k$. (The same result was obtained independently by Karlin and Yao [17].) By comparison, the probability achieved by our protocol is $1 - (c + t/(2en))^{k/2}$ where the constant $c = (2e - 1)/(2e)$.

2. *Failure Models*

Correctness proofs for fault-tolerant algorithms have a game-theoretic character. They argue that the algorithms behave appropriately, even when the faults are being caused by an intelligent adversary. The capabilities attributed to this adversary have a profound effect on the design of algorithms meant to defeat it. Indeed, there are cases in which no algorithm is capable of defeating sufficiently powerful adversaries [14, 20].

[1] Since the preliminary version of this work was published [8], there have been significant advances in protocols for Byzantine agreement. Building on our notion of random elections, Dwork et al. [11] give an $O(\log \log n)$ expected-time Byzantine agreement protocol tolerating $t < n/4$ failures, and Feldman and Micali [12] improve this to give constant expected time protocols tolerating $t < n/2$ failures.

In Byzantine fault models, the adversary can control the behavior of some processors, causing them to send arbitrary messages whenever it likes. Such an adversary is extremely powerful, and defeating it seems to require complex and expensive algorithms. If one is modeling physical failures (as opposed to intentional attacks), such an adversary may be unrealistically powerful.

Consider the following example. On October 27, 1980, the ARPANET suffered a catastrophic failure as the result of hardware failures in two processors. Two spurious messages were generated that brought down the whole network for a period of several hours. Clearly, the network protocols were not capable of surviving even a small number of Byzantine faults. Instead of changing the protocols, hardware error-detection was added in the next generation processors, reducing the likelihood of repetition of this Byzantine failure to an extremely small probability [23]. Rather than implementing protocols to defeat a Byzantine adversary, the network designers effectively chose to weaken the adversary.

The new ARPANET implementation might be best described by an omission fault model, in which processors never send spurious messages, but some messages may fail to arrive at their destination. The adversary is thus limited to specifying which messages will be delivered to their destination, and which will not. The failure models we consider here are variants of failure by omission.

For deterministic protocols, an adversary, causing failures to produce the worst possible performance, can determine the outcome of a strategy in advance. With randomization, this is no longer possible, so that it may be advantageous for the adversary to decide its strategy adaptively, as random bits are generated and used. Therefore, in modeling the power of the adversary, it is crucial to specify the extent to which the adversary is adaptive, and the information it has available to determine its strategy. We consider three limitations on the adaptiveness of the adversary. Each of these is concerned solely with the communication system that connects the processors, and thus assumes that the processors are themselves nonfaulty. However, as we elaborate below, the situation in which processors are allowed to fail in a "fail-stop" manner is a special case of one of our models.

Static Faults. Throughout the life of the system, messages sent by at most t processors fail to reach their destination on time (within the round they are sent). Most previous work on omission fault models has focused on this type of fault. In the traditional fail-stop model, processors fail by halting prematurely, but the communication network always delivers all messages that have been sent. Within this model, our definition of the consensus problem is flawed, since we require that *all* processors agree on a value, and it is hopeless to require a faulty processor to do anything. If we relax this requirement to *all nonfaulty* processors, it is not hard to see that static communication faults include the case of fail-stop processor faults.

Dynamic-Broadcast. During each round, messages sent by at most t processors fail to reach their destination (but this may happen to a different set of t processors each round). A processor that sends a message that does not reach its destination is said to be *erratic.* These models are more general than static fault models. They are similar to models studied in [21].

Dynamic-Reception. Each processor receives all but at most t messages sent to it during every round (so that, if all processors are supposed to broadcast every round, each processor receives at least $n - t$ messages). However, any two processors may fail to hear from a different set of t others. These models are more general than dynamic-broadcast models, and are similar to the models we use for the asynchronous case.

We present algorithms for dynamic-broadcast and dynamic-reception models. Because these models are more general than the fail-stop or static models, our algorithms will work in these cases as well.

In addition to the limitations on the adaptiveness of the adversary mentioned above, we consider two different limitations on the knowledge available to the adversary in determining its strategy.

Message-Oblivious. The adversary's choice of failure, that is, which messages will not be delivered, is independent of the contents of the messages. However, this choice can depend, for example, on the pattern of communication or on the length of messages. Before giving a more precise definition, we first introduce a formal description of a synchronous execution of a protocol in this model.

At round $k + 1$ of a protocol, the prior k rounds of execution can be described in the following way. Consider a layered, directed graph consisting of $k + 1$ vertices for each processor p, (p, i), $i = 1, \dots, k + 1$, where there is an edge from (p, i) to $(q, i + 1)$ whenever p sends a message to q at round i. A subgraph of this graph represents the messages actually delivered. These graphs will be known as the transmission and reception graphs, and together will be referred to as the communication pattern. To complete the description of the prior execution, we add labels to the edges of the distribution graph, where the labels correspond to the contents of the messages. We define the ith layer of these graphs to be the subgraphs induced on the vertices with second coordinate i and $i + 1$.

Each processor p's view of the communication pattern consists of the subgraphs of nodes labeled by p, together with the labeled out-edges of those nodes in the transmission graph (the messages p sent), and the in-edges in the reception graph (the messages p received). A protocol for p determines a distribution of a new local state, out-edges and labels for node $(p, k + 1)$, as a function of p's local state and p's view of the first k layers of the communication pattern, together with p's input value. An adversary determines a distribution of in-edges for the $k + 1$st layer of the reception graph as a function of the n processor protocols and input values, the first k layers of the communication pattern, and the $k + 1$st layer of the transmission graph. An adversary is message-oblivious if for any given input vector to the processors, any communication pattern up to round k, and any kth layer of the transmission graph, the probability distribution of the kth layer of the reception graph is independent of the labels of the communication pattern through the first k layers (inclusive).

In [6], a weaker probabilistic adversary was considered, called a *fair scheduler*. At round i, a fair scheduler delivers to processor p a random subset with $n - t$ messages out of all messages sent to processor p at this round. Furthermore, the sets of messages sent to different processors are mutually independent. Bracha and Toueg have demonstrated a constant expected run-time with constant fraction of failures for executions under fair schedulers.[2]

[2] It is worthwhile to clarify the relation between our message-oblivious adversaries and Bracha and Toueg fair schedulers. It is clear that every fair scheduler is also message oblivious: The identity of messages it delivers to processor p at round i certainly does not depend on the content of the messages, as it is chosen at random. However, the class of message-oblivious schedulers strictly contains the class of fair schedulers. As an example, consider a scheduler that delivers, at every round, the messages of processors $1, 2, \dots, n - t$ to processors $1, 2, \dots, n/2$, and the messages of processors $t + 1, t + 2, \dots, n$ to processors $n/2 + 1, n/2 + 2, \dots, n$. Such scheduler is certainly message oblivious, but not fair. It is not hard to see that when applying this scheduler to the (deterministic) algorithm of [6], the system stabilizes at a nonterminating state if the initial configuration is that processors $1, 2, \dots, n/2$ have input 0 and processors $n/2 + 1, n/2 + 2, \dots, n$ have input 1 (whereas our algorithm will terminate in constant expected time). Whether message-oblivious adversaries or fair schedulers are better models of real-life systems is outside the scope of this paper.

The second model places fewer restrictions on the adversary's knowledge of communication in the network.

Message-Dependent. The adversary is limited to polynomial resources (time and space), but its choice of failures may depend on the contents of the messages.

Note that our definitions assume that the adversary has full knowledge of the hardware and software running at each processor and of the communication over the network (subject to the limitations above), but does not know the local state of the individual processors during execution (which may depend on the outcome of local coin tosses not observed by the adversary). For example, it will be important that decryption keys are stored in local memory and are part of the local state. We assume that the initial values can be seen by the adversary. For each combination of adaptiveness and knowledge constraints, we present an algorithm to achieve consensus in constant expected time.

3. *The Message-Oblivious Case*

In this section we show how to toss a weakly global coin in message-oblivious models. For the dynamic-broadcast failure model, the coin will have the property that for each outcome (heads or tails), there is some constant probability of that outcome being received by *every* processor. For the dynamic-reception failure model, there is some constant probability that for each outcome, at least $\lfloor n/2 \rfloor + t + 1$ processors will receive that outcome, provided t is bounded away from $n/4$.

The algorithm is perhaps the most natural one. A leader randomly volunteers, and this leader tosses a coin. More precisely, consider the following algorithm: the procedure LEADER produces a local biased bit where the probability of a 1 ("I volunteer") is equal to $1/n$; the procedure RANDOM BIT produces a local unbiased bit.

```
Code for processor $P$:
1. function COIN TOSS$_1$:
2. $l_p \leftarrow$ LEADER
3. $c_p \leftarrow$ RANDOM BIT
4. broadcast $(c_p, l_p)$
5. receive all $(c, l)$ messages
6. if all messages received with $l = 1$ have the same $c$
7.    then COIN TOSS$_1 \leftarrow c$ of these messages
8.    else COIN TOSS$_1 \leftarrow$ local coin toss
```

THEOREM 1. *The function COIN TOSS$_1$ produces a weakly global coin in the dynamic-broadcast message-oblivious fault model, where the constant probability for either common outcome is at least* $(1 - \beta)/2e$, *provided* $t \leq \beta n$ (*where* β *is any constant less than* 1).

PROOF. In a single execution of COIN TOSS$_1$, the probability that exactly one processor volunteers is

$$\binom{n}{1} \cdot \frac{1}{n}\left(1 - \frac{1}{n}\right)^{n-1} = \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{1}{e}.$$

In the analysis, we restrict attention to executions of the procedure when this event happens. How can the adversary thwart a good coin toss? Only by preventing the leader's message from getting to all other processors. However, he must select the set of at most t erratic processors with no information about the random bits, so that if the leader is not among those picked by the adversary, its messages will reach all processors. Except for the contents of the messages, all processors actions are identical. Hence, all the messages of the leader reach their destination with probability at least $(n - t)/n \geq 1 - \beta$. The second coin toss of the leader was made independently of the above conditions, and the probability of each outcome is the same. Putting the pieces together, we get the claimed bounds. □

Remark. While it suffices to require $\beta < 1$ in order to achieve a weakly global coin, we actually will require $\beta \leq \frac{1}{2}$. This requirement is needed in the consensus protocol (see Section 6).

The protocol can also be viewed in the following way. The tossing of the $1/n$ biased coin is an approach to obtain a distribution where the maximum of n trials is likely to be unique. In this context, the leader is the processor who tossed the unique maximum. All processors receive the other processors' values, determine the maximum and hence the leader, and choose the unbiased bit of this processor. By choosing other distributions, it is easy to see that the probability of a unique leader can be pushed arbitrarily close to 1. In implementing the protocol, this means that it is possible to trade off additional bits transmitted in order to reduce the expected number of rounds to reach consensus. For example, if the leader identification consists of $3 \log n$ unbiased bits instead of a single bit l, there is a very high probability, $\geq 1 - 1/2n$, that the maximum of n bit-sequences will be unique.

THEOREM 2. *The function COIN TOSS$_1$ produces a weakly global coin in the dynamic-reception message-oblivious fault model, when $t \leq n(1/4 - \epsilon)$ for some constant $\epsilon > 0$. If $t = n(1/4 - \epsilon)$, the probability for either common outcome is at least $\alpha/2e$, where $\alpha = 8\epsilon/(4\epsilon + 5)$.*

PROOF. Once again, we focus on the case that exactly one processor volunteers, which happens with probability at least $1/e$. Recall that in the dynamic-reception fault model, every processor receives at least $n - t$ different messages each round, but different processors may receive messages from different sets of senders. Recall also, that the conditions for a weakly global coin only require that at least $\lfloor n/2 \rfloor + t + 1$ processors agree on the outcome of the coin. This happens if the leader succeeds in reaching this many processors. Accordingly, call processors whose messages are received by $\lfloor n/2 \rfloor + t + 1$ processors *persuasive*. Since the failures are chosen independently of the identity of the leader, it is sufficient to show that the fraction, α, of the processors that are persuasive during each round is at least a constant independent of n.

We next bound α using a simple combinatorial argument. Since everyone receives at least $n - t$ messages, the number of messages received is at least $n(n - t)$. Assume exactly αn processors are persuasive—then the number of messages received is at most $\alpha n^2 + (n - \alpha n)(\lfloor n/2 \rfloor + t)$. This number is achieved if each persuasive processor has n of its messages received, and the rest lack only one message received to be persuasive themselves. From $n(n - t) \leq \alpha n^2 + (n - \alpha n)(\lfloor n/2 \rfloor + t)$, we derive $(\lceil n/2 \rceil - 2t)/(\lceil n/2 \rceil - t) \leq \alpha$. If $t = n(\frac{1}{4} - \epsilon)$, α is at least $8\epsilon/(4\epsilon + 5)$.

Thus there is at least probability $1/e$ that there is a single leader, for each value (heads or tails), there is probability $\frac{1}{2}$ that the leader's other coin has that value, and there is at least α probability that the leader is persuasive. By the message-oblivious assumption, all these events are independent so that overall the probability of each outcome is at least $\alpha/2e$. □

By modifying the protocol, it is possible to significantly strengthen the number of faults tolerated in the dynamic-reception fault model. Before giving this new protocol, we first describe a basic building block that will be useful in several constructions.

3.1 SIMULATING DYNAMIC-BROADCASTS WITHIN A DYNAMIC-RECEPTION MODEL. We shall show that three rounds of broadcasting within the synchronous dynamic-reception model can simulate one round of a synchronous dynamic-broadcast while maintaining the property of message-obliviousness. The simulation consists of one round in which each processor broadcasts the original desired message for dynamic-broadcast. (To simplify the discussion, we assume every processor has such a message to send.) In the following two rounds, every processor sends his message plus his view of every other processor's message.

We begin by showing that after executing this protocol, there is a set of at least $n - t$ processors whose message has been relayed to all n processors, assuming that $t < n/2$. This is done by a simple counting argument. Consider the second round of the simulation. We show now that there must be at least one processor p whose second round messages reach $t + 1$ processors. If all processors reach no more than t, then M, the total number of messages successfully transmitted in the second round, is at most $M \leq nt$. But each processor receives at least $n - t$, so that $n(n - t) \leq M$. Thus we get $n(n - t) \leq nt$, contradicting the assumption that $t < n/2$. Every processor receives at least $n - t$ messages in each round, so that processor p must have attempted to relay at least this many messages to each processor in round two. Since there are $t + 1$ processors that have been relayed these messages at the end of round two (from p), every processor will be relayed these messages from one of the $t + 1$ processors by the end of round three. This proves that this three round dynamic-reception simulation gives us the structure of one round of dynamic-broadcast. It is not hard to see that one fewer round of echoing is not sufficient to guarantee the structure of a dynamic-broadcast round.

We now show that message-obliviousness is preserved by this simulation. First notice that the pattern of sending and receiving messages in the simulation itself does not depend on message contents. From the definition of a message-oblivious adversary, the ith layer of the reception graph is independent of the labeling of the transmission graph given the pattern of communication up to this point. The analogous statement holds for the $i + 1$st layer, given the ith layer and the previous pattern of communication. From the definition of conditional probability, we get that the probability of any communication for both the ith and $i + 1$st layers is independent of the previous labelings of the pattern of communication. In our protocol, this implies that the set of at least $n - t$ processors that reach at least $t + 1$ processors two rounds later, is independent of the contents of the messages sent. Once this set reaches $t + 1$ processors, the adversary cannot stop the set of messages from reaching all n processors in the next round. Since the set is independent of the contents of the messages sent, the pattern of the successful transmissions in the simulated dynamic broadcast is independent of the contents of the messages. Thus we have shown that message-obliviousness is preserved.

The above *two-round echoing* scheme is a general tool. Applying it for the case of producing a weakly global coin, we get the following modified procedure.

Code for processor P with two-round echoing:

```
1. **function** COIN TOSS$_2$:
2. $l_p \leftarrow$ LEADER
3. $c_p \leftarrow$ RANDOM BIT
```

```
4. broadcast $(c_p, l_p)$
5. receive all $(c, l)$ messages
6. broadcast $(c_p, l_p)$ and all $(c_i, l_i)$ pairs received
7. receive all compound $(c_1, l_1), \ldots, (c_n, l_n)$ messages
8. broadcast $(c_p, l_p)$ and all $(c_i, l_i)$ pairs received
9. receive all compound $(c_1, l_1), \ldots, (c_n, l_n)$ messages
10. if all messages received with $l = 1$ have the same $c$
11.     then COIN TOSS$_2$ $\leftarrow c$ of these messages
12.     else COIN TOSS$_2$ $\leftarrow$ local coin toss
```

Although the echoing in this protocol requires a factor of n more bits to be transmitted, it can tolerate up to $t = \lceil n/2 \rceil - 1$ failures and the fraction of processors whose messages reach everyone is at least $(n - t)/n$.

Summarizing the above discussion, we have obtained the following result:

THEOREM 3. *The function COIN TOSS$_2$ produces a weakly global coin in the dynamic-reception message-oblivious fault model, when $t < n/2$. The probability for either common outcome is at least* $(1 - (t/n))/2e$.

It is critical to the correctness of this protocol that the adversary's choice of messages delivered each round be independent of the contents of the messages. A stronger adaptive adversary might simply check each message as it is sent; if the processor is a potential leader (its message is $(b, 1)$), then the adversary blocks the message. This stronger adversary can also be defeated, as long as the contents of the messages are unintelligible to him. In this case, any attempt at blocking the leader's message is still an essentially random act, because the adversary cannot understand the messages. This suggests that encryption could be a useful tool in designing a protocol that can defeat a more powerful adversary.

4. *The Message-Dependent Case*

In this section we show how cryptographic techniques can be used to toss a weakly global coin in the presence of an adaptive adversary using a message-dependent strategy. We prove that if the adversary can block the weakly global coin, then it can break the cryptosystem. Therefore, if we assume that the cryptosystem is secure, and that the adversary is limited to polynomial computing resources, then it cannot prevent consensus within constant expected time.

Let E be a probabilistic encryption scheme that hides one bit [15]. We briefly describe the properties that E should possess for our purposes. Given a natural number h, the security parameter, E maps the bit 1 at random into a string o in a set $O \subset \{0, 1\}^h$ and maps the bit 0 at random into a string z in a set $Z \subset \{0, 1\}^h$. Given a random string $r \in O \cup Z$, we assume that no polynomial-time algorithm (that is, polynomial in h) can distinguish the case $r \in O$ from $r \in Z$, with success probability greater than $(1/2) + (1/n^c)$ for any constant $c > 0$. On the other hand, there is a polynomial-time algorithm that, given additional secret information, distinguishes between the two cases with probability 1. The scheme E can be based on any trapdoor function [23]. In particular, the familiar RSA cryptosystem can be used, with 0 encrypted by $E(x)$, where x is chosen at random among all numbers in Z_N with least significant bit 0, and 1 encrypted by $E(x)$, where x is chosen at random among all numbers in Z_N with least significant bit 1 [1]. (For this example, we assume that RSA is hard to invert.) It is important to reiterate that the main theorem of this section is based on the following hypothesis:

(*) The encryption function E cannot be inverted in random polynomial time without the secret trapdoor information.

We first make the assumption that all processors use the same public key E whose decryption key they all hold (but to which the adversary has no access). At the end of this section we indicate how this assumption can be removed, at some expense in the number of faults tolerated.

The only modification to the algorithm of the previous section is to replace the broadcasting of (c, l) (line 4 of the COIN TOSS$_1$ function) by the broadcasting of $(E(c), E(l))$. The modified code is now:

```
Code for processor $P$:
1. function COIN TOSS$_3$:
2. $l_p \leftarrow$ LEADER
3. $c_p \leftarrow$ RANDOM BIT
4. broadcast $(E(c_p), E(l_p))$
5. receive and decrypt all $(c, l)$ messages
6. if all messages received with $l = 1$ have the same $c$
7.    then COIN TOSS$_3 \leftarrow c$ of these messages
8.    else COIN TOSS$_3 \leftarrow$ local coin toss
```

We now prove that the new protocol is as hard to break as the cryptosystem it uses.

THEOREM 4. *Under the assumption* $(*)$, *if all processors hold the same encryption and decryption key, then for polynomially many repeated calls of the function COIN TOSS*$_3$, *each call produces a weakly global coin in the message-dependent fault models. This procedure is correct, provided* $t \leq \beta n$, *where* β *is any constant less than* 1 *for the static and dynamic-broadcast case, and* $t < n/2$ *for the dynamic-reception case. The probabilities of each outcome are as in Theorems* 1, 2, *and* 3, *respectively.*

PROOF. We first prove the result for the static and dynamic-broadcast model. We again restrict attention to the event that exactly one processor volunteers, and that its random bit is 1. (The case of 0 is handled identically.) This event occurs with probability at least $1/2e$. Suppose the adversary can block some of the messages of the leader with probability $\geq t/n + \epsilon$ (where $\epsilon > 0$ is a constant). We show that in fact the adversary has the power to *distinguish* between the encryption of (RANDOM BIT, 0) and the encryption of (1, 1). (An adversary can distinguish between these two events if there exists a constant $\epsilon > 0$ such that the difference between the probability that the adversary outputs a "1" on the encryption of (RANDOM BIT, 0) differs by at least ϵ from the probability that he outputs "1" on input the encryption of (1, 1).) Using a theorem of Goldwasser and Micali [15], this leads to a polynomial-time algorithm for the adversary to invert E.

The proof consists of two parts. The first part is to show that polynomially many previous executions of the COIN TOSS$_3$ give the adversary no information that it cannot get by itself. In other words, the polynomial-bounded adversary can simulate polynomially many executions of this protocol by itself, without having any secret information, with *exactly* the same probability distribution as occurred in the "real" executions. This claim follows from the fact that all the processors are doing in every round of the protocol is to pick $2n$ bits according to a known (easily computable) probability distribution, encrypt these bits under the public-key E, and then use the encrypted values in the underlying agreement algorithm (which is known to the adversary). To simulate one round of this protocol, the adversary simply picks $2n$ bits according to the probability distribution, encrypts them under E, and with these inputs, simulates the rounds of message exchange as it would in an actual execution. (A repetition of the argument would work for any polynomial number of rounds k.) Whenever the protocol calls for a processor to make an action based on the decrypted messages, the adversary consults the original bits (which it generated earlier) and uses them.

Now suppose the adversary can cause any event to occur with respect to the message distribution in the $k + 1$st execution of the protocol, based on the first k executions. (Since the adversary is restricted to polynomial time, it suffices to consider k, which is polynomial in n and the security parameter h.) In that case, the adversary could also simulate exactly the same distribution of messages in the first k rounds by himself, without invoking the actual processors, and then cause that event to occur right at the first execution of the COIN TOSS$_3$ protocol. (For a stronger definition of protocols that release no extra knowledge, and detailed discussion, see [16].)

In the second part of the proof we show by a simulation that subroutine COIN TOSS$_3$ produces a weakly global coin. As before, we condition on the event that there is a unique leader. (Recall that this happens with probability at least $1/e$.) Suppose the adversary can make the leader erratic with probability $\geq t/n + \epsilon$. We partition the event, "a unique leader exists" into the two disjoint events, A = "a unique leader exists and his coin toss is 1" and B = "a unique leader exists and his coin toss is 0". To make the leader erratic with probability $\geq t/n + \epsilon$, the adversary must succeed in making the leader erratic with probability $\geq t/n + \epsilon$ in at least one of the events A or B. In the remainder of the proof, we condition on the event A (the proof for event B is identical).

To succeed in blocking the leader's message in A means that the adversary implements a blocking algorithm $\mathscr{BL}$ that, given as inputs n encrypted pairs

$$(E(c_1), E(l_1)), \ldots, (E(c_n), E(l_n)),$$

(where $n - 1$ of the l_is are 0, and their corresponding c_is are random bits, exactly one l_i is 1, and the c_i corresponding to this i is 1), outputs $n - t$ pairs, that contain the $(E(1), E(1))$ with probability no greater than $(n - t)/n - \epsilon$. (The messages that $\mathscr{BL}$ outputs correspond to the processors that are not erratic, while the blocked ones are those originating in processors that are made erratic.)

We build a *distinguisher* for the encryption function E. The distinguisher is a polynomial-time algorithm that "behaves" differently when given as input the pair $(E(1), E(1))$ versus the pair $(E(\text{RANDOM BIT}), E(0))$ (RANDOM BIT $\in \{0, 1\}$ is chosen uniformly at random). To this end, we first create $n - 1$ pairs of probabilistic encryptions

$$(E(\text{RANDOM BIT}_1), E(0)), \ldots, (E(\text{RANDOM BIT}_{n-1}), E(0)),$$

(where RANDOM BIT$_i \in \{0, 1\}$ is randomly chosen).

Given a pair $(E(x), E(y))$, the $n - 1$ pairs are joined to it and we feed the n pairs to $\mathscr{BL}$ (in a random order). If $x = 1$, $y = 1$, then this is a random instance of the event "exactly one leader volunteered and its random bit is 1". Therefore, according to the assumption, $\mathscr{BL}$ will output the original pair $(E(x), E(y))$ with probability no greater than $(n - t)/n - \epsilon$. If, on the other hand, x = RANDOM BIT, $y = 0$, then the inputs to $\mathscr{BL}$ are n pairs whose elements are probabilistic encryptions of identical sources. Hence the outputs are just a random subset of $n - t$ out of these n encryptions, and so the original pair $(E(x), E(y))$ is output with probability at least $(n - t)/n$.

The net effect of the whole procedure is that if $x = 1$, $y = 1$ then $(E(x), E(y))$ is output with probability $\leq (n - t)/n - \epsilon$, while if x = RANDOM BIT, $y = 0$, then $(E(x), E(y))$ is output with probability $\geq (n - t)/n$. Thus a distinguisher for E is constructed using the adversary's $\mathscr{BL}$. Invoking [15], we see that this distinguisher can be used to invert E in polynomial time, contradicting assumption $(*)$.

For the dynamic-reception model, we can once again consider a variant of the protocol that echoes the encrypted c_i and l_i values for two additional rounds. This ensures that some set of $n - t$ messages reaches all n processors (provided $t < n/2$). As in the message-oblivious case, we must once again argue that the leader's message is delivered with probability at least $(n - t)/n$. To do this, we need only make a small modification of the proof for the dynamic-broadcast case. We can view the composite three rounds of transmission and echoing as one black box, in which the adversary picks the identity of $n - t$ messages that will be delivered to all n processors. (Note that one can determine from the ensemble of messages received in the third round that $n - t$ messages are delivered to all n processors.) As in the previous proof, conditioning on the value of the leader's bit and the existence of a unique leader, the indistinguishability properties of the probabilistic encryption imply that no probabilistic polynomial-time adversary can do any better than blocking at random. □

Theorem 4 is based on the assumption that the processors have already agreed on a common public key E. This represents an additional assumption about the initial state of the system. At the cost of a more complex protocol, this assumption can be avoided.

4.1 Weakly Global Coins without Common Public Keys. As we remarked earlier, the problem of key distribution can be solved by having each processor p broadcast its own (individually generated) public key E_p. This is necessary so that other processors can send encrypted messages to p. Provided $t < n/2$, the algorithms below will flip a weakly global coin.

In the dynamic-broadcast model, processors spend an extra initial round broadcasting their public keys. This is done with every coin-toss execution. This guarantees that there are $n - t$ processors whose public keys are known to everyone. During the first round of the coin-toss broadcast, each processor encrypts messages with the public key of the recipient, or sends nothing if the recipient's public key is not known. In a second round of broadcast, all first-round messages are broadcast in the clear (unencrypted). The code follows.

```
Code for processor $P$:
 1. **function** COIN TOSS$_4$:
 2. generate and broadcast encryption key $E_p$
 3. receive all $E_q$ messages
 4. $l_p \leftarrow$ LEADER
 5. $c_p \leftarrow$ RANDOM BIT
 6. for each $E_q$ received in step 3 send $(E_q(c_p), E_q(l_p))$
 7. receive and decrypt all $(c, l)$ messages
 8. broadcast all $(c, l)$ messages received in step 7
 9. receive all $(c, l)$ messages
10. **if** all messages received with $l = 1$ have the same $c$
11.     **then** COIN TOSS$_4 \leftarrow c$ of these messages
12.     **else** COIN TOSS$_4 \leftarrow$ local coin toss
```

As before, consider the case that there is a unique leader chosen during the first round of the coin toss. Since the first-round messages are encrypted, an argument exactly analogous to that for Theorem 4 establishes that the leader's messages will be received in step 7 by at least $n - t$ recipients with probability at least $\frac{1}{2}$. Since $n - t > t$, one of these recipients will forward the leader's messages to everyone during the final clear round, steps 8 and 9. Thus, COIN TOSS_4 produces a weakly global coin in the dynamic-broadcast model for $t < n/2$.

In the dynamic-reception case, processors run the dynamic-broadcast algorithm under the simulation from Section 3.1, running three rounds of broadcasting and forwarding to implement one round of the dynamic-broadcast algorithm. (This applies to steps 2–3, 6–7, and 8–9 in the code.) One additional change must be made to the dynamic-broadcast algorithm—the simulation assumes that the same message is broadcast each round. Thus, the vector of encrypted values must be broadcast in step 6;

6′. broadcast $\langle (E_1(c_p), E_1(l_p))), \ldots, (E_n(c_p), E_n(l_p))\rangle$,

where $(E_i(c_p), E_i(l_p)) =$ "?" if E_i not received.

By invoking the same counting argument as before, there must be at least $n - t$ processors whose encryption keys are transmitted to everyone, and these $n - t$ processors will all in turn receive the encrypted messages of at least $n - t$ processors. Again, an argument analogous to the proof of Theorem 4 shows that when there is a single leader, there is a constant probability that it will be one of the latter $n - t$ processors. Since $n - t > t$, the leader's message will then be successfully forwarded to all the processors in the ensuing clear rounds. To summarize

THEOREM 5. *Under the assumption* $(*)$, *but without assuming common, predistributed encryption and decryption keys, polynomially many repeated calls of the function COIN* TOSS_4 *each produce a weakly global coin in the message-dependent dynamic-broadcast and dynamic-reception fault models, provided that* $t < n/2$.

5. *The Asynchronous Case*

In this section we abandon the assumption that processors run in synchronous rounds. Processors may run arbitrarily fast or slow, and messages may arrive out of order, or take arbitrarily long to arrive, even in the absence of failures. We make the following assumption about the nature of failures in the asynchronous model.

Asynchronous Failures. Except for a set of at most t sending processors, all messages sent by every processor are eventually delivered.

The definition implies that if m messages are sent by distinct processors to the same processor p, then p eventually receives at least $m - t$ of those messages.

We consider two failure models for the asynchronous case, the asynchronous message-oblivious and asynchronous message-dependent models. These both assume the asynchronous failure assumption, adding, respectively, the message-oblivious and message-dependent limitations from the synchronous case. In these models, the adversary has full control of the order and timing of arriving messages and of the rates of internal clocks, and is therefore more powerful than in the synchronous case. The adversary is limited in only two ways. The constraints of the failure assumption require it to eventually deliver enough messages, and the message-oblivious and message-dependent limitations restrict the information it may use to determine its strategy.

Message-Oblivious. The adversary's order of events (and, in particular, choice of delayed and undelivered messages) is independent of the contents of the messages.

Before giving a more precise definition, we first introduce a formal description of an asynchronous execution of a protocol. Our definition is taken from Fischer et al. [14]. An *execution* is a sequence of events that can be applied, in that order, starting from the initial configuration of the system. An *event* (m, p) is the receipt of a message m that is either the empty message or is from processor p's message buffer (that is, a message that was previously sent to p and not received yet). As in the synchronous case, each processor's protocols determine, upon the receipt of a message, a distribution of actions (the new local state and up to n messages sent). These messages are then placed in the addressees' message buffers. The adversary determines, as a function of the protocols, the input vector and the asynchronous execution, a distribution over the set of possible next events. An adversary is message-oblivious if for any given set of protocols, (including the input vector to the processors), and any past execution (specified by events $EV_1, EV_2, \ldots, EV_k$), the probability distribution of the next event, EV_{k+1} is independent of the message contents of nonempty messages of the first k events.

Message-Dependent. The adversary is limited to polynomial resources (time and space), but its choice of failures may depend on the contents of the messages.

In general, defining the notion of time for an asynchronous system is not a simple matter (see [2] and [13]). However, the protocols we are using are of a restricted type, in which time is naturally defined. These protocols all consist of alternating broadcast and reception phases. In the broadcast phase, a processor sends a message to all n processors. In the reception phase, the processor waits to receive messages from exactly $n - t$ processors. This is followed by a local computation, the next broadcast phase, and so on. We assume that processors begin each consensus protocol with the same value in their local round counter. In our algorithms, processors append the current value of the round counter to each message. Each processor counts local rounds, consisting of a broadcasting phase and a reception phase. During the reception phase, the processor waits for exactly $n - t$ messages with the current round number (some of which may already be received, and stored locally). For simplicity, we assume that extra messages with a given round number are discarded. In general, no processor should wait for more than $n - t$ messages from a given round, since failures may prevent more than this many messages from ever arriving. The definition of local time guarantees that no processor is more than one round ahead of the majority of other processors (recall that $t < n/2$). Of course, the slowest processors could lag far behind.

In spite of the adversary's increased power in the asynchronous case, a two-round echoing variant of the synchronous algorithm will still guarantee that agreement is reached in constant expected time, provided $t < ((3 - \sqrt{5})/2)n \approx 0.38n$.

Before we give the proof, let us first remark on the difficulties arising in the asynchronous vs. the synchronous case. One might be tempted to argue that exactly the same proofs work, since "*once the coin tosses are hidden* (*by assumption or by encryption*), *the adversary cannot know which messages to block and so everything works just as it did in the synchronous case.*" This naive argument is incorrect because an adversary can, in general, infer information about messages from the way that processors who receive these messages react to them. If the reaction of each processor to $n - t$ coin-toss messages is sufficient to infer that a single processor volunteered, the adversary can successively deliver different subsets of messages to different processors, implementing a simple elimination procedure to determine the identity of the leader. The leader's messages can then be held back from the remaining processors until they have finished the coin toss, rendering the leader useless. (Notice that the adversary could not perform such elimination in the synchronous case, where the response of processors is not observable until after the end of the round, by which time every processor already received its incoming messages for the current round.) To exemplify these notions, suppose we deal with a different protocol, in which a processor that received $n - t$ messages with round number i, among which a unique message is a leader's message, sends its next message to that leader only (and broadcasts to all n processors otherwise). In such case, the identity of the ith round leader can be inferred from the (unlabeled) communication pattern alone. Thus a message-oblivious adversary can block the leader's messages to all other processors.

It is possible to hide the identity of the leader within the consensus algorithm, by making the communication pattern independent of the identity of the leader (as our algorithms do). However, consensus protocols are meant as general-purpose tools, and it is not possible to anticipate fully the context in which they may be run. Thus, once any processor leaves the coin-toss or agreement protocol, it may behave in an arbitrary way, releasing arbitrary information to the adversary (such as publishing cryptographic keys). These protocols must ensure that information leaked by the faster processors will not jeopardize correctness by allowing the adversary undue influence over the slower processors. The asynchronous protocols below use the imposed round structure and explicit synchronization rounds to satisfy these requirements.

Specifically, in the case that there is a single leader, the identity of the leader is hidden at least until the fastest processor completes the execution of the protocol. If the leader is persuasive, the coin has the additional property that the majority value of the coin (i.e., the unique value assumed by $\lfloor n/2 \rfloor + 1$ processors) has been determined by this point. This is an important property for asynchronous coin tosses to have, in particular, for our application.

Because of the round structure we impose, the leader's messages are only effective if they are among the *first* $n - t$ messages for that round to arrive at $\lfloor n/2 \rfloor + t + 1$ other processors. For the asynchronous case this will be our definition of a *persuasive* processor for a given round. Our algorithms work by guaranteeing a positive constant probability that a single volunteer will be persuasive. Without making it explicit in the code, we implicitly assume that a round counter is locally maintained and incremented by each processor. When we say that a processor receives $n - t$ messages, we mean that it reads messages from its buffer until receiving $n - t$ messages with its current round number. The code for the asynchronous, message-oblivious model is as follows:

```
Code for processor $P$:
 1. function ASYNCHRONOUS COIN TOSS$_1$:
 2. $l_p \leftarrow$ LEADER
 3. $c_p \leftarrow$ RANDOM BIT
 4. broadcast $(c_p, l_p)$
 5. receive the first $n - t$ $(c, l)$ messages with current round number
 6. broadcast the vector $\langle (c_1, l_1), \ldots, (c_n, l_n) \rangle$ where $(c_i, l_i) =$ "?" if not received
 7. receive $n - t$ vectors $\langle (c_1, l_1), \ldots, (c_n, l_n) \rangle$ with current round number
 8. broadcast the vector $\langle (c_1, l_1), \ldots, (c_n, l_n) \rangle$ where $(c_i, l_i) =$ "?" if not received
 9. receive $n - t$ vectors $\langle (c_1, l_1), \ldots, (c_n, l_n) \rangle$ with current round number
10. if all messages received with $l = 1$ have the same $c$
11.     then COIN TOSS$_1$ $\leftarrow c$ of these messages
12.     else COIN TOSS$_1$ $\leftarrow$ local coin toss
```

We call step 4 the *coin-distribution phase*, step 6 the *first echoing phase*, and step 8 the *second echoing phase*.

THEOREM 6. *The function ASYNCHRONOUS COIN TOSS$_1$ produces a weakly global coin in the asynchronous, message-oblivious fault model, provided $t < ((3 - \sqrt{5})/2)n$.*

PROOF. In fact, we prove an even stronger statement, namely that for each invocation of asynchronous coin toss, there is a positive constant probability that each of the two outcomes of the coin will be seen (eventually) by *all* processors (the definition required only $n/2 + t + 1$ processors to see the same outcome). To prove that, we "freeze" the execution at the point when the first processor p reaches step 10. Up to this point, the actions of all processors did not depend on the contents of the messages sent and received in the current invocation of the asynchronous coin-toss procedure. We argue that at this point there is a set of S of exactly $n - t$ processors whose coins (c_p, l_p) have already reached at least $t + 1$ processors during the first echoing round. If there are more processors with this property, choose S as any subset of size $n - t$. This implies that the coins of processors in this set S will eventually be relayed to all n processors. It is crucial to notice that the members of the set S are already determined at this freezing point, and cannot be changed later, regardless of any future scheduling by the adversary.

To prove the existence of such S, we slightly modify the counting argument used in the synchronous dynamic-reception case. Consider the second round of the simulation (steps 6 and 7). We show now that at this first echoing phase, at least one processor q had its messages reach $t + 1$ processors. If all processors reached no more than t, the total number of messages that were already transmitted in the first echoing phase, M, is at most nt. Since processor p reached step 10, that means that it received the second-round echoing messages from $n - t$ processors. But each of these $n - t$ processors moved to second-round echoing after receiving $n - t$ first echoing phase messages, so that $(n - t)^2 \le M$. Thus we get $(n - t)^2 \le nt$. Substituting $t = \gamma n$, this inequality implies $(1 - \gamma)^2 \le \gamma$, contradicting the assumption that $\gamma < (3 - \sqrt{5})/2$. To wrap up the argument, observe that every processor receives at least $n - t$ messages in the distribution phase. Denoting by S the processors whose messages reached q at the coin-distribution phase, that processor q must have attempted to relay all S messages to each processor in the first echoing phase. Since there are $t + 1$ processors at the end of the first echoing phase that have been relayed these messages (from q), every processor will eventually be relayed these messages from one of the $t + 1$ processors by the end of the second echoing phase.

Of course, the adversary's actions before the freezing point determine the set S. But from the discussion above, the pattern and length of all messages sent/received by all processors up to the freezing point is independent of message contents. Thus, for a message-oblivious adversary, the choice of the set S is made independently of the messages' contents. Conditioning on the existence of a single leader, the leader will be in the set S with probability exactly $|S|/n$, which is at least $(n - t)/n > \frac{1}{2}$. □

To defeat a message-dependent adversary in the asynchronous case, we make the same alteration as in the synchronous case, encrypting the random bits.

Code for processor P:

```
1. **function** ASYNCHRONOUS COIN TOSS$_2$:
2. $l_p \leftarrow$ LEADER
3. $c_p \leftarrow$ RANDOM BIT
4. broadcast $(E(c_p), E(l_p))$
5. receive and decrypt the first $n - t$ $(E(c), E(l))$ messages with current round number
6. broadcast the vector $\langle E(c_1, l_1), \ldots, E(c_n, l_n)\rangle$ where $(c_i, l_i) =$ "?" if not received
```

```
 7. receive $n - t$ vectors $\langle E(c_1, l_1), \ldots, E(c_n, l_n)\rangle$ with current round number
 8. broadcast the vector $\langle E(c_1, l_1), \ldots, E(c_n, l_n)\rangle$ where $(c_i, l_i)$ = "?" if not received
 9. receive $n - t$ vectors $\langle E(c_1, l_1), \ldots, E(c_n, l_n)\rangle$ with current round number
10. if all messages received with $l = 1$ have the same $c$
11.     then COIN TOSS$_1$ $\leftarrow c$ of these messages
12.     else COIN TOSS$_1$ $\leftarrow$ local coin toss
```

THEOREM 7. *Under the assumption* (∗), *if all processors hold the same encryption and decryption key, then polynomially many repeated calls of the function ASYNCHRONOUS COIN TOSS$_2$ produce a weakly global coin in the asynchronous message-dependent model, provided* $t < ((3 - \sqrt{5})/2)n$.

PROOF SKETCH. As in Theorem 4, we argue that an adversary who can prevent a successful coin toss is capable of breaking the cryptosystem. By the argument in the proof of Theorem 6, the two-round echoing guarantees the existence of a set S with $\geq n - t$ processors, whose encrypted coins eventually reach everyone. Furthermore, the identity of S is determined before any processor made a step that depends on the contents encrypted under E. Conditioning on the existence of a unique leader, the adversary can successfully block the coin toss only if he can discard the leader from S. The adversary acts based on the n pairs of encrypted coins $(E(c_1), E(l_1)), \ldots, (E(c_n), E(l_n))$. By the same argument as in Theorem 4, if the adversary succeeds substantially more often than he would by guessing the leader's identity at random, he could use this capability to invert the cryptosystem. Since the probability of successfully placing the leader outside S by guessing his identity at random is $\leq t/n < \frac{1}{2}$, we are done. □

Similar to the message-dependent synchronous case, we perform a four-round key distribution phase as part of the protocol, where the number of faults tolerated is $t < ((3 - \sqrt{5})/2)n$. In performing the key distribution, we follow a round of broadcast by a three-round echoing scheme, which will be sufficient to ensure the following: there exists sets S_1 and R_1 of $n - t$ processors each, where the key of each processor in S_1 is received by each processor in R_1 before the fastest processor starts the current epoch's coin-flipping protocol. As in the proof of Theorem 7, there exists a set S with $n - t$ processors, so that for every p in S, the message, $\langle(E_1(c_p), E_1(l_p))), \ldots, (E_n(c_p), E_n(l_p))\rangle$ (where $(E_i(c_p), E_i(l_p))$ = "?" if E_i was not received by p), will eventually be received by all n processors. As before, the identity of S is determined before any processor makes a step that depends on the contents of encrypted messages. $|R_1 \cap S| \geq n - 2t > n/5$, so that a blocking argument similar to the proof of Theorem 4, shows that the unique leader is in this intersection with probability at least $\frac{1}{5}$. This implies the following theorem.

THEOREM 8. *Under the assumption* (∗), *but without assuming common, predistributed encryption and decryption keys, repeated calls of a modified function ASYNCHRONOUS COIN TOSS$_2$ preceded by a four-round encryption-key-distribution phase, produce a weakly global coin in the message-dependent asynchronous fault model, provided that* $t < ((3 - \sqrt{5})/2)n$.

6. *Using a Weakly Global Coin in Achieving Consensus*

In this section we present an agreement algorithm that can be implemented using a weakly global coin. For simplicity of presentation, the algorithm given here is binary (reaching agreement on one bit), and is basically a modification of those in [4] and [6].

We begin with an informal description of the algorithm. The algorithm is organized as a series of epochs of message exchange. Each epoch consists of several rounds. The round structure is provided automatically in the synchronous models. In the asynchronous models, the round structure is imposed locally by each processor, as was discussed earlier. In this case, reaching consensus in "constant expected time" means that each processor will complete the protocol within a constant expected number of local rounds.

We describe the algorithm for the processor P. (All processors run the same code.) Epoch and round numbers are always the first two components of each message. The variable CURRENT holds the value that processor P currently favors as the answer of the agreement algorithm. At the start of the algorithm CURRENT is set to processor P's input value. In the first round of each epoch, processor P broadcasts CURRENT. Based on the round-1 messages received, processor P changes CURRENT. If it sees at least $\lfloor n/2 \rfloor + 1$ round-1 messages for some particular value, then it assigns that value to CURRENT; otherwise, it assigns the distinguished value "?" to CURRENT. In the second round of each epoch, processor P broadcasts the new CURRENT. This is followed by a synchronization round, in which all processors broadcast waiting messages, then wait until $n - t$ such messages are received. This guarantees that at least $n - t$ processors have finished the previous round before the fastest processor leaves this round. Next, the COIN TOSS subroutine is run. (Of course, in an asynchronous model this statement is a bit imprecise, since the subroutine is first initiated at the point that the fastest processor reaches the subroutine call.) Based on the round-2 messages received, processor P either changes CURRENT again, or decides on an answer and exits the algorithm at the end of the next epoch. Let ANS be the most frequent value (other than "?") in round-2 messages received by P. Let NUM be the number of such messages. There are three cases depending on the value of NUM. If NUM $\geq \lfloor n/2 \rfloor + 1$, then processor P decides on the value ANS and exits the algorithm by the end of the next epoch. If $\lfloor n/2 \rfloor \geq$ NUM ≥ 1, then processor P assigns the value ANS to the variable CURRENT and continues the algorithm. If NUM $= 0$, then processor P assigns the result of the coin toss to the variable CURRENT, and continues the algorithm.

Code for processor P:

```
1.  procedure AGREEMENT(INPUT):
2.  CURRENT ← INPUT
3.  TERM.NEXT ← "OFF"
4.  for e ← 1 to ∞ do
5.     broadcast (e, 1, CURRENT)
6.     receive (e, 1, *) messages
7.     if for some v there are ≥ ⌊n/2⌋ + 1 messages (e, 1, v)
8.        then CURRENT ← v
9.        else CURRENT ← "?"
10.    broadcast (e, 2, CURRENT)
11.    receive (e, 2, *) messages
12.    if there exists v ≠ "?" such that (e, 2, v) was received
13.       then ANS ← the value v ≠ "?" such that (e, 2, v) messages are most frequent
14.       else ANS is undefined
15.    NUM ← number of occurrences of (e, 2, ANS) messages
16.    broadcast (e, 3, "waiting")
17.    receive (e, 3, "waiting") messages
18.    COIN ← COIN TOSS
19.    if TERM.NEXT = "ON" then terminate
20.    if NUM ≥ ⌊n/2⌋ + 1 then decide ANS, set CURRENT ← ANS and TERM.NEXT
          ← "ON"
```

```
21.   else if NUM $\geq$ 1
22.       then CURRENT $\leftarrow$ ANS
23.       else CURRENT $\leftarrow$ COIN
```

We make several remarks about the algorithm. COIN TOSS, depending on the fault model, is one of the protocols described earlier for producing a weakly global coin. In message descriptions, "$*$" is a wild-card character that matches anything. Notice that once a processor has decided, it participates in the protocol for another epoch. Although not explicitly given in the code, during this extra epoch the processor ignores all "receive" commands, since otherwise it may be left waiting for messages from processors that have already terminated. The extra epoch is needed because, once the first processor decides and terminates, the other processors may not decide until the next epoch (as we argue below). The extra broadcasts by decided processors are solely to ensure that these "tardy" processors receive a sufficient number of messages during each round of that epoch. (Recall that in the asynchronous fault models, processors must wait for $n - t$ messages during each reception.)

If the input values are sufficiently biased towards a particular value, the protocol will reach agreement in one epoch. If this is not the case, the protocol uses the weakly global COIN TOSS function to prevent the system (abetted by the adversary) from "hovering" at an indeterminate point indefinitely. With each call to COIN TOSS, there is a constant probability that the outcome will bias the system sufficiently to reach agreement quickly. Thus, agreement will be reached in constant expected time.

Define *value* as a legal input to the algorithm, either 0 or 1. Specifically, "?" is not a value.

The following lemma is used in proving the desired properties of the agreement algorithm, and proved by a simple counting argument:

LEMMA 9. *During each epoch, both of the values* 0 *and* 1 *are never sent in any execution of round* 2 (*step* 10).

Theorem 10 will establish that this algorithm never produces conflicting decisions and that in each epoch there is at least one coin-toss value that will lead to termination of the algorithm.

Before presenting this theorem, it is necessary to introduce several key notions that are particularly important in the analysis of the asynchronous case. The value ANS is critical in the analysis of the protocol. At any instant of an execution of the protocol, an epoch e is *bivalent* if, for both $v = 0$ and $v = 1$, there exists an execution of the protocol that continues from that instantaneous position, for which there exists a processor that has an ANS value in epoch e equal to v. Furthermore, let k_e be the number of processors that have not determined whether ANS is 0, 1 or undefined for epoch e at the point that the fastest processor begins the coin toss for epoch e. Note that in all the synchronous models discussed, $k_e = 0$ at the point that the COIN TOSS protocol is executed in round e. This may not be the case in the asynchronous cases, where the epoch may still be bivalent at the point when the fastest processor initiates the execution of COIN TOSS for that epoch. However, the round of "waiting" messages ensures that at the point when the COIN TOSS is first initiated, k_e is at most t (since the fastest processor must have received $n - t$ "waiting" messages in order to continue, and these processors have already executed through step 16). Note that if an epoch is bivalent, then any processor that has already determined ANS at this point has ANS = "undefined".

THEOREM 10. *The algorithm has the following three properties*:

Validity: *If value v is distributed as input to all processors, then all processors decide v during epoch* 1.

Agreement: *Let e be the first epoch in which a processor decides. If processor P decides v in epoch e, then by the end of epoch e* + 1 *all processors decide v.*

Termination: (*a*) *In any epoch, e, if the epoch is not bivalent at the point when the fastest processor begins executing step* 18, *then there is at least one value that, if it is adopted by* $\lfloor n/2 \rfloor + t + 1$ *processors executing the assignment in step* 18, *will cause each processor to decide by the end of epoch e* + 1, *and otherwise*, (*b*) *in any epoch, e, if there is a value that is adopted by* $\lfloor n/2 \rfloor + t + 1$ *processors executing the assignment in step* 18, *then epoch e* + 1 *is not bivalent at the point that the majority value of COIN TOSS in epoch e is uniquely determined.*

As we argue below, the termination property guarantees that a weakly global coin will lead to a decision with constant probability. The agreement property guarantees that once a single processor decides, all other processors will decide in the next epoch, regardless of the adversary's behavior. In particular, this holds for the asynchronous, message-dependent model, the one in which the adversary has the most power. The proofs follow by the techniques of [4], and we highlight only the interesting distinctions between the asynchronous case and the synchronous case, which were presented in detail in [7] and [9].

The only significant difficulty that an asynchronous model presents in proving the termination of the protocol. As the criterion indicates there are two cases to consider. The first case, (a), is actually similar to the synchronous case; if the epoch is not bivalent at the point that the fastest processor initiates step 18, then if the majority value of COIN TOSS is equal to the only possible value for ANS (that is not undefined), then the $\lfloor n/2 \rfloor + t + 1$ processors that receive the majority value of the toss all have the same CURRENT value at the start of the next epoch $e + 1$, and so every processor will receive at least $\lfloor n/2 \rfloor + 1$ of them and set CURRENT for the second round of epoch $e + 1$ to that value. This unanimity will cause each processor to decide in epoch $e + 1$, and terminate in the next one.

The second case is a bit more delicate. Suppose that at least c processors receive the value b of the coin, where $c \geq \lfloor n/2 \rfloor + t + 1$. Thus all but at most $n - c$ processors that execute step 18 get b. If at the point that the fastest processor executes step 18, the epoch is bivalent, then $n - k_e$ processors have an undefined value of ANS for that epoch, and will execute step 23. Therefore, at least $n - k_e - (n - c) = c - k_e$ processors set their value of CURRENT for the start of the next epoch to b. In other words, at most $n - c + k_e$ processors get $\neg b$ as CURRENT for the start of the next epoch. Since

$$n - c + k_e \leq n - \left(\left\lfloor \frac{n}{2} \right\rfloor + t + 1\right) + t \leq \left\lfloor \frac{n}{2} \right\rfloor,$$

we see that the next epoch is not bivalent as soon as the majority value of the coin is determined, and the termination condition is proved.

Finally, we argue that this really implies termination in constant expected time. For this we require that with constant probability, COIN TOSS generates a weakly global coin whose value is determined by the point that the fastest processor completes the execution of the routine. As we remarked earlier, each of our coin-flipping routines satisfies this condition in the appropriate fault model. In order to complete the proof, we need only observe that in epochs e and $e + 1$, the probability of terminating by condition (a) if e is not bivalent at the crucial point is at least a constant (by the property of weakly global coins), and otherwise the probability of invoking (b) to force $e + 1$ to be nonbivalent is at least a constant (by the property discussed above).

All of the variants of the coin-toss procedure that we have considered take a constant number of rounds. Combining Theorem 10 with the various versions of the coin-toss procedure, we get

THEOREM 11. *Using the agreement algorithm with coin toss as a subroutine, agreement is reached in constant expected number of rounds, provided the number of faults t satisfies*

(*a*) $t < n/2$ *for the all variants of the synchronous model*;
(*b*) $t < ((3 - \sqrt{5})/2)n$ *for all variants of the asynchronous model.*

Notice that implicit in the proof of Theorem 11 is the explicit probability that the protocol finishes by round k. If we consider any of the synchronous models, we see that the waiting round in each epoch can be omitted, so that a straightforward calculation gives that the probability of the protocol terminating within k rounds is $1 - (c + t/(2en))^{k/2}$, where $c = (2e - 1)/2e$.

It is natural to ask whether the number of erratic processors tolerated can be significantly improved. A result of Bracha and Toueg [6] shows that no randomized consensus protocol can tolerate more than $n/2$ fail-stop faults in an asynchronous model.

7. *Lower Bounds*

In this section we show that our upper bound is almost optimal in a strong sense. We demonstrate a lower bound on the tail of the distribution of nontermination probabilities for any randomized agreement algorithm. This lower bound holds for the case of a nonadaptive adversary in the fail-stop model, and therefore in the stronger failure models as well.

Let $\mathscr{A}$ be a randomized agreement algorithm that is resilient to t processor failures. Such an algorithm, together with the n input values and n (possibly infinite) $0 - 1$ strings (outcome of individual coin tosses) totally determine the behavior of each processor. Denote by q_k the maximum probability, over all (nonadaptive) adversarial strategies and over all combinations of input values, that $\mathscr{A}$ does not terminate in k rounds ($k \leq t$).

THEOREM 12

$$q_k \geq \frac{1}{2}\left(2\left\lceil \frac{n}{\lfloor t/k \rfloor} \right\rceil\right)^{-k}.$$

PROOF. In [10], Dolev and Strong prove that the worst-case time to reach agreement in a synchronous system with up to t fail-stop faults is $t + 1$ rounds. From any deterministic algorithm, the proof explicitly constructs a chain $\mathscr{S}_1$, $\mathscr{S}_2, \ldots, \mathscr{S}_m$ of partially specified executions in the fail-stop model. Each $\mathscr{S}_i$ consists of k rounds, $k \leq t$, and it specifies the identity of the faulty processors, their failure time, and the identity of receivers of their last-round message. Thus every $\mathscr{S}_i$ can be viewed as a k-round strategy of a nonadaptive adversary. These partially specified executions include the initial input values to each processor. Together with the n coin-tossing strings $\vec{c} = (c_1, c_2, \ldots, c_n)$, each $\mathscr{S}_i$ gives a complete specification of an execution, which we denote by $\mathscr{S}_i^{\vec{c}}$. For every $\vec{c}$, the following properties hold:

(1) $\mathscr{S}_i^{\vec{c}} \sim \mathscr{S}_{i+1}^{\vec{c}}$ (i.e., both executions look the same for at least one processor that is nonfaulty through the kth round).
(2) If all processors agree in $\mathscr{S}_1^{\vec{c}}$, then they must agree on the value 1.
(3) If all processors agree in $\mathscr{S}_m^{\vec{c}}$, then they must agree on the value 0.
(4) $m \le 2(2\lceil \frac{n}{\lfloor t/k \rfloor} \rceil)^k$

Details of the construction may be found in [10] (or see [9] and [19] for a simpler exposition); we note here that Properties 1 through 3 are proved explicitly in that reference. Property 4 is a simple counting argument: the construction is recursive, with k levels of recursion and $(2\lceil \frac{n}{\lfloor t/k \rfloor} \rceil)$ recursive calls at each level. Each call is made twice at the top level.

We show that $q_k \ge 1/m$. Substituting (4), this establishes the result.

Assume to the contrary that $q_k < 1/m$. Then the probability (over all $\vec{c}$'s) that $\mathscr{A}$ *does not* terminate in $\mathscr{S}_2^{\vec{c}}$ or in $\mathscr{S}_2^{\vec{c}} \cdots$ or in $\mathscr{S}_m^{\vec{c}}$ is at most $m \cdot q_k < 1$. Hence the set of $\vec{c}$'s for which $\mathscr{A}$ terminates in all $\mathscr{S}_i^{\vec{c}}$ $(1 \le i \le m)$ has measure > 0. For each $\vec{c}$ in this set, all correct processors will decide on the value 1 in $\mathscr{S}_1^{\vec{c}}$ (by Property 2). Hence, by Property 1, there is a correct processor that will decide on the value 1 in $\mathscr{S}_2^{\vec{c}}$, and therefore, by the agreement requirement, all correct processors will decide on the value 1 in $\mathscr{S}_2^{\vec{c}}$. Carrying this argument inductively, it follows that for all $1 \le i \le m$, all correct processors will decide on the value 1 in $\mathscr{S}_i^{\vec{c}}$. But for $i = m$, this contradicts Property 3. □

8. *Stronger Adversaries and Future Directions*

One limitation of the adversary that was crucial for the performance of our protocols is that the adversary does not know the internal state of processors, even when they are made faulty. The reason for this requirement is that otherwise, by delivering all messages to one specific processor, the adversary can find out the identity of the unique leader by examining the state of the receiving processor. The adversary can then block the messages of the unique leader from reaching all other processors.

We believe that there is a simple modification of the protocols that make them immune to an adversary who can "peek into the memory" of failed processors. The basic idea is that instead of sending a pair of (possibly encrypted) bits ("leader" bit, "coin" bit), to all processors, a secret sharing scheme with threshold t is used (e.g., [22]). The message to processor i will consist of the ith piece of the secret ("leader" bit, "coin" bit). Suppose the adversary makes up to t processors faulty and gets to see the contents of their memory. This does not help in understanding the contents of any sender's message. In particular, the adversary cannot use these pieces to identify the unique leader. To reconstruct the secrets, all processors later broadcast all the pieces of secrets that they have received. The adversary cannot prevent such reconstruction of the secret of any nonfaulty sender, since any $t + 1$ pieces can be used. It appears that this approach can be carried out in all variants of the adversary model that were considered. This would yield consensus protocols with constant expected running time for $t < \beta n$ (where the exact value of $\beta < \frac{1}{2}$ depends on the model), which tolerate an adversary who knows the internal state of up to t failed processors.

Finally, we note that our protocols do not work in the presence of even a single Byzantine failure. A faulty processor can simply claim, at every round, that it is a leader, thus rendering the coin-tossing subroutine ineffective. It remains an interesting question to obtain Byzantine agreement procedures that are both as simple and as efficient.

ACKNOWLEDGMENTS. We would like to thank Brian Coan and Steve Mahaney as well as the anonymous referees for their comments on earlier versions of this manuscript.

REFERENCES

NOTE: Reference [18] is not cited in text.

1. ALEXI, W., CHOR, B., GOLDREICH, O., AND SCHNORR, C. P. RSA and Rabin: Certain parts are as hard as the whole. *SIAM J. Comput. 17* (1988), 194–209.
2. AWERBUCH, B. Complexity of network synchronization. *J. ACM 32*, 4 (Oct. 1985), 804–823.
3. AWERBUCH, B., BLUM, M., CHOR, B., GOLDWASSER, S., AND MICALI, S. How to implement Bracha's $O(\log n)$ Byzantine agreement algorithm. Unpublished manuscript.
4. BEN-OR, M. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing.* ACM, New York 1983, pp. 27–30.
5. BRACHA, G. An $O(\log n)$ expected rounds randomized Byzantine generals algorithm. *J. ACM 34*, 4 (Oct. 1987), 910–920.
6. BRACHA, G., AND TOUEG, S. Asynchronous consensus and broadcast protocols. *J. ACM 32*, 4 (1985), 824–840.
7. CHOR, B., AND COAN, B. A simple and efficient randomized Byzantine agreement algorithm. *IEEE Trans. Softw. Eng. SE-11*, 6 (1984), 531–539.
8. CHOR, B., MERRITT, M., AND SHMOYS, D. B. Simple constant-time consensus protocols in realistic failure models. In *Proceedings of the 4th Annual ACM Symposium on Principles of Distributed Computing.* ACM, New York, 1985, pp. 152–162.
9. COAN, B. Achieving consensus in fault-tolerant distributed systems: Protocols, lower bounds and simulations. Ph.D. dissertation. MIT, Cambridge, Mass., 1987.
10. DOLEV, D., AND STRONG, H. R. Polynomial algorithms for multiple processor agreement. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing.* ACM, New York, 1982, pp. 401–407.
11. DWORK, C., SHMOYS, D., AND STOCKMEYER, L. Flipping persuasively in constant expected time. In *Proceedings of the 27th Symposium on Foundations of Computer Science.* IEEE, New York, 1986, pp. 222–232.
12. FELDMAN, P., AND MICALI, S. Optimal algorithms for Byzantine agreement. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing.* ACM, New York, 1988, pp. 148–161.
13. FISCHER, M. J., AND LYNCH, N. A. On describing the behaviour and implementation of distributed systems. *Theoret. Comput. Sci. 13* (1981), 17–43.
14. FISCHER, M. J., LYNCH, N. A., AND PATERSON, M. S Impossibility of distributed consensus with one faulty process *J. ACM 32*, 2 (Apr. 1985), 374–382.
15. GOLDWASSER, S., AND MICALI, S. Probabilistic encryption. *J. Comput. Syst. Sci. 28*, 2 (1984), 270–299.
16. GOLDWASSER, S., MICALI, S., AND RACKOFF, C. The knowledge complexity of interactive proof systems. *SIAM J. Comput. 18*, 1 (1989), 186–208.
17. KARLIN, A. R., AND YAO, A. C. Probabilistic lower bounds for Byzantine agreement and clock synchronization. Unpublished manuscript.
18. LAMPORT, L., SHOSTAK, R., AND PEASE, M. The Byzantine generals problem. *ACM Trans. Prog. Lang. Syst. 4*, 3 (July 1982), 382–401.
19. MERRITT, M. Unpublished manuscript.
20. PEASE, M., SHOSTAK, R., AND LAMPORT, L. Reaching agreement in the presence of faults. *J. ACM 27*, 2 (Apr. 1980), 228–234.
21. PINTER, S. *Distributed Computation Systems.* Ph.D. dissertation, Boston Univ., Boston, Mass., 1983.

22. RABIN, M. O. Randomized Byzantine generals. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science.* IEEE, New York, 1983, pp. 403–409.
23. ROSEN, E. C. Vulnerability of network control protocols: An example. *ACM SIGSOFT Softw. Eng. Notes, 6,* 1 (1981), 6–8.
24. SHAMIR, A. How to share a secret. *Commun. ACM 22,* 11 (Nov. 1979), 612–613.
23. YAO, A. C. Theory and applications of trapdoor functions. In *Proceedings of the 23rd IEEE Symposium on Foundation of Computer Science.* IEEE, New York, 1982, pp. 80–91.

RECEIVED JULY 1986; REVISED JANUARY 1988 AND NOVEMBER 1988; ACCEPTED NOVEMBER 1988