Distributed Computing (1987) 2:80–94

DISTRIBUTED COMPUTING

Simulating authenticated broadcasts to derive simple fault-tolerant algorithms*

T.K. Srikanth and Sam Toueg
Cornell University, Ithaca, NY 14853, USA

T.K. Srikanth *He received the B.Tech. degree in Mechanical Engineering from the Indian Institute of Technology, Madras, in 1981. He received the M.S. and Ph.D. degrees in computer science from Cornell University, in 1985 and 1986, respectively. He is currently a research scientist at XOX Corporation, Ithaca, New York. His research interests include distributed computing, fault-tolerance, and geometric modeling.*

Sam Toueg *He received the B.Sc. Degree in computer science from the Technion, Israel Institute of Technology, in 1976, and the M.S.E., M.A., and Ph.D. degrees in computer science from Princeton University, in 1977, 1978, and 1979, respectively. He spent a post-doctoral year at the IBM Thomas J. Watson Research Center, at Yorktown Heights, in the Systems Analysis and Algorithms division. In 1981 he joined the Department of Computer Science at Cornell University, Ithaca, NY, where he is currently an Associate Professor. His current research interests include distributed computing, fault-tolerance, computer networks, and distributed database systems. Dr. Toueg is a member of the Association for Computing Machinery SIGACT and SIGCOMM.*

Abstract. Fault-tolerant algorithms for distributed systems with arbitrary failures are simpler to develop and prove correct if messages can be authenticated. However, using digital signatures for message authentication usually incurs substantial overhead in communication and computation. To exploit the simplicity provided by authentication without this overhead, we present a broadcast primitive that provides properties of authenticated broadcasts. This gives a methodology for deriving non-authenticated algorithms. Starting with an authenticated algorithm, we replace signed communication with the broadcast primitive to obtain an equivalent non-authenticated algorithm. We have applied this approach to various problems and in each case obtained simpler and more efficient solutions than those previously known.

Key words: Distributed systems – Fault-tolerance – Byzantine agreement – Authentication

* Partial support for this work was provided by the National Science Foundation under grant MCS 83-03135

Offprint requests to: T.K. Srikanth

1 Introduction

Fault-tolerance is an important issue in distributed systems. However, reasoning about distributed computations is difficult, and particularly so when arbitrary types of failures can occur. In this paper, we study techniques that impose restrictions on the visible behavior of faulty processes and thereby simplify the task of designing fault-tolerant algorithms.

To illustrate our approach, we first consider the problem of reaching agreement among processes when some of them may be faulty. This problem, called the *Byzantine Generals Problem or Byzantine Agreement*, is a central issue in the design of fault-tolerant systems (Lamport et al. 1982; Mohan et al. 1983; Garcia-Molina et al. 1984) Formally, Byzantine Agreement requires that when a message is sent by a transmitter to a set

of processes the following two conditions are satisfied:

Agreement

All correct processes agree on the same message.

Validity

If the transmitter is correct, then all correct processes agree on its message.

We assume a set of n processes, of which no more than t are faulty. A process is *correct* if it always follows the agreement algorithm; it is *faulty* otherwise. Correct processes must reach agreement on a message $m \in \mathbf{M} \cup \{\text{"sender faulty"}\}$, where $\mathbf{M}$ is the set of messages the transmitter can send. We make no assumptions about the behavior of faulty processes – they can even be malicious in attempting to foil agreement. We assume a completely connected network and a reliable message system in which a process receiving a message can identify the immediate sender of the message.

One way to restrict the visible behavior of faulty processes is to assume that the message system is authenticated (Lamport et al. 1982; Dolev and Strong 1983; Merritt 1984). Informally, authentication prevents a process from changing a message it relays, or introducing a new message into the system and claiming to have received it from some other process. This restriction on the behavior of faulty processes not only simplifies the design of fault-tolerant algorithms, but often results in algorithms that are simpler, more efficient, and tolerate more faults in the system than algorithms without authentication (Fischer 1983).

We first consider synchronized algorithms that proceed in synchronized *phases*. Informally, a phase is an interval of time where processes first send messages (according to their states), wait to receive messages sent by other processes in the same phase, and then change their states accordingly.

If authentication is not available, the best known Byzantine Agreement algorithm requires $2t+3$ phases and has a message complexity of $O(nt+t^3 \log t)$ bits (Dolev et al. 1982).[1] This algorithm is unintuitive and hard to understand, as are most non-authenticated algorithms. On the other hand, Dolev and Strong derive an authenticated Byzantine Agreement algorithm that is easy to understand and prove correct (Dolev and Strong 1982). Thus, we see that the assumption that messages are authenticated simplifies the development of fault-tolerant algorithms. This is because authentication imposes restrictions on the otherwise arbitrary behavior of faulty processes.

Cryptographic techniques that provide digital signatures can be used for message authentication (Rivest et al. 1978). However, all known cryptographic schemes have disadvantages. They all require some computational and communication overhead. Furthermore, none of them has been proven unconditionally secure from attacks by malicious processes. In fact, malicious processes can break such schemes by computing or guessing the signature of another process. Although the probability of such an occurrence can be very small, it is nevertheless non-zero.

We describe a methodology for deriving non-authenticated algorithms with the simplifying assumption that messages *are* authenticated, but without paying the costs of digital signatures. The idea is as follows. We first derive an algorithm assuming message authentication. We then identify the properties of authentication the algorithm uses and derive a broadcast primitive that provides these properties *without* using signatures. Finally, we automatically convert the authenticated algorithm into a nonauthenticated one by just replacing signed communication in the original algorithm with our communication primitive. This translation method results in a non-authenticated algorithm which is as simple as the original authenticated algorithm; furthermore, it has the same proof of correctness. However, it may tolerate fewer faults than the corresponding authenticated algorithm.

Apart from simplifying the design of fault-tolerant algorithms, this approach also unifies a large class of results. Previous work has provided different and more or less unrelated solutions for a problem depending on whether or not authentication is available. For example, the simple authenticated algorithm by Dolev and Strong (Dolev and Strong 1982) did not seem to help solve the non-authenticated version of the problem (Dolev et al. 1982). Other examples are the problems of Byzantine Elections (Merritt 1984) and clock synchronization (Halpern et al. 1984; Lundelius and Lynch 1984). Our broadcast primitive and translation technique yield uniform solutions for both systems.

We illustrate this new approach on two synchronous agreement problems. The first application gives an efficient non-authenticated algorithm for Byzantine Agreement that improves on known algorithms by terminating in $2t+1$ phases using

[1] A recent result by Coan (1986) shows that for any $\varepsilon > 0$, Byzantine Agreement can be achieved in $(1+\varepsilon)(t+1)$ phases with a polynomial message complexity

$O(nt^2 \log n)$ bits. Further, this algorithm is as simple as the authenticated algorithm in (Dolev and Strong 1982) from which it is derived.

We then consider the Byzantine Elections problem. This problem was solved assuming message authentication (Merritt 1984). By replacing signed communication with our primitive, we automatically obtain the first known non-authenticated algorithm for Byzantine Elections. Its communication complexity is the same as that of the authenticated algorithm.

We then extend this approach to asynchronous systems and use it to derive a non-authenticated randomized Byzantine Agreement algorithm from an authenticated one (Toueg 1984). The resulting algorithm has a lower message complexity than the original authenticated one.

This approach has also been applied to derive a simple, efficient early-stopping Byzantine Agreement algorithm (Toueg et al. 1987), and to derive algorithms for synchronizing clocks (Srikanth and Toueg 1987).

It should be noted that authentication is different from *secrecy* (Tanenbaum 1981). Cryptographic techniques can be used to achieve both. Our communication primitives provide authentication, not secrecy.

2 Properties of authenticated broadcasts

Consider an algorithm that proceeds in synchronous rounds[2] and uses authenticated broadcasts. A process p *broadcasts* a message m in round k by sending signed copies of the triple (p, m, k) to all processes (including itself). A process that receives (p, m, k) *accepts* it if it can verify p's signature. We denote these two operations by *broadcast* (p, m, k) and *accept* (p, m, k).

Since a message broadcast by a correct process in a synchronous system is received by all correct processes in the same round, and since we assume that signatures of correct processes cannot be forged, authenticated broadcasts satisfy the following two properties:

1. (*Correctness*) If correct process p broadcasts (p, m, k) in round k, then every correct process accepts (p, m, k) in the same round.
2. (*Unforgeability*) If process p is correct and does not broadcast (p, m, k), then no correct process ever accepts (p, m, k).

One advantage of authentication is that it prevents faulty processes from modifying broadcasts of correct processes or from sending messages on behalf of correct processes. This restriction is captured by the *unforgeability* property. Authentication has another advantage as explained below.

If a correct process accepts a message signed by a process p, it cannot be sure that other processes have also accepted this message. However, by *relaying* p's signed message, it ensures that every process receives it, verifies p's signature and accepts the message. Hence, authentication provides the following property: if a process relays all the signed messages it accepts in a round, then it ensures that all processes accept these messages by the next round. Therefore, if processes immediately relay every message they accept, then message authentication provides the following additional property:

3. (*Relay*) If a correct process accepts (p, m, k) in round $r \geq k$, then every other correct process accepts (p, m, k) in round $r+1$ or earlier.

Note that *correctness* implies that messages broadcast by *correct* processes are accepted by all correct processes in the same round. On the other hand, a message broadcast by a *faulty* process p and later relayed by other processes might be accepted by a correct process many rounds after it was first broadcast by p. Thus, a correct process might accept (p, m, k) in some round $r \geq k$.

Receiving a relayed message m with p's signature does not necessarily imply that p is indeed the originator of the message. In fact, p could have been faulty and given its signature to another process. This process could then originate m "signed by p" (Lamport et al. 1982). Therefore, *unforgeability* only guarantees that if a process accepts (p, m, k), it can infer that, *if* p is correct, then p is the originator of the message.

3 An algorithm using authenticated broadcasts

We now consider an authenticated algorithm for Byzantine Agreement. This algorithm, presented in Fig. 1, is similar to that in Dolev and Strong (1983) restricted to the case where messages are binary, i.e., processes attempt to reach agreement on a value $m \in \{0,1\}$. Algorithms for multivalued agreement are described in Sect. 6. The following is an informal description of the binary algorithm.

The algorithm proceeds in synchronous rounds. In this algorithm, the only value broadcast by correct processes is 1. The value 0 is decided upon by default. A correct transmitter broadcasts 1 in round 1 if agreement is to be reached on the value 1. Otherwise, the transmitter remains silent. Process p sets the variable *value* to 1 in

[2] For the present, a round is just a phase

```
process p:                                   /*m∈{0,1}*/
if p is the transmitter then value:=m else value:=0;
for r:=1 to t+1 do
  if value=1 and
      p has not broadcast a message in earlier rounds
    then broadcast (p, 1, r);
         relay the r−1 messages accepted in previous rounds
         that caused value to be set to 1;
  if in rounds r' ≤ r, accepted (p_k, 1, r_k)
     from r distinct processes p_k, including the transmitter p_1
     then value:=1;
od
decide value.
```

Fig. 1. A binary Byzantine Agreement algorithm using authenticated broadcasts

round r, and thereby *decides* on 1, if it has accepted r messages, each message signed by a distinct process, one of which is the transmitter. If p sets *value* to 1 in round r, then in round $r+1$, it signs and broadcasts its own message, and relays the r messages that caused it to set its *value* to 1. The *correctness* and *relay* properties of authenticated broadcasts ensure that p's signed message, and all r messages that caused p to set *value* to 1, are accepted by all correct processes by round $r+1$. Hence, they too will set *value* to 1, and decide on 1.

If 0 is the value to be agreed upon, then a correct transmitter never broadcasts any message. By *unforgeability* of authenticated broadcasts, no correct process ever accepts any message originating from the transmitter. Hence, no correct process sets *value* to 1, and all correct processes decide on the default value 0.

Theorem 1. *The authenticated algorithm in Fig. 1 achieves Byzantine Agreement in $t+1$ rounds with $O(n^2\ t \log n)$ message bits.*

Proof. The proof is similar to that in Dolev and Strong (1983). Note that broadcasts are authenticated and processes immediately relay the signed messages they accept. Therefore, broadcasts satisfy the *correctness, unforgeability* and *relay* properties.

We first show that *Validity* is achieved. That is, if the transmitter p_1 is correct, then every correct process decides on the transmitter's value.

(i) If agreement is to be reached on the value 1, the transmitter initially sets *value* to 1 and broadcasts $(p_1, 1, 1)$ in round 1. By *correctness*, every correct process accepts $(p_1, 1, 1)$ in round 1. Therefore, every correct process sets *value* to 1. Once a process sets *value* to 1, it does not change it and hence decides on 1.

(ii) If agreement is to be reached on 0, the transmitter initially sets *value* to 0 and does not broadcast any message in round 1. Therefore, by *unforgeability* it will never broadcast any message in any other round either. Hence, by *unforgeability*, no correct process can ever accept a message from the transmitter. Therefore, every correct process retains *value* = 0 throughout the algorithm, and when it terminates, decides on 0.

To show that *Agreement* is reached, we consider the following two cases:

(i) If some correct process p first sets *value* to 1 at the end of round $r < t+1$, it must have accepted messages $(p_k, 1, r_k)$ from at least r distinct processes p_k including the transmitter p_1 (note that $p \neq p_k$ for $1 \leq k \leq r$). In round $r+1$, it broadcasts $(p, 1, r+1)$. By the *correctness* and *relay* properties, in round $r+1$ every correct process accepts $(p, 1, r+1)$ and $(p_k, 1, r_k)$ for $1 \leq k \leq r$, and thus sets *value* to 1.

(ii) If a correct process first sets *value* to 1 in round $t+1$, it must have accepted messages $(p_k, 1, r_k)$ from $t+1$ distinct processes p_k. At least one of these processes must be correct. This correct process, say p_i, broadcast $(p_i, 1, r_i)$ in round $r_i \leq t+1$. Therefore, p_i set *value* to 1 in round $r_i - 1 < t+1$. By case (i), every correct process sets *value* to 1 by the end of round $t+1$.

Thus, if a correct process sets *value* to 1, every correct process sets *value* to 1 and decides on 1. Otherwise, every correct process has *value* = 0 and decides on 0. Therefore, *Agreement* is satisfied.

The algorithm requires $t+1$ rounds. Each process broadcasts at most one message and relays up to $O(t)$ signed messages. Thus, correct processes send a total of $O(n^2 t)$ messages. Assuming that each signature requires $O(\log n)$ bits, the algorithm requires $O(n^2 t \log n)$ bits of information exchange. □

Note that the proof of correctness relies only on the *correctness, unforgeability* and *relay* properties provided by signed broadcasts.

4 An implementation of authenticated broadcasts

Implementing authentication using digital signatures is one way of providing the properties of *correctness, unforgeability* and *relay* described in Sect. 2. In this section, we describe a broadcast primitive that provides these three properties of authenticated broadcasts without using signatures (Fig. 2). Informally, we achieve this by requiring that to broadcast a message, a process has to use a set of processes as "witnesses" of this event. A correct process accepts a message only when it knows that there are sufficient witnesses to this

Algorithm for broadcasting and accepting (p, m, k):

Round k:

Phase $2k-1$: process p sends $(init, p, m, k)$ to all processes;

Phase $2k$: each process executes the following
for each $m \in \mathbf{M}$:

if received $(init, p, m, k)$ from p in phase $2k-1$
then send $(echo, p, m, k)$ to all;

if received $(echo, p, m, k)$ from at least $n-t$ distinct processes in phase $2k$
then *accept* (p, m, k);

Round $r \geq k+1$;

Phase $2r-1$, $2r$: each process executes the following:

if received $(echo, p, m, k)$ from at least $n-2t$ distinct processes in previous phases
and not sent $(echo, p, m, k)$
then send $(echo, p, m, k)$ to all;

if received $(echo, p, m, k)$ from at least $n-t$ distinct processes in this and previous phases
then *accept* (p, m, k);

Fig. 2. A broadcast primitive to simulate authenticated broadcasts

broadcast. This prevents a faulty process from claiming to have received a message that was not sent to it, and allows a correct process that accepts a message to later prove that the message was indeed sent. This primitive requires $n > 3t$ processes. If $n \leq 3t$, no broadcast primitive can provide these three properties without using signatures (this is shown in Sect. 5).

The primitive in Fig. 2 simulates the authenticated broadcast of a message m by a process p during round k of a synchronized algorithm. It uses two kinds of messages. The sender initially sends messages of type *init* to all processes (including itself). These processes now act as "witnesses" to this broadcast. Each such process then sends a message of type *echo* to all processes. A process that receives $t+1$ echoes of a message becomes a witness to the broadcast, for it knows that at most t processes are faulty and therefore there is at least one correct witness among these $t+1$ processes. A process that receives $2t+1$ echoes *accepts* the message.

The primitive can be viewed at two different levels. At one level, processes exchange high-level messages with the primitive operations *broadcast* and *accept*. A round is the interval of time taken to exchange these *logical* messages. At a lower level of the algorithm, these logical messages are implemented by the exchange of messages of type *init* and *echo*. We define a *phase* to be the interval of time it takes for a process to send such low-level messages to all processes, receive low-level messages sent by processes in the same phase, and perform some local computation. A high-level message broadcast by a correct process is accepted by all correct processes after two phases of synchronized exchange of low-level messages. Therefore, each round of the primitive is implemented by two phases of message exchange: round r is made up of phases $2r-1$ and $2r$.

To prove that this primitive provides the three properties of authentication, we first establish the following lemma.

Lemma 1. *If a correct process sends* $(echo, p, m, k)$ *then* p *must have sent* $(init, p, m, k)$ *to at least one correct process in phase* $2k-1$.

Proof. Let l be the earliest phase in which any correct process q sends $(echo, p, m, k)$. If $l > 2k$, process q must have received $(echo, p, m, k)$ messages from at least $n-2t$ distinct processes. Therefore, it must have received $(echo, p, m, k)$ from at least one correct process in phase $l-1$ or earlier. Hence, some correct process sends $(echo, p, m, k)$ before phase l, a contradiction. Therefore, $l = 2k$, and q must have received $(init, p, m, k)$ in phase $2k-1$. □

Theorem 2. *The broadcast primitive in Fig. 2 provides the correctness, unforgeability and relay properties.*

Proof

Correctness: Since p is correct, every process receives $(init, p, m, k)$ in phase $2k-1$ and every correct process sends $(echo, p, m, k)$ in phase $2k$. Hence, every process receives $(echo, p, m, k)$ from at least $n-t$ distinct processes in phase $2k$ and every correct process accepts (p, m, k) in phase $2k$, i.e., in round k.

Unforgeability: If p is correct and does not broadcast (p, m, k), it does not send any $(init, p, m, k)$ message in phase $2k-1$. If any correct process ever accepts (p, m, k), it must have received $(echo, p, m, k)$ messages from at least $n-t$ processes. Hence, at least $n-2t$ correct processes must have sent $(echo, p, m, k)$ messages. By Lemma 1, p must have sent $(init, p, m, k)$ to at least one correct process in phase $2k-1$, a contradiction.

Relay: Suppose a correct process q accepts (p, m, k) during phase i, where $i = 2r-1$ or $2r$. Process q must have received $(echo, p, m, k)$ from at least $n-t$ distinct processes by phase i. Hence, every correct process receives messages $(echo, p, m, k)$ from at least $n-2t$ distinct processes by phase i, and therefore sends

(*echo*, *p*, *m*, *k*) by phase $i+1$. Hence, every correct process receives (*echo*, *p*, *m*, *k*) from at least $n-t$ *distinct processes by phase* $i+1$ and accepts (*p*, *m*, *k*) by the end of phase $i+1$, i.e., in round $r+1$ or earlier. □

As presented here, the broadcast primitive does not terminate in a fixed number of rounds. If the broadcaster is faulty, other faulty processes can collude with the sender so that arbitrarily many rounds elapse before any correct process accepts the broadcaster's message. However, in an application that terminates in r rounds, processes stop executing the primitive at the end of round r.

In computing the message complexity of the primitive, we consider only messages sent by correct processes. It is not possible to restrict the number of messages sent by faulty processes.

Lemma 2. *The total number of messages sent by all the correct processes for each broadcast by a correct process is* $O(n^2)$.

Proof. When a correct process *p* broadcasts (*p*, *m*, *k*), each correct process broadcasts one (*echo*, *p*, *m*, *k*) message to each process. Hence, the total number of messages sent by all correct processes is $O(n^2)$. □

Although the primitive requires $O(n^2)$ messages for each broadcast by a correct process, it also provides for the automatic relay of all broadcasts accepted by correct processes. Thus, processes need never explicitly relay messages they accept. In the applications we study, this compensates for the cost of each broadcast.

To reduce the message complexity from that shown in Lemma 2, we can use the following well-known technique. We isolate a set of $3t+1$ processes called the *reflectors*, and execute the primitive with n replaced by $3t+1$ as follows: only reflectors send *echo* messages to all the other processes, other processes only receive these messages and accept messages according to the primitive. It can be verified that Theorem 2 still holds. Since there are $O(t)$ reflectors, the total number of messages sent by all correct processes for each broadcast by a correct process is $O(nt)$.

However, a broadcast of a *faulty* process, with the collusion of other faulty processes, can still cause correct processes to send $O(nt)$ messages for each round of the algorithm. We can modify the primitive to reduce the number of messages due to faulty broadcasts. This modified primitive (described in the Appendix) imposes additional restrictions on the behavior of faulty processes and thereby reduces the number of messages sent by correct processes: In an algorithm in which each correct process broadcasts at most R high-level messages, at most $O(Rnt)$ messages are sent by all correct processes for all the broadcasts of any single process.

5 A simple non-authenticated algorithm for Byzantine Agreement

The proof of the authenticated Byzantine Agreement algorithm of Fig. 1 only needs the *correctness*, *unforgeability*, and *relay* properties of authenticated broadcasts. Cryptographic techniques that provide digital signatures can be used for message authentication, and hence for the implementation of authenticated broadcasts with those properties. However, the correctness of the authenticated algorithm does not depend on this particular implementation, and any other implementation of authenticated broadcasts providing these three properties can be used instead. In Sect. 4, we described a broadcast primitive providing these three properties. Replacing signed communication in the authenticated algorithm with this primitive directly yields an equivalent non-authenticated algorithm. In addition, the *relay* property of the primitive ensures that messages accepted by correct processes are automatically relayed to all other processes. Hence, processes need not explicitly relay accepted messages.

In Fig. 3, we present the non-authenticated algorithm derived by just replacing signed communication in the authenticated algorithm of Fig. 1 with the broadcast primitive of Fig. 2. Each logical round of the algorithm corresponds to two phases of the underlying primitive.

Theorem 3. *The non-authenticated algorithm in Fig. 3 achieves Byzantine Agreement in* $2t+2$ *phases with* $O(n^2 t \log n)$ *message bits.*

```
process p:                                   /*m∈{0,1}*/
if p is the transmitter then value:=m else value:=0;
for r:=1 to t+1 do
  if value=1 and
       p has not broadcast a message in earlier rounds
    then broadcast (p, 1, r);
  if in rounds r' ≤ r, accepted (p_k, 1, r_k)
     from r distinct processes p_k, including the transmitter p_1
     then value:=1;
od
decide value.
```

Fig. 3. A non-authenticated algorithm for binary Byzantine Agreement using the broadcast primitive of Fig. 2

Proof. The proof of correctness is identical to that of the authenticated algorithm in Theorem 1.

The algorithm requires $2t+2$ phases since each round of the algorithm is implemented by two phases of the underlying primitive. Since each process broadcasts at most one message, correct processes send a total of $O(n^2 t)$ messages using the primitive in the Appendix. We assume that each message is $O(\log n)$ bits long. □

The broadcast primitive requires $n > 3t$ processes. Therefore, although the authenticated algorithm (in Fig. 1) can tolerate any number of faulty processes, the equivalent non-authenticated algorithm (in Fig. 3) requires $n > 3t$ processes. It is well-known that no non-authenticated Byzantine Agreement algorithm exists for $n \leq 3t$ (Lamport et al. 1982). Therefore, if $n \leq 3t$, no broadcast primitive can provide the properties of *correctness, unforgeability,* and *relay* without using signatures. If such a primitive existed, it could be used to convert the authenticated algorithm of Fig. 1 into a non-authenticated one for $n < 3t$ processes, contradicting the lower bound given in Lamport et al. (1982). Hence, our broadcast primitive is optimal with respect to the number of faulty processes that can be tolerated.

We now outline some simple optimizations to reduce the message and time complexity of the non-authenticated algorithm. The first optimization is to reduce the message complexity by isolating a set of $3t+1$ *active* processes (Dolev et al. 1982; Dolev and Strong 1983). The remaining processes are denoted *passive*. Only active processes broadcast messages according to the algorithm. Passive processes only accept messages, and decide on a value 1 only if they accept this message from at least $t+1$ distinct processes. Otherwise, they decide on 0. The proof of Theorem 3 can be easily extended to show the correctness of this algorithm. Since there are $O(t)$ active processes, each broadcasting at most once, we have:

Corollary 1. *Byzantine Agreement can be reached in $2t+2$ phases with $O(nt^2 \log n)$ message bits.*

It is clear that in the *authenticated* algorithm, messages signed and broadcast in the last round will not be relayed further. Hence, processes need not sign these messages. Consider a modified authenticated algorithm where processes broadcast *unsigned* messages in round $t+1$. Translating this algorithm into a non-authenticated one saves one phase: in round $t+1$, our broadcast primitive is not needed and a one-phase unsigned broadcast suffices. However, we must guarantee that a message accepted by a correct process in round t (i.e., by phase $2t$) is accepted by all correct processes by phase $2t+1$, i.e., within one additional phase. This is achieved by using a simple version of the primitive in round t, for example, that of Fig. 2. Therefore:

Corollary 2. *Byzantine Agreement can be reached in $2t+1$ phases with $O(nt^2 \log n)$ message bits.*

The communication complexity can be further reduced at the cost of an extra phase, with another well-known technique (Dolev et al. 1982; Dolev and Strong 1983). The $3t+1$ active processes described earlier run the agreement algorithm strictly among themselves, and broadcast their decision (0 or 1) directly to all the passive processes only at the end. The passive processes decide on the majority value. Since there are $O(t)$ active processes which reach agreement in $2t+1$ phases, this algorithm terminates in $2t+2$ phases and requires $O(nt + t^3 \log t)$ message bits.

Corollary 3. *Byzantine Agreement can be reached in $2t+2$ phases with $O(nt + t^3 \log t)$ message bits.*

6 Multivalued Byzantine Agreement

Binary valued Byzantine Agreement algorithms can be extended to multivalued ones using standard techniques. These schemes add a phase to the running time of the binary valued algorithms. Algorithms such as that in Dolev and Strong (1983) have been directly developed for the multivalued case and are no more expensive in time than the corresponding binary valued algorithms. Our goal is to develop a non-authenticated multivalued algorithm that requires no more phases than the binary algorithm in Fig. 3. This can be achieved by using our primitive to translate an authenticated multivalued Byzantine Agreeement algorithm (similar to that in Dolev and Strong (1983)) into a non-authenticated one.

In the authenticated multivalued algorithm, each correct process broadcasts at most 2 messages. However, faulty processes could try to broadcast more than twice each. This would result in a high message complexity for the translated algorithm, if we use the primitive of Fig. 2. Hence, to translate this authenticated algorithm to an *efficient* non-authenticated one, we must use the primitive in the Appendix.

The proofs of the authenticated and non-authenticated algorithms, written in terms of the properties of authenticated broadcasts, are identical. We

```
process p:                                   /*m∈M*/
if p is the transmitter then values:={m} else values:=∅;
for r:=1 to t+1 do
  for each m∈values that p has not broadcast in previous
      rounds do
    if p has not yet broadcast 2 distinct values
      then broadcast (p, m, r); od
  for each m∈M:
    if in rounds r'≤r, accepted (p_k, m, r_k),
      from r distinct processes p_k,
          including the transmitter p_1
      then values:=values ⋃ {m};
od
if |values|=1   then decide on the value m∈values
                else decide "sender faulty";
```

Fig. 4. A non-authenticated algorithm for multivalued Byzantine Agreement using the broadcast primitive

present only the non-authenticated algorithm (Fig. 4) and its proof of correctness.

The multivalued algorithm is a straightforward modification of the binary algorithm presented in Sect. 3. Each process maintains a set *values* of potential decision values. A process p adds m to this set in round r if it has accepted messages containing m from r distinct processes including the transmitter p_1. In the next round, p broadcasts m if it has not yet broadcast 2 distinct values. On termination, if the set *values* of process p contains exactly one element, it decides on that value. Otherwise it decides that the sender is faulty.

Theorem 4. *Multivalued Byzantine Agreement can be achieved in* $2t+2$ *phases with* $O(n^2 t(\log n + \log|\mathbf{M}|))$ *message bits using the non-authenticated algorithm in Fig. 4.*

Proof. The proof of correctness is similar to that of Theorem 1 and that in (Dolev and Strong 1983). We first prove *Validity*.

If the transmitter p_1 is correct and agreement is to be reached on the value m, then p_1 broadcasts $(p_1, m, 1)$ in round 1. By *correctness*, every correct process accepts this message in round 1, and adds m to its set *values*. The transmitter does not broadcast any other message in round 1. Hence, by *unforgeability*, the transmitter will never broadcast any *other* message in any round. Thus, by *unforgeability* no correct process can accept any message (p_1, m', r_1) for $m \neq m'$. Therefore, every correct process has $values = \{m\}$ when it stops and decides on m.

To show that *Agreement* is satisfied, we consider two cases:

(i) If a correct process p first adds m to its set *values* in round $r < t+1$, it must have accepted messages (p_k, m, r_k) from r distinct processes p_k including the transmitter p_1. Note that $p \neq p_k$, $1 \leq k \leq r$. In round $r+1$, p broadcasts $(p, m, r+1)$ if it has not already broadcast two distinct values. In that case, by *correctness* and *relay*, every correct process accepts these $r+1$ messages and adds m to *values*.

(ii) If a correct process p first adds m to its set *values* in round $t+1$, it must have accepted messages (p_k, m, r_k) from $t+1$ distinct processes p_k. At least one of these processes must be correct. This correct process must have first added m to its set *values* in round $r-1 < t+1$. By case (i), every correct process adds m to its set *values*.

Let p be the correct process whose set *values* has the maximum number of elements at the termination of the algorithm. If process p has $|values| \leq 2$, it follows from (i) and (ii) that the set *values* of every correct process contains at least the elements that p has added to its set. Since the set *values* of p is the largest, it follows that every correct process has the same elements in *values*. Therefore, all correct processes reach identical decisions. Specifically, if process p has $|values| = 1$, then all correct processes have the same element m in *values* and decide on m. If process p has $|values| = 0$ or $|values| = 2$, then all correct processes decide that the sender is faulty.

If process p has $|values| > 2$, it follows from cases (i) and (ii) that the set *values* of every correct process contains at least two of the elements that p has added to its set. Therefore, every correct process has $|values| \geq 2$ and decides that the sender is faulty. This proves that *Agreement* is satisfied.

The algorithm requires $t+1$ rounds, i.e., $2t+2$ phases of communication. Since each correct process broadcasts at most two messages, correct processes send a total of $O(n^2 t)$ messages using the broadcast primitive in the Appendix. As described in the proof of Corollary 1, this can be reduced to $O(nt^2)$ messages. Since messages are $O(\log n + \log|\mathbf{M}|)$ bits long, the total bit complexity is $O(nt^2(\log n + \log|\mathbf{M}|))$. □

7 Byzantine Elections

The problem of Byzantine Elections (Merritt 1984) involves the forecasting of election results in a synchronous network of unreliable processes. If an election is not close, these algorithms allow accurate forecasting of results in less than $t+1$ rounds.

Round 1:
Each voter signs and broadcasts its vote.

Round j, $2 \leq j \leq t+1$:
Each witness w does the following:
For every voter i:
if witness w has accepted an $i-vote$ with at least $j-2$ distinct affidavits
then the vote is valid.
Sign any new valid $i-votes$, producing a new affidavit.
broadcast every valid $i-vote$ or affidavit for a valid vote that was not broadcast by w in earlier rounds.

Decision procedure for round j, $1 \leq j \leq t+1$:
if process p has accepted exactly one $i-vote$ with at least $j-1$ distinct affidavits (including its own, if p is a witness),
then decide on the value signed as process i's vote,
else decide *error* as i's vote.

Fig. 5. Authenticated algorithm for Notarized Byzantine Elections (Merritt 1984)

Thus, although processes might not agree on the votes of individual processes until the algorithm terminates, they obtain enough information to predict the outcome of the election in fewer than $t+1$ rounds. No non-authenticated algorithm for Byzantine Elections was known. Using our broadcast primitive, we now derive one from the authenticated algorithm proposed by Merritt (1984).

Consider a system in which the processes have to agree on the votes of a set of v processes called *voters*. In this paper, we only consider the algorithm for Notarized Elections (Merritt 1984), where a set of w processes are assigned to be *witnesses*, with $w \geq 2t$. Witnesses do not themselves vote, they just sign and forward messages from the voters and other witnesses. During each round of the algorithm, each process (voter or witness) chooses a value as the vote of each voter. We assume there are $w = 2t$ witnesses and hence a total of $n = v + 2t$ processes.

The requirements of an algorithm for Byzantine Elections are:

(1) During any round j of the algorithm, there is never any disagreement on a correct process's vote.
(2) All correct processes reach Byzantine Agreement on every vote when the algorithm terminates.
(3) After round j, $1 \leq j \leq t+1$, values chosen as the votes of at most $t-j+1$ processes are different from those eventually chosen. This allows processes to arrive at a decision earlier than round $t+1$ if the election is not close.

An authenticated algorithm for Notarized Byzantine Elections from Merritt (1984) is presented in Fig. 5. A value signed by a voter i is an $i-vote$. The signature of a witness of an $i-vote$ is an *affidavit* for that *i-vote*.

The complete proof of correctness of this algorithm is found in Merritt (1984). The proof is outlined below, highlighting those portions that identify the properties of authentication required by the algorithm.

(1) If a process accepts an affidavit from a correct witness p for an $i-vote$ in round j, then every process accepts the $i-vote$ and at least t affidavits for it by round $j+1$. This follows from the fact that if a correct witness finds an $i-vote$ valid, it broadcasts an affidavit for that vote and also relays the vote and all the affidavits that caused it to find the $i-vote$ valid. Hence, by *correctness* and *relay*, every correct witness finds the $i-vote$ valid in round j, and broadcasts an affidavit.
(2) If a correct process changes its decision for some process i after round j, then at least $j-1$ witnesses are faulty. This is shown by considering the various situations in which a correct process could change its decision. In each case, either the $j-1$ witnesses that caused it to decide on a value at round j or the witnesses that cause it to later change its decision are faulty.
(3) There is never any disagreement on the vote of correct processes. A correct process i broadcasts its vote to all other processes in round 1. By *correctness*, every correct process accepts this message in round 1, and decides on this value. Since process i does not broadcast any other vote, by *unforgeability*, no correct process finds any other $i-vote$ valid.
(4) At the end of round $t+1$, correct processes agree on every vote. By (3), agreement is guaranteed on votes of correct processes. If a correct witness broadcasts an affidavit for an $i-vote$, then it also relays the $i-vote$ and all the affidavits that caused it to find the $i-vote$ valid. Hence, by *correctness* and *relay*, every process finds the $i-vote$ valid in the next round and further, every correct witness also broadcasts an affidavit for this $i-vote$. If a process finds an $i-vote$ valid at the end of round $t+1$, it must have accepted the $i-vote$ and accepted affidavits from at least t witnesses. If voter i is faulty, at least one of these witnesses is correct and hence every other correct process finds the $i-vote$ valid. From this, it can be seen that correct processes either agree on an $i-vote$ or decide on *error* as the vote.
(5) In (2), we saw that if any value is changed after round j, there are at least $j-1$ faulty wit-

Round 1:
Each voter broadcasts its vote.

Round j, $2 \leq j \leq t+1$:
Each witness w does the following:
For every voter i:
if witness w has accepted an $i-vote$ with at least $j-2$ distinct affidavits **then** the vote is valid.
broadcast an affidavit for a valid vote that was not broadcast by w in earlier rounds.

Decision procedure for round j, $1 \leq j \leq t+1$:
if process p has accepted exactly one $i-vote$ with at least $j-1$ distinct affidavits (including its own, if p is a witness),
then decide on process i's vote,
else decide *error* as i's vote.

Fig. 6. A non-authenticated algorithm for Notarized Byzantine Elections

Rules for broadcasting and accepting (p, m, k):

Round k (phase k):
process p sends $(init, p, m, k)$ to all processes.

Each process executes the following for each $m \in \mathbf{M}$:

if received $(init, p, m, k)$ from p
then send $(echo, p, m, k)$ to all;

if received $(echo, p, m, k)$ from at least $n-t$ distinct processes in previous phases
then *accept* (p, m, k);

if received $(echo, p, m, k)$ from at least $n-2t$ distinct processes in previous phases
and not sent $(echo, p, m, k)$
then send $(echo, p, m, k)$ to all;

Fig. 7. A primitive to simulate asynchronous authenticated broadcasts

nesses. Hence, at most $t-j+1$ voters are faulty, and hence correct processes always agree on the votes of the remaining (correct) voters. That is, agreement is reached on $max(0, v+j-t-1)$ votes.

Each witness relays a vote and $O(t)$ affidavits for each voter. Thus, the total number of messages exchanged is $O(nvt^2)$.

From the proof outlined above, we see that the properties of authentication required by the algorithm are the three properties described in Sect. 2. Hence, the broadcast primitive of Fig. 2 can be used in place of authenticated communication, resulting in an equivalent non-authenticated algorithm for Byzantine Elections. Once again, processes need not relay accepted votes or affidavits since the underlying primitive achieves this. The non-authenticated algorithm is described in Fig. 6. This algorithm requires a total of $n > 3t$ processes, of which at least $2t$ are designated as witnesses.

Theorem 5. *The algorithm of Fig. 6 solves the Byzantine Elections problem without authentication in* $2t+2$ *phases and* $O(nvt^2)$.

Proof. The proof of correctness is exactly the same as that of the authenticated algorithm, stated in terms of the three properties of authenticated systems.

Each witness broadcasts an affidavit for each vote of each voter, hence the total number of messages broadcast is $O(nvt^2)$, the same as that in the original authenticated algorithm. □

8 Asynchronous authenticated broadcasts

We now consider systems where communication is asynchronous. Messages sent by correct processes are eventually received by all correct processes, but this could take an arbitrarily long time. Hence, there can be no fixed bound on the duration of a phase, and the phases are not synchronized. The properties that authenticated broadcasts in asynchronous systems satisfy are therefore weaker versions of those described in Sect. 2, and can be stated as follows:

1. (*Correctness*) If correct process p broadcasts (p, m, k), then every correct process accepts (p, m, k).
2. (*Unforgeability*) If correct process p does not broadcast (p, m, k) then no correct process ever accepts (p, m, k).
3. (*Relay*) If a correct process accepts (p, m, k), then every other correct process accepts (p, m, k).

The synchronous broadcast primitive of Fig. 2 can be easily modified to derive an asynchronous primitive with the three properties described above (Fig. 7).

Theorem 6. *The primitive of Fig. 7 achieves the properties of* correctness, unforgeability, *and* relay *in asynchronous systems.*

Proof. The proof closely follows that of Theorem 2 for synchronous systems. We use the assumption that messages sent out by correct processes are eventually received by all correct processes. Details are left to the reader. □

As in the synchronous system, we can show that a broadcast by a *correct* process using the primitive of Fig. 7 causes correct processes to send a total of at most $O(n^2)$ messages. However, a *faulty* process may originate many broadcasts in each round, thus causing correct processes to send more than $O(n^2)$ messages for these broadcasts.

It is possible to modify this asynchronous primitive, so that correct processes send a total of at most $O(n^2)$ messages for all broadcasts originated by each process in each round. This modified primitive is the asynchronous version of the one in the Appendix.

In general, the asynchronous primitive of Fig. 7 can be used in developing non-authenticated algorithms that do not proceed in synchronized phases. An example of this is clock synchronization, as shown in Srikanth and Toueg (1987). Another example is presented in the next section.

9 Randomized Asynchronous Byzantine Agreement

An asynchronous randomized algorithm for Byzantine Agreement was presented by Rabin (1983). This was improved by Toueg (1984) to overcome arbitrary failures of up to a third of the processes. The latter algorithm consists of iterations with two rounds each. The first round uses authenticated broadcasts and the second round uses a non-authenticated broadcast primitive described in that paper. The algorithm, presented in Fig. 8, reaches agreement in an expected number of iterations that is a small constant independent of n and t.

In the authenticated algorithm of Fig. 8, a process p justifies the message it sends in the second round by broadcasting the list of signed messages it receives in the first round. The proof of correctness only needs the three properties of authentication provided by our asynchronous broadcast primitive. Therefore, we can convert this algorithm to a non-authenticated one by simply replacing authenticated broadcasts with the primitive of Fig. 7.

With our primitive, signed messages accepted by process p in the first round are automatically relayed to all, and therefore, p does not have to explicitly relay (broadcast) this list of accepted messages. Hence, the translation *reduces* the original communication complexity by a factor of n.

As in Rabin's algorithm (Rabin 1983), the resulting algorithm still requires that a correct dealer use encryption to distribute shares of the secret random bits to each process individually, *before* the agreement algorithm starts.

```
Process P_i: M := M_i
for k = 1 to k = R do
(* Round 1 *)
  broadcast M;
  wait to accept messages from n − t distinct processes;
  proof := set of accepted messages;
  count(1) := number of accepted messages with a value of 1;
  if count(1) ≥ n − 2t then M' := 1
    else M' := 0;
(* Round 2 *)
  echo_broadcast [M', proof];
  wait to accept [m, proof]-messages, with correct proofs,
      from n − t distinct processes;
  count(1) := number of accepted messages with m = 1;
  s_k := compute_secret(k);
  if (s_k = 0 and count(1) ≥ 1) or
      (s_k = 1 and count(1) ≥ 2t + 1) then M := 1
    else M := 0;
od
```

Fig. 8. An authenticated asynchronous binary agreement protocol (Toueg 1984)

10 Discussion

In some applications, it might be desirable to restrict processes to broadcast a single message per round of the algorithm. That is, in addition to the properties of *correctness*, *unforgeability*, and *relay*, we might be interested in achieving the following property:

4. (*Uniqueness*) If a correct process accepts (p, m, k) in round k, no correct process accepts (p, m', k) in round k where $m \neq m'$.

The broadcast primitive of Fig. 2 can be easily extended to achieve this property with the following modification: in phase $2k$, a correct process sends $(echo, p, m, k)$ only for the first *init* message it receives from process p in phase $2k-1$. All additional *init* messages from p in phase $2k-1$ are ignored. We now show that *uniqueness* holds.

Assume that two correct processes accept (p, m, k) and (p, m', k), respectively, in round k, with $m \neq m'$. Then, at least $n-t$ processes sent $(echo, p, m, k)$ and at least $n-t$ processes sent $(echo, p, m', k)$ in phase $2k$. Therefore, at least one correct process sent both $(echo, p, m, k)$ and $(echo, p, m', k)$ in phase $2k$, a contradiction.

The Crusader Agreement problem, a weaker version of the Byzantine Agreement problem, has been defined in Dolev (1982). The problem requires that, when a *transmitter* sends a message to a set of processes, the following conditions are satisfied:

(1) If the transmitter is correct, then all correct processes agree on its message.
(2) If the transmitter is faulty, all correct processes that do not decide that the transmitter is faulty agree on the same message.

A known solution to the Crusader Agreement algorithm (Dolev 1982) achieves the properties of *correctness, unforgeability* and *uniqueness*. However, it does not have the *relay* property, and there-

fore it cannot model the relaying of signed messages, a crucial property of message authentication. The first two phases of our primitive achieve Crusader Agreement.

In some authenticated algorithms such as those in Dolev and Strong (1983) and Halpern et al. (1984), a process accepting a message appends its signature to this message and then broadcasts it. When a process receives a message with a list of signatures, it can verify from these signatures that each process on the list actually broadcast the message. Non-authenticated algorithms can be derived from such algorithms in one of two ways. The first approach involves modifying the authenticated algorithm so that when a process accepts a message, it signs the message, and also relays all the messages that caused it to accept the message. Thus, messages contain exactly one signature each. We used this approach for the authenticated Byzantine Agreement algorithm in Section 3.

Another approach is to require that when a process q accepts a message of the form $m{:}p_1{:}p_2 \ldots {:}p_k$, where each p_i is the signature of a process, q considers the message to be valid only if has also accepted each prefix of the message, i.e., it has also accepted each of $m{:}p_1$, $m{:}p_1{:}p_2, \ldots,$ $m{:}p_1{:}p_2 \ldots {:}p_{k-1}$. This provides a simple and direct, although inefficient, method for deriving a non-authenticated algorithm from the authenticated one.

11 Conclusions

The design of fault-tolerant algorithms for systems with arbitrary failures is a very difficult task. These algorithms are usually complex, unintuitive, and difficult to prove correct. Assuming message authentication greatly simplifies the design of such algorithms. Moreover, authenticated algorithms are usually simpler, and easier to prove correct. However, known implementations of message authentication, such as digital signatures, incur substantial computation and communication costs. Furthermore, no such implementation is proven secure under malicious attack. In this paper, we described a methodology to reduce the difficult task of designing non-authenticated algorithms to the simpler task of designing authenticated ones.

Our methodology is based on a communication primitive that provides properties of message authentication without using cryptographic techniques. The idea is to first derive a fault-tolerant algorithm assuming that messages are authenticated. We then convert this algorithm into a non-authenticated one by substituting authenticated communication with our primitive. The resulting non-authenticated algorithm is as simple as the authenticated one from which it is derived.

In this paper, we have applied this approach to several problems. The first application gave a Byzantine Agreement algorithm requiring $2t+1$ rounds of message exchange and $O(nt^2 \log n)$ message bits. This algorithm is conceptually as simple as the original authenticated algorithm and has the same proof of correctness. The best previously known such algorithm needed $2t+3$ rounds for the same message complexity.

We also applied this methodology to the Byzantine Elections problem, and it resulted in the first known solution that does not use authentication. This solution is as simple as, and has the same message complexity as, the authenticated algorithm from which it is derived.

We then extended the primitive to asynchronous systems. We translated an authenticated randomized Byzantine Agreement algorithm (Toueg 1984) into a non-authenticated one with lower message complexity and no extra phases.

More recently, the approach outlined in this paper has been adopted to derive a simple and efficient early-stopping Byzantine Agreement algorithm (Toueg et al. 1987), and to achieve optimal clock synchronization (Srikanth and Toueg 1987).

We believe that the methodology we have described is one of the first powerful tools for understanding and developing non-authenticated fault-tolerant algorithms for systems with arbitrary failures.

Appendix

A message efficient broadcast primitive

In some algorithms, correct processes broadcast at most R broadcasts, for some R. For example, in the authenticated algorithm of Fig. 1, $R=1$, and in the one of Fig. 4, $R=2$. In this section, we modify the primitive of Fig. 2 to ensure that faulty processes do not execute more broadcasts than that required by the algorithm. This reduces the number of messages sent by witnesses and hence the overall message complexity.

We introduce additional types of messages: *init'* sent by processes in the third phase of a broadcast, and *echo'* sent by processes in the remaining phases of the broadcast. The primitive is described in Fig. 9. As before, each round r corresponds to phases $2r-1$ and $2r$ of the primitive. The value m to be broadcast can be drawn from some arbi-

Algorithm for broadcasting and accepting (p, m, k):

/* Processes execute at most R broadcasts in the algorithm in which this primitive is applied. */

Round k:

Phase $2k-1$: process p sends $(init, p, m, k)$ to all processes.

Each process executes the following for each $m \in \mathbf{M}$:
Phase $2k$:

if received $(init, p, m, k)$ from p in phase $2k-1$
and received at most R $(init, p, \_, \_)$ messages from p in all previous phases
then send $(echo, p, m, k)$ to all;

if received $(echo, p, m, k)$ from at least $n-t$ distinct processes in phase $2k$
then *accept* (p, m, k);

Round $k+1$:

Phase $2k+1$:

if received $(echo, p, m, k)$ from at least $n-2t$ distinct processes q in phase $2k$
and received at most R $(echo, p, \_, \_)$ messages from q in all previous phases
then send $(init', p, m, k)$ to all;

Phase $2k+2$:

if received $(init', p, m, k)$ from at least $n-t$ distinct processes in phase $2k+1$
then send $(echo', p, m, k)$ to all;

if received $(echo', p, m, k)$ from at least $n-t$ distinct processes in phase $2k+2$
then *accept* (p, m, k);

Round $r \geq k+2$:

Phase $2r-1, 2r$:

if received $(echo', p, m, k)$ from at least $n-2t$ distinct processes in previous phases
and not sent $(echo', p, m, k)$
then send $(echo', p, m, k)$ to all;

if received $(echo', p, m, k)$ from at least $n-t$ distinct processes in this and previous phases
then *accept* (p, m, k);

Fig. 9. A primitive that permits up to R authenticated broadcasts per process

trary set **M**. In what follows, message fields marked by an "$\_$" (underscore) can contain arbitrary values. For example, $(\_, p, \_, \_)$ refers to all messages sent by process p, and $(echo, p, \_, \_)$ refers to all messages of type *echo* sent by p.

Lemma 3. *If a correct process ever sends* $(echo', p, m, k)$, *then at least one correct process must have sent* $(echo', p, m, k)$ *in phase* $2k+2$.

Proof. Let l be the earliest phase in which any correct process q sends $(echo', p, m, k)$. If $l \geq 2k+3$, process q must have received $(echo', p, m, k)$ messages from at least $n-2t$ distinct processes, i.e., it must have received $(echo', p, m, k)$ from at least one correct process in phase $l-1$ or earlier. Hence, some correct process sends $(echo', p, m, k)$ before phase l, a contradiction. Therefore, $l = 2k+2$.

Lemma 4. *If a correct process ever sends* $(echo', p, m, k)$, *then* p *must have sent* $(init, p, m, k)$ *to at least one correct process in phase* $2k-1$.

Proof. By Lemma 3, if a correct process ever sends $(echo', p, m, k)$, then some correct process q must have sent $(echo', p, m, k)$ in phase $2k+2$. Therefore, process q must have received $(init', p, m, k)$ from at least $n-t$ processes in phase $2k+1$. At least $n-2t$ of these processes are correct, and each of them must have received at least $n-2t$ $(echo, p, m, k)$ messages in phase $2k$. Hence, at least one correct process must have sent $(echo, p, m, k)$ in phase $2k$ and it must have received $(init, p, m, k)$ from p in phase $2k-1$. □

Theorem 7. *The broadcast primitive in Fig. 9 has the properties of* correctness, unforgeability, *and* relay.

Proof

Correctness: Since p is correct, every correct process receives $(init, p, m, k)$ in phase $2k-1$ and sends $(echo, p, m, k)$ in phase $2k$. Hence, every process receives $(echo, p, m, k)$ from at least $n-t$ correct processes in phase $2k$ and every correct process accepts (p, m, k) at the end of phase $2k$, i.e. at the end of round k.

Unforgeability: If p is correct and does not execute broadcast (p, m, k), it does not send any message $(init, p, m, k)$ in phase $2k-1$, and no correct process sends $(echo, p, m, k)$ in phase $2k$. Hence, no correct process can accept (p, m, k) in phase $2k$. If some correct process accepts (p, m, k) at the end of phase $2k+2$ or later, it must have received $(echo', p, m, k)$ messages from at least $n-t$ distinct processes, i.e., it must have received $(echo', p, m, k)$ from at least $n-2t$ correct processes. By Lemma 4, p must have sent $(init, p, m, k)$ to at least one correct process in phase $2k-1$, a contradiction. Thus, no correct process ever accepts (p, m, k).

Relay: Let q be a correct process that accepts (p, m, k) in phase i where $i = 2r-1$ or $2r$. If $r = k$, then q must have received $(echo, p, m, k)$ from at least $n-t$ distinct processes in phase $2k$. Hence, in the same phase, every correct process receives $(echo, p, m, k)$ from at least $n-2t$ distinct processes and sends $(init', p, m, k)$ in phase $2k+1$. Therefore, every correct process sends $(echo', p, m, k)$ during phase $2k+2$ and every correct pro-

cess accepts (p, m, k) at the end of phase $2k+2$, i.e., round $r+1$. Now. suppose $r \geq k+1$. Process q must have received $(echo', p, m, k)$ from at least $n-t$ distinct processes by phase i. Hence, every correct process receives $(echo', p, m, k)$ from at least $n-2t$ distinct processes by phase i, and therefore sends $(echo', p, m, k)$ at or before phase $i+1$. Thus, every correct process receives $(echo', p, m, k)$ from at least $n-t$ distinct processes by phase $i+1$, and therefore accepts (p, m, k) by the end of phase $i+1$, i.e., in round $r+1$ or earlier. □

As before, in computing the message complexity of the primitive, we only consider messages sent by correct processes.

Lemma 5. *In an algorithm in which the primitive of Fig. 9 is applied, each correct process sends a total of* $O(Rn)$ $(\_, p, \_, \_)$ *messages for each process* p.

Proof. Consider a given process p. Each correct process p_i accepts at most R $(init, p, \_, \_)$ messages from p and thus sends at most R $(echo, p, \_, \_)$ messages to each process. p_i considers at most R $(echo, p, \_, \_)$ messages from each other process p_j throughout the algorithm. Since p_i sends $(init', p, m, k)$ only on receiving at least $n-2t$ $(echo, p, m, k)$ messages, and since $n > 3t$, each correct process sends at most $Rn/(n-2t) \leq 3R$ $(init', p, \_, \_)$ messages.

We now show that at most 6R distinct $(echo', p, \_, \_)$ messages are sent (to all) by the set of correct processes (and hence by each individual correct process). Let C be a set of $n-t$ correct processes. By Lemma 3, if a correct process sends $(echo', p, m, k)$ in any phase, then some correct process q must have seent $(echo', p, m, k)$ in phase $2k+2$. For correct process q to send $(echo', p, m, k)$ in phase $2k+2$, it must receive $(init', p, m, k)$ messages from at least $n-t$ processes, i.e., from at least $n-2t$ correct processes in C. Therefore, the sending of an $(echo', p, m, k)$ by a correct process requires the sending of $(init', p, m, k)$ by $n-2t$ correct processes in C to all processes, i.e., a total of $(n-2t)n$ such messages sent by processes in C. Since each correct process sends at most $3R$ $(init', p, \_, \_)$ messages to all processes, then processes in C send a total of at most $3R(n-t)n$ $(init', p, \_, \_)$ messages. Hence, the total number of distinct $(echo', p, \_, \_)$ sent by the set of correct processes is bounded by $3R(n-t)n/(n-2t)n \leq 6R$. Thus, each correct process sends at most $6R$ $(echo', p, \_, \_)$ messages.

Therefore, each correct process sends at most 10R $(\_, p, \_, \_)$ messages to all processes, i.e., a total of $O(n)$ $(\_, p, \_, \_)$ messages. □

Corollary 3. *For each process p, the total number of* $(\_, p, \_, \_)$ *messages sent by all correct processes throughout an algorithm is* $O(Rn^2)$.

As in Sect. 4, we reduce the message complexity by isolating a set of $3t+1$ processes called the *reflectors*, and execute the primitive with $n = 3t+1$ as follows: only reflectors send *init'*, *echo* and *echo'* messages to all the other processes and other processes only receive these messages and accept messages according to the primitive. It can be verified that the proofs of Theorem 7 and Lemmas 3 and 4 still hold. Since there are $O(t)$ reflectors, by Lemma 5, the total number of $(\_, p, \_, \_)$ messages sent by all correct processes for each process p is $O(Rnt)$.

Acknowledgements. We would like to thank Özalp Babaoğlu, John Gilbert and Fred Schneider for their helpful comments and suggestions on earlier drafts of this paper.

References

Coan AB (1986) A communication-efficient canonical form for fault-tolerant distributed protocols. Proc Fifth Annu ACM Symp on Principles of Distributed Computing, Calgary, Canada (August 1986), pp. 63–72

Dolev D (1982) The Byzantine Generals Strike again. J Algorithms 3(1):14–30

Dolev D, Strong HR (1982) Polynomial algorithms for multiple process agreement. Proc 14th Annual ACM Symp Theory Comput, San Francisco, California, (May 1982), pp 404–407

Dolev D, Strong HR (1983) Authenticated algorithms for Byzantine Agreement. SIAM J Comput 12(4):656–666

Dolev D, Fischer MJ, Fowler R, Lynch NA, Strong HR (1982) An efficient algorithm for Byzantine Agreement without authentication, Inf Control, vol 52, no 3

Fischer MJ (1983) The consensus problem in unreliable distributed systems (A Brief Survey), YALEU/DCS/RR-273 (June 1983)

Garcia-Molina H, Pittelli F, Davidson S (1984) Applications of Byzantine Agreement in database systems. Tech Rep TR 316, Princeton University (June 1984)

Halpern JY, Strong HR, Dolev D (1984) Fault-tolerant clock synchronization, Proc Third Annual ACM Symp Principles of Distributed Computing, Vancouver, Canada, (August 1984) pp 89–102

Lamport L, Shostak R, Pease M (1982) The Byzantine Generals problem. ACM Trans Program Lang Syst 4:382–401

Lundelius J, Lynch N (1984) A new fault-tolerant algorithm for clock synchronization. Proc Third Annual ACM Symp on Principles of Distributed Computing, Vancouver, Canada (August 1984), pp 75–88

Lynch N, Fischer M, Fowler R (1982) A simple and efficient Byzantine Generals algorithm. Proc Second IEEE Symp Reliability in Distributed Software and Data Base Systems, Pittsburgh, Pennsylvania, pp 46–52

Merritt M (1984) Elections in the presence of faults. Proc 3rd Symp Principles of Distributed Computing, Vancouver, Canada

Mohan C, Strong HR, Filkenstein S (1983) Method for distributed transaction commit and recovery using Byzantine

Agreement within clusters of processors. Proc 2nd Symp Principles of Distributed Computing (August 1983). Montreal, Canada, pp 89–103

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

Rabin M (1983) Randomized Byzantine generals. Proc 24th Symp Foundations of Computer Science, Tucson, Arizona (November 1983) pp 403–409

Rivest RL, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. Commun ACM 21(2):120–126

Srikanth TK, Toueg S (1987) Optimal clock synchronization. Proc 4th Symp Principles of Distributed Computing, Minaki, Canada (August 1985). To appear in the Journal of the ACM (July 1987)

Tanenbaum AS (1981) Computer networks. Prentice-Hall software series

Toueg S (1984) Randomized asynchronous Byzantine Agreements. Proc 3rd Symp Principles of Distributed Computing, Vancouver, Canada (August 1984)

Toueg S, Perry KJ, Srikanth TK (1987) Fast distributed agreement. Proc 4th Symp Principles of Distributed Computing, Minaki, Canada (August 1985). Also appeared in SIAM J Comput Vol. 16, No. 3, June 1987