# Simulating Authenticated Broadcasts to Derive Simple Fault-Tolerant Algorithms[†]

T.K. Srikanth
Sam Toueg

TR 84-623
July 1984
(April 1985)

Department of Computer Science
Cornell University
Ithaca.

To appear in : "Distributed Computing" (Springer & Verlag)

# Simulating authenticated broadcasts to derive simple fault-tolerant algorithms†

*T.K. Srikanth*
*Sam Toueg*

Cornell University
Ithaca, New York 14853

**ABSTRACT**

Fault-tolerant algorithms for distributed systems 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 simulates 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.

## 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 [Moha83, Garc84]. Formally, Byzantine Agreement requires that when a message is sent by a transmitter to a set of processes,

(1) All correct processes agree on the same message,
(2) 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 algorithm; it is *faulty* otherwise. Correct processes must reach agreement on a message $m \in \mathbf{M} \cup$ {"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

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

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 [Lamp82, Dole83, Merr84, etc]. 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.

We first consider synchronized algorithms to achieve Byzantine Agreement. These algorithms proceed in synchronized *phases* where processes first send messages (according to their states), then 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 [Dole82b]. 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 [Dole82a]. 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 [Rive78]. 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.

Our goal is to exploit some of the advantages of authentication without paying the price of digital signatures. In this paper, we present a methodology for deriving non-authenticated algorithms. We start with an authenticated algorithm and identify the properties of authentication it needs. We then derive a broadcast primitive that has these properties without using signatures. Finally, we replace signed communication in the authenticated algorithm with this broadcast primitive to get an equivalent non-authenticated algorithm. Hence, the resulting algorithm is as simple as the original authenticated algorithm, and furthermore its correctness follows directly from that of the authenticated algorithm.

Previous work has provided different and more or less unrelated solutions for the authenticated and non-authenticated versions of a problem. For example, the simple authenticated algorithm by Dolev and Strong [Dole82a] did not seem to help solve the non-authenticated version of the problem [Dole82b]. Other examples are the problems of Byzantine Elections [Merr84] and clock synchronization [Halp84, Lund84]. Our approach is the first that unifies solutions for the authenticated and non-authenticated models.

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 $O(nt^2 \log n)$ bits. Further, this algorithm is as simple as the authenticated algorithm in [Dole82a] from which it is derived.

We then consider the Byzantine Elections problem. This problem was formulated recently by Merritt and was solved assuming message authentication [Merr84]. 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 [Toue84a]. 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 [Toue84b], and to derive algorithms for synchronizing clocks [Srik84].

## 2. Properties of authenticated broadcasts

Consider an algorithm that proceeds in synchronous rounds† 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 other processes. 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 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 correct process $p$ does not broadcast $(p,m,k)$, then no correct process ever accepts $(p,m,k)$.

A process accepting a signed message in a certain round can relay this message to all other processes and thus ensure that all processes accept this message by the next round. Therefore, if processes immediately relay every message they accept, then we have 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.

By Property 1, broadcasts 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 $r \geq k$.

† For the present, a round is a phase.

It should be noted that 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 malicious and given its signature to another process. This process could then originate $m$ "signed by $p$" [Lamp82]. Therefore, a process that accepts $(p,m,k)$ can only infer that, *if $p$ is correct*, then $p$ is the originator of the message (Property 2).

## 3. An algorithm using authenticated broadcasts

We now consider an authenticated algorithm for Byzantine Agreement. The algorithm presented in Figure 1 is a simple modification of that in [Dole83]. For simplicity, round numbers are omitted from messages. In this algorithm, a process *extracts* some messages from among the messages it receives. Informally, a process extracts a message $m$ if it believes that the transmitter indeed sent $m$ and if it can convince others about this.

**Theorem 1**: Byzantine Agreement can be achieved in $t+1$ phases with $O(n^2 t \log n)$ message bits using the authenticated algorithm in Figure 1.

*Proof*: The proof of correctness of the algorithm of Figure 1 is similar to that in [Dole83] and is outlined below.

Case 1: The transmitter $s$ is correct. By Property 1, every correct process accepts $(s,m)$ in round 1 and extracts $m$. The transmitter, being correct, does not broadcast any other message, and by Property 2, no correct process accepts any other

---

**Round** 1: the transmitter $s$ broadcasts $(s,m)$;

**Round** $i+1$, $i=1,2, \ldots ,t$: each process $q$:

    **if** accepted $(p,m)$ from at least $i$ distinct processes $p$,
        including the transmitter $s$, in rounds 1 to $i$
    **then** extract $m$;

    **if** $m$ is the first or second message extracted
        **and** $m$ has not been extracted before
    **then** broadcast $(q,m)$ and
        relay all the $i$ messages that caused $m$ to be extracted;

**Decision**:

    **if** extracted exactly one message $m$ **then** decide $m$
    **else** decide "sender faulty".

Figure 1. An authenticated algorithm for Byzantine Agreement using signed broadcasts.

---

message from $s$. Hence, every correct process extracts only the message $m$ and decides $m$.

Case 2: The transmitter is faulty.

(i) If some correct process $q$ extracts $m$ at the end of round $i<t+1$, then it must have accepted messages $(p,m)$ from $i$ distinct processes including the transmitter $s$. If it has extracted fewer than two messages so far, $q$ broadcasts $(q,m)$ in round $i+1$. By Properties 1 and 3, every correct process accepts $(q,m)$ and all the $i$ messages $(p,m)$ by round $i+1$ and hence extracts $m$ if it has not already done so.

(ii) If some correct process $q$ extracts $m$ at the end of round $t+1$, it must have accepted $(p,m)$ from at least $t+1$ distinct processes. At least one of these processes must be correct, must have extracted $m$ in round $i<t+1$, and broadcast $m$ in round $i+1$. From (i), every correct process extracts $m$ by the end of round $t+1$.

Therefore, either every correct process extracts the same message $m$ and decides $m$, or all correct processes extract more than one message each and decide that the sender is faulty.

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

## 4. An implementation of authenticated broadcasts

The proof of the authenticated algorithm of Figure 1 shows that Properties 1, 2 and 3 are the only properties of authenticated broadcasts that this algorithm needs. One way to provide these three properties is to implement authentication using digital signatures. However, the correctness of the authenticated algorithm does not depend on this particular implementation, and any other implementation providing these properties could be used instead. In this section, we describe a broadcast primitive that achieves these three properties without using signatures. This primitive requires that $n>3t$. We later show that if $n\leq 3t$, then no broadcast primitive can provide the three properties without using signatures.

The primitive is a modification of the Echo primitive presented in [Brac83, Toue84a]. Two kinds of messages are used: *init* sent out initially by the sender to all other processes, and *echo* sent by the other processes according to the algorithm described below. When a process receives sufficient echoes of a message it *accepts* the message. The primitive in Figure 2 simulates the authenticated broadcast of a message $m$ by a process $p$ during round $k$ of a synchronized algorithm.

With this primitive, it takes two phases of synchronized message exchange for a broadcast by a correct process to be accepted. Therefore, each round of the authenticated algorithm requires two phases of the primitive: round $r$ corresponds to phases $2r-1$ and $2r$ of the primitive.

Rules 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 any $m \in \mathbf{M}$:

**if** received $(init,p,m,k)$ from $p$ in phase $2k-1$
**and** received only one *init* message 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 for any $m \in \mathbf{M}$:

**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 previous phases
**then** *accept* $(p,m,k)$;

Figure 2. A broadcast primitive to simulate authenticated broadcasts.

Before describing the properties of this primitive, we prove 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 $l=2k$, $q$ must have received $(init,p,m,k)$ in phase $2k-1$. 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. □

**Theorem 2**: The broadcast primitive in Figure 2 has the properties of *correctness*, *unforgeability* and *relay*, and in addition 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 $m \neq m'$.

*Proof*:

*Property 1 (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 $k$.

*Property 2 (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 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.

*Property 3 (Relay)*: Let $q$ have accepted $(p,m,k)$ 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.

*Property 4 (Uniqueness)*: 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 $2k$. This implies that at least one correct process sent both $(echo,p,m,k)$ and $(echo,p,m',k)$ in phase $2k$, a contradiction. □

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 processes accepts the 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 $p$ is $O(n^2)$.

*Proof*: Since each correct process sends an $(echo,p,m,k)$ messages to all the processes when $p$ broadcasts $(p,m,k)$, the total number of messages sent by all correct processes is $O(n^2)$. □

To reduce the message complexity, we isolate a set of $3t+1$ processes called the *reflectors*, and execute the primitive with $n=3t+1$ 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 the properties of Theorem 2 still hold. Since there are $O(t)$ reflectors, the total number of messages sent by all correct processes for each broadcast by a process $p$ is $O(nt)$.

However, each faulty process, in collusion with other faulty processes, can cause correct processes to send $O(n^2)$ messages for each round of the algorithm. The number of messages can be reduced by using the modified broadcast primitive described in the Appendix.

## 5. A simple non-authenticated algorithm for Byzantine Agreement

The broadcast primitive of Figure 2 achieves all the properties of authenticated broadcasts required for the correctness of the authenticated algorithm of Figure 1. Replacing signed communication with this primitive directly yields an equivalent non-authenticated algorithm. In addition, the *relay* property of the primitive ensures that messages that are accepted by correct processes are automatically relayed to all other processes. Hence, processes need not relay accepted messages. The non-authenticated algorithm is presented in Figure 3. As described earlier, each logical round of the algorithm corresponds to two phases of the underlying primitive.

**Theorem 3**: Byzantine Agreement can be achieved in $2t+2$ phases with $O(n^2 t \log n)$ message bits using the non-authenticated algorithm in Figure 3.

*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 correct process executes a maximum of two broadcasts, correct processes send a total of $O(n^2 t)$ messages if the primitive described in the Appendix is used. Assuming that each message is $O(\log n)$ bits long, correct processes exchange a total of $O(n^2 t \log n)$ bits. □

---

**Round** 1: the transmitter $s$ broadcasts $(s,m)$;

**Round** $i+1$, $i=1,2,\ldots,t$: each process $q$:

> **if** accepted $(p,m)$ for at least $i$ distinct processes $p$,
> including the transmitter $s$, in rounds 1 to $i$
> **then** extract $m$;
>
> **if** $m$ is the first or second message extracted
> **and** $m$ has not been extracted before
> **then** broadcast $(q,m)$;

**Decision**:

> **if** extracted exactly one message $m$ **then** decide $m$
> **else** decide "sender faulty".

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

---

Although the authenticated algorithm (in Figure 1) can handle any number of faulty processes, the broadcast primitive requires $n > 3t$ processes, and therefore, the equivalent non-authenticated algorithm has the same requirement. It is well-known that no non-authenticated Byzantine Agreement algorithm exists for $n \leq 3t$ [Lamp82]. Therefore, if $n \leq 3t$, no broadcast primitive can provide Properties 1 to 3 without using signatures. 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 [Dole82b, Dole83]. The remaining processes are denoted *passive*. Only active processes broadcast messages according to the algorithm. Passive processes only accept messages, and extract a value $m$ if they accept this message from $t+1$ distinct processes. The decision procedure for passive processes is the same as that for active ones. 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 twice, we have the following result:

**Corollary 1**: Byzantine Agreement can be achieved in $2t+2$ phases using $O(nt^2 \log n)$ bits.

It is clear that in the authenticated algorithm, the messages broadcast in the last round will not be relayed further. Hence, processes need not sign these messages. Therefore, they can simply broadcast unsigned messages in round $t+1$ of the non-authenticated algorithm, thus saving one phase. In round $t$, we use a simple modification of the broadcast primitive so that if a correct process accepts a message in round $t$, every other correct process does so by the next phase. Thus we have the following result:

**Corollary 2**: Byzantine Agreement can be achieved in $2t+1$ phases using $O(nt^2 \log n)$ bits.

The communication complexity can be further reduced at the cost of an extra phase, with another well-known technique [Dole82b, Dole83]. The $3t+1$ active processes described earlier run the agreement algorithm strictly among themselves, and broadcast their decision 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.

## 6. Byzantine Elections

The problem of Byzantine Elections [Merr84] 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. 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 Election was known. In this section, we derive one from the authenticated algorithm proposed by Merritt [Merr84] using our broadcast primitive.

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 [Merr84], 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) choses a value as the vote of each voter. We assume there are 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 [Merr84] is presented in Figure 4. 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 [Merr84]. 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 Properties 1 and 3, 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 property 2, every correct process accepts this message in round 1, and decides on this value. Since process $i$ does not broadcast any other vote, by Property 2, 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 Properties 1 and 3,

**Round** 1:
Each voter signs and broadcasts its vote.

**Round** $j$ , $2 \le j \le t+1$:
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 $j$, $1 \le j \le t+1$:
**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.

Figure 4. Authenticated algorithm for Notarized Byzantine Elections [Merr84].

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 process 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 witnesses. 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 Section 2. Hence, the broadcast primitive of Figure 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 Figure 5. This algorithm requires a total of $n>3t$ processes, of which at least $2t$ are designated as witnesses.

**Round** 1:
Each voter broadcasts its vote.

**Round** j , $2 \le j \le t+1$:
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, $1 \le j \le t+1$:
**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.

Figure 5. A non-authenticated algorithm for Notarized Byzantine Elections.

**Theorem 4**: Byzantine Elections without authentication can be solved in $t+1$ rounds or $2t+2$ phases with $O(nvt^2)$ messages with the algorithm of Figure 5.

*Proof* : The proof of correctness is exactly the same as that of the authenticated algorithm [Merr84], 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. □

## 7. 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 of processes are not synchronized. The properties that authenticated broadcasts in asynchronous systems satisfy are therefore weaker versions of those described in Section 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 $(p,m,k)$.

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 any $m \in \mathbf{M}$:

**if** received $(init,p,m,k)$ from $p$ **and** received only one $(init,p,\_,k)$ 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;

Figure 6. A primitive to simulate asynchronous authenticated broadcasts.

3. *(Relay)* If a correct process accepts $(p,m,k)$, then every other correct process accepts $(p,m,k)$.

The synchronous broadcast primitive of Figure 2 can be easily modified to derive an asynchronous primitive with the three properties described above (Figure 6).

**Theorem 5**: The primitive of Figure 6 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 our assumption that messages sent out by correct processes are eventually received by all correct processes. □

As in the synchronous system, we can show that a broadcast by a correct process using the primitive of Figure 6 requires $O(n^2)$ messages. However, in each round, each faulty process can again cause each correct process to send $O(n \cdot min(t,|\mathbf{M}|))$ messages. It is possible to modify this asynchronous primitive, so that each correct process sends only $O(n)$ messages for each process in each round.

In general, the asynchronous primitive of Figure 6 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 [Srik84]. Another example is presented in the next section.

## 8. Randomized Byzantine Agreement

A randomized algorithm for Byzantine Agreement was presented by Rabin [Rabi83]. This was improved by Toueg [Toue84a] to overcome malicious failures of up to a third of the processes. The latter algorithm consists of iterations with two rounds. The first round uses authenticated broadcasts and the second round uses a non-authenticated broadcast primitive described in that paper. The algorithm is presented in Figure 7. The algorithm reaches agreement in an expected number of iterations that is a small constant independent of $n$ and $t$.

In the original algorithm, 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. From the proof of correctness of the algorithm [Toue84a], we see that the only properties of authentication needed by this algorithm are the three properties that are described in Section 7 and are provided by our asynchronous broadcast primitive. Therefore, we can translate this authenticated algorithm to a non-authenticated one by simply replacing the direct authenticated broadcast used in the first round with the primitive of Figure 6.

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 [Rabi83], the resulting algorithm still requires that a correct dealer use encryption to distribute shares of the secret random bits to each

---

Process $P_i$: $M := M_i$

**for** $k=1$ **to** $k=R$ **do**

(* Round 1 *)

  **broadcast** $M$;
  wait to accept $M$-messages from $n-t$ distinct processes;
  proof := set of accepted messages;
  count(1) := number of accepted messages with $M = 1$;
  **if** count(1) $\geq n-2t$ **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) $\geq 1$) **or** ($s_k = 1$ and count(1) $\geq 2t+1$) **then** $M := 1$
    **else** $M := 0$;

**od**

Figure 7. An authenticated asynchronous binary agreement protocol [Toue84a].

---

process individually, *before* the agreement algorithm starts.

## 9. Discussion

The Crusader Agreement problem, a weaker version of the Byzantine Agreement problem, has been defined in [Dole82c]. The Crusader Agreement algorithm proposed in [Dole82c] achieves the properties of *correctness*, *unforgeability* and *uniqueness*. However, this solution does not have the *relay* property, and therefore it cannot model the relaying of signed messages which is crucial in simulating authentication. The first two phases of our primitive achieve Crusader Agreement.

In some authenticated algorithms such as those in [Dole83, Halp84], 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 $m{:}p_1{:}p_2 \cdots {:}p_k$, 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 $m{:}p_1$, $m{:}p_1{:}p_2, \ldots, m{:}p_1{:}p_2 \cdots {:}p_{k-1}$. This provides a simple and direct, although inefficient, method for deriving a non-authenticated algorithm from the authenticated one.

## 10. Conclusions

Non-authenticated algorithms for systems with arbitrary failures are generally regarded as complicated, unintuitive, and difficult to derive and prove correct. On the other hand, authenticated algorithms are usually much simpler and easier to derive. However, there are disadvantages to known cryptographic implementations of authentication. Furthermore, developing authenticated solutions did not generally help in deriving non-authenticated solutions.

To take advantage of the simplicity of authenticated algorithms without incurring the disadvantages of digital signatures, we developed a broadcast primitive that achieves the properties of authenticated broadcasts. Using this broadcast primitive, we presented a methodology for deriving non-authenticated algorithms. Informally, one starts by developing an authenticated algorithm. Replacing authenticated communication in this algorithm with the primitive gives an equivalent non-authenticated algorithm.

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 then 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 [Toue84a] to produce a non-authenticated algorithm 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 [Toue84b], and to achieve optimal clock synchronization [Srik84].

We believe that the methodology we have described is one of the first powerful tools for understanding and developing non-authenticated algorithms. The properties of the broadcast primitive effectively restrict the visible behavior of faulty processes, and therefore simplify the problem of designing algorithms for systems with arbitrary types of failures.

## Acknowledgements

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

## References

Brac83 G. Bracha and S. Toueg, Resilient consensus protocols, *Proc. 2nd Symposium on the Principles of Distributed Computing*, Montreal, Canada, pp. 12-26, Aug. 1983.

Dole82a D. Dolev and R. Strong, Polynomial algorithms for multiple process agreement, *Proc. 14th Annual ACM Symposium on Theory of Computing*, San Francisco, California, pp. 404-407, May 1982.

Dole82b D. Dolev, M.J. Fischer, R. Fowler, N.A. Lynch, H.R. Strong, An efficient Byzantine Algorithm without authentication, Technical Report RJ3428, IBM, Mar. 82.

Dole82c D. Dolev, The Byzantine Generals Strike again, *J. Algorithms* 3,1 (1982), pp. 14-30.

Dole83 D. Dolev and H. R. Strong, Authenticated algorithms for Byzantine Agreement, *SIAM J. Comput.*, vol. 12, no. 4, pp. 656-666, Nov. 1983.

Fisc83 M.J. Fischer, The consensus problem in unreliable distributed systems (A Brief Survey), YALEU/DCS/RR-273, June 1983.

Garc83 H. Garcia-Molina, F. Pittelli, and S. Davidson, Applications of Byzantine Agreement in database systems, Technical Report TR 316, Princeton University, June 1984.

Halp84 J.Y. Halpern, R. Strong, and D. Dolev, Fault-tolerant clock synchronization, *Proceedings Third Annual ACM Symposium on Principles of Distributed Computing*, Vancouver, Canada, pp. 89-102, Aug. 1984.

Lamp82 L. Lamport, R. Shostak, and M. Pease, The Byzantine Generals problem, *ACM Transactions on Programming Languages and Systems 4,* pp. 382-401, 1982.

Lund84 J. Lundelius and N. Lynch, A new fault-tolerant algorithm for clock synchronization, *Proceedings Third Annual ACM Symposium on Principles of Distributed Computing*, Vancouver, Canada, pp. 75-88, Aug. 1984.

Lync82 N. Lynch, M. Fischer, and R. Fowler, A simple and efficient Byzantine Generals algorithm, *Proc. Second IEEE Symposium on Reliability in Distributed Software and Data Base Systems*, Pittsburgh, Pennsylvania, pp. 46-52, 1982.

Merr84 M. Merritt, Elections in the presence of faults, *Proc. 3rd Symposium on the Principles of Distributed Computing*, Vancouver, Canada, Aug. 1984.

Moha83 C. Mohan, H.R. Strong, and S. Filkenstein, Method for distributed transaction commit and recovery using Byzantine Agreement within clusters of processors, *Proc. 2nd Symposium on the Principles of Distributed Computing*, Montreal, Canada, pp.89-103, Aug. 1983.

Peas80 M. Pease, R. Shostak and L. Lamport, Reaching agreement in the presence of faults, *Journal of the ACM*, vol. 27, no. 2, pp. 228-234, April 1980.

Rabi83 M. Rabin, Randomized Byzantine generals, *Proc. 24th Symposium on Foundations of Computer Science*, Tucson, Arizona, pp. 403-409, Nov. 1983.

Rive78 R.L. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM*, *21*, 2, pp. 120-126, Feb. 1978

Srik84 T.K. Srikanth and S. Toueg, Optimal Clock Synchronization, Technical Report TR 84-656, Cornell University, Dec. 1984. To appear in the *Proc. 4th Symposium on the Principles of Distributed Computing*, Minaki, Canada, Aug. 1985.

Toue84a S. Toueg, Randomized asynchronous Byzantine Agreements, *Proc. 3rd Symposium on the Principles of Distributed Computing*, Vancouver, Canada, Aug. 1984.

Toue84b S. Toueg, K.J. Perry and T.K. Srikanth, Fast Distributed Agreement, Technical Report TR 84-621, Cornell University, July 1984. To appear in the *Proc. 4th Symposium on the Principles of Distributed Computing*, Minaki, Canada, Aug. 1985.

## Appendix

In this appendix, we modify the primitive of Section 4 to restrict the number of broadcasts executed by faulty processes to that dictated by the algorithm. Specifically, if correct processes execute at most $R$ broadcasts throughout an algorithm, then by applying this primitive, faulty processes are also restricted to executing $O(R)$ broadcasts. For example, in the authenticated algorithm of Figure 1, $R=2$. 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 Figure 8. As before, each round $r$ corresponds to phases $2r-1$ and $2r$ of the primitive.

**Lemma 3**: 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 $l=2k+2$, $q$ 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 $2k-1$. 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. □

**Theorem 6**: The broadcast primitive in Figure 8 has the properties described in Section 2, namely *correctness*, *unforgeability*, *relay*, and *uniqueness*.

*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 3, $p$ 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*: 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 process accepts $(p,m,k)$ at the end of phase $2k+2$, i.e., round $r+1$. If $r \geq k+1$, let $q$ accept $(p,m,k)$ in phase $i$ where $i=2r-1$ or $2r$. $q$ 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 $i+1$.

Rules 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 any $m \in \mathbf{M}$:
*Phase* $2k$:

**if** received $(init,p,m,k)$ from $p$ in phase $2k-1$
**and** received only one $(init,p,\_,k)$ message in phase $2k-1$
**and** received at most $R$ $(init,p,\_,\_)$ messages 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 only one $(echo,p,\_,k)$ message from each such $q$ 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 previous phases
**then** *accept* $(p,m,k)$;

Figure 8. A primitive that permits up to $R$ authenticated broadcasts per process.

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.

*(Uniqueness)*: 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 $2k$. This implies that at least one correct process sent both $(echo,p,m,k)$ and $(echo,p,m',k)$ in phase $2k$, a contradiction. □

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

**Lemma 4**: In an algorithm in which the primitive of Figure 8 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. A process $p_i$ considers at most one $(echo,p,\_,k)$ message from each other process $p_j$ for each $k$ and at most $R$ such messages 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 $n \geq 3t+1$, each correct process sends at most $\lfloor Rn/(n-2t) \rfloor \leq 2R$ $(init',p,\_,\_)$ messages. For a correct process to send a $(echo',p,m,k)$ message 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. Since each correct process sends at most $2R$ $(init',p,\_,\_)$ messages, there can be at most $\lfloor 2R(n-t)/(n-2t) \rfloor \leq 3R$ distinct $(echo',p,\_,\_)$ messages sent by each correct process. Hence, each correct process sends at most $6R$ $(\_,p,\_,\_)$ messages to all the processes. That is, each correct process sends $O(Rn)$ $(\_,p,\_,\_)$ messages. □

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

To reduce the message complexity further, we isolate 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 properties of Theorem 6 and Lemma 3 still hold. Since there are $O(t)$ reflectors, by Lemma 3, the total number of $(\_,p,\_,\_)$ messages sent by all correct processes for each process $p$ is $O(nt)$.