From Dependable Computing for Critical Applications–5, Champaign, IL, September 1995, pp. 139–157; Volume 10 of the series in Dependable Computing and Fault Tolerant Systems published by IEEE Computer Society Press.

Byzantine Agreement with Authentication: Observations and Applications in Tolerating Hybrid and Link Faults*

Li Gong†, Patrick Lincoln, and John Rushby
Computer Science Laboratory
SRI International
Menlo Park, California 94025, USA

Abstract

We show that the assumptions required of the authentication mechanism in Byzantine agreement protocols that use "signed messages" are stronger than generally realized, and require more than simple digital signatures. The protocols may fail if these assumptions are violated. We then present new protocols for Byzantine agreement that add authentication to "oral message" protocols so that additional resilience is obtained with authentication, but with no assumptions required about the security of authentication when the number and kind of faults present are within the resilience of the unauthenticated protocol.

Our analysis is performed under a "hybrid" fault model that admits manifest (e.g., crash) and symmetric faults as well as arbitrary (i.e., Byzantine) faults. We also extend the classical signed messages protocol to this fault model, and show that its fault tolerance is matched by one of our new protocols. We then explore the behavior of these various protocols under the combination of hybrid processor faults and communications link faults. Using formal state-exploration techniques, we examine cases beyond those guaranteed by simple worst-case bounds and find that the resilience of one of the new protocols exceeds that of the others in these regions.

The new protocols are superior to other known protocols in properties and measures of practical interest, and we recommend them for general use. They are particularly attractive in security-critical systems where authentication may be subjected to sophisticated cryptographic attack, and in safety-critical embedded systems where it may be necessary to use very short signatures, but where maximum resilience is required.

1 Introduction

A fundamental requirement in fault-tolerant systems based on the "state machine" approach [27] is for replicated processors to reach agreement on the values of single-source data, such as sensor samples. In its abstract form, this is the problem of Byzantine Agreement (and its variant, the problem of "Interactive Consistency," also known as "source congruence," "distributed consensus," and "reliable multicast") [16, 23]. There are two broad classes of protocols for achieving Byzantine agreement. Those based on "oral message" assumptions place no restrictions on what a faulty processor may do; those based on "written message" assumptions disallow faulty processes making undetectable modifications to messages as they are relayed from one processor to another, and also disallow processors manufacturing messages that purport to come from another processor. It is generally stated that the written messages assumptions can be satisfied using cryptographic authentication methods (i.e., "digital signatures"), and protocols based on these assumptions are therefore often called "signed messages" or "authenticated" protocols [5, 11, 16].

Both oral and written message protocols proceed in "rounds" and the parameters of interest include: how many faults can be tolerated by a given number of processors, and how many rounds and how many messages are required? Theoretical studies also consider the size of the messages, or the total number of bits transmitted. The advantage of written messages protocols is that they can generally withstand more faults than oral message protocols, and often require fewer messages. For example, oral message protocols require $3t + 1$ processors to withstand t faults, while written messages protocols require only $t + 2$ (the problem is vacuous unless there are at least two nonfaulty pro-

*This work was supported in part by the National Aeronautics and Space Administration, Langley Research Center, under contract NAS1-20334, by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under contract F49620-95-C0044, and by the National Science Foundation under contract CCR-9509931.

†Li Gong is now with JavaSoft and can be reached at gong@eng.sun.com

cessors). However, both classes of protocols provably require $t+1$ rounds in the worst case [5, 11], though "early stopping" protocols (which are most easily constructed under the written messages assumptions) use fewer rounds when the actual number of faults is less than t [2, 7, 8, 10, 12].

It would seem that the written messages protocols have significant advantages over their oral message counterparts (e.g., asymptotically, a three-fold advantage in number of faults tolerated). However, these advantages may not be so significant in practice. In embedded applications, the most severe practical constraint on these protocols is the number of rounds: a given application will generally fix the number r of rounds it can afford (generally two). This, in turn, fixes the number of faults that can be tolerated at $r-1$, independently of the class of protocols chosen.[1] The class of protocols does affect the number of processors required: e.g., two-round written message protocols require three processors to tolerate a single fault, while oral message protocols require four. But if other purposes (e.g., clock synchronization) already require four or more processors, there seems no compelling reason to use written message protocols. In fact, there is an argument against these protocols which Chris Walter, one of the developers of the MAFT architecture for fault-tolerant flight control [15] expressed to us as follows: "you have to assume that digital signatures satisfy the requirements for written messages, and in life-critical systems we prefer to make as few assumptions as possible." It turns out that this caution is justified.

In the rest of the paper, we first describe the various assumptions that such protocols (we will call them "authenticated protocols") depend on, highlighting the risks in placing the correctness of Byzantine agreement on the effectiveness of cryptographic protocols for which currently there is no method of assurance that is definitive and generally accepted. We note, however, that authenticated protocols can tolerate more faults than oral message protocols, and we show that this advantage is retained when the analysis is extended to a hybrid fault model that counts faults more carefully than the purely Byzantine fault model.

We then consider the addition of authentication to variants of the Oral Messages protocol and show that this increases the number of faults they can tolerate if the assumptions on the authentication mechanism are warranted, without compromising their innate fault tolerance if those assumptions are violated. Assuming authentication, we show that one of these new protocols can tolerate as many hybrid faults as the classical Signed Messages protocol.

We then examine the two-round versions of the various protocols under an enlarged fault model that includes communications link faults. For many applications, this is the most realistic class of protocol and fault-model, and we provide evidence, derived from formal state-exploration techniques, that one of the authenticated oral message protocols provides the greatest fault tolerance.

2 Byzantine agreement, fault models, and message assumptions

In the classical Byzantine Generals problem, there are a number of participants, which we call "processors." A distinguished processor, which we call the *transmitter*, possesses a value to be communicated to all the other processors, which we call the *receivers*. (These correspond to the "Commanding General" and "Lieutenant Generals," respectively, in the terminology of Lamport, Shostak, and Pease [16].) It is assumed that there are point-to-point communications paths between each pair of processors. The Byzantine Agreement problem can be studied under several different sets of assumptions. We consider both "Oral" and "Written" message assumptions, and a "Hybrid" fault model. The *Oral Messages* assumptions are the following [16, p. 387].

A1: Every message that is sent between nonfaulty processors is correctly delivered.

A2: The receiver of a message knows who sent it (assumption of private channels).

A3: The absence of a message can be detected (assumption of synchrony).

Written Messages assumptions add the following to those of oral messages [16, p. 391].

A4(a): Messages sent by a nonfaulty processor (under the hybrid fault model—see later—this becomes a non-arbitrary-faulty processor) cannot be altered or manufactured by other processors.

A4(b): Any nonfaulty receiver can identify the processor that originated a message, if that processor is nonfaulty (again, under the hybrid fault model this becomes a non-arbitrary-faulty processor). Note that A2 concerns the case of a direct path from sender to receiver, whereas A4(b) concerns a message from an "originating sender"

[1] The small number of rounds and the deterministic processor and communications scheduling used in embedded applications also obviate the benefits of early stopping.

that is possibly relayed by other processors before reaching the receiver.

There are n processors in total, of which some (possibly including the transmitter) may be faulty. In the classical Byzantine Generals problem, there are no constraints other than those given above on the behavior of faulty processors. This leads to pessimistic estimates of the number of faults that can be tolerated because all faults are regarded as the worst possible. We therefore consider a "hybrid" fault model (originally due to Thambidurai and Park [29] and also investigated by Walter, Suri, and Hugue [30]) that distinguishes certain simpler kinds of fault as well as those that are unconstrained. The fault modes we distinguish for processors are *arbitrary-faulty*, *symmetric-faulty*, and *manifest-faulty*. A manifest fault is one that can be detected by mechanisms present in all nonfaulty processors (e.g., missing or improperly formatted messages). The other two fault modes yield behaviors that are not detectably bad: a symmetric fault presents the same faulty behavior to every nonfaulty processor; an arbitrary fault is completely unconstrained (i.e., Byzantine) and may present (possibly) different aberrant behaviors to some nonfaulty processors, and good behavior to others.

The above characterization of the hybrid fault model is a generic one; for Byzantine agreement, the characterization of fault modes has to be refined in terms of the processor behaviors relevant to this problem (see [26] for a different characterization in terms relevant to clock synchronization). The basic step in an agreement protocol is for a processor to transmit a value v to several other processors. The interpretation of a manifest fault in this context is one that produces detectably missing values (e.g., timing, omission, or crash faults), or that produces a value that all nonfaulty recipients can detect as bad (e.g., it fails checksum or format tests). Symmetric faults deliver *wrong*, rather than missing or manifestly corrupted values—but do so consistently, so that all receivers of a given transmission obtain the *same* wrong value $v' \neq v$. Arbitrary faults are unconstrained, and can deliver correct, wrong, or manifestly faulty values in any combination.

Under these assumptions, the Byzantine Agreement problem is to devise a protocol that will allow each receiver p to compute an estimate ν_p of the transmitter's value satisfying the following conditions:

Agreement: If receivers p and q are nonfaulty, then they agree on the value ascribed to the transmitter—that is, for all nonfaulty p and q, $\nu_p = \nu_q$.

Validity: If receiver p is nonfaulty, the value ascribed to the transmitter by p is

- The value actually sent, if the transmitter is nonfaulty or symmetric-faulty,
- The distinguished value E, if the transmitter is manifest-faulty.

All the Byzantine agreement protocols we consider proceed in rounds: in the first round, the transmitter sends a value to all the other processors; in subsequent rounds, these processors exchange the values received among themselves in order to detect inconsistencies; each receiver then decides on one value among those received and exchanged. How this decision is made, and how the exchanges are done, depends on the protocol considered.

Notice that the additional assumptions for written messages essentially constrain the behavior of symmetric- and arbitrary-faulty receivers: under oral message assumptions, such receivers can alter or manufacture messages purporting to come from other processors in the later rounds—this is prohibited under written messages assumptions. Authenticated protocols attempt to satisfy the written messages assumptions using digital signatures: each processor signs the messages that it sends. Any receiver can check the authenticity of a message and confirm the identity of its claimed originator by checking the signature. There are several digital signature schemes that provide these basic properties [4, 9, 22, 25]. However, in the following section we show that these schemes must be used very carefully.

3 Authentication issues

The messages that are passed among the processors in authenticated protocols have the form $\{\{\ldots\{v\}_p\ldots\}_q\}_r$ which symbolizes the value v in a message signed and sent by processor p, received signed and forwarded by processors $\ldots, q$ and finally received, signed and forwarded by processor r. If processor p is nonfaulty, then at no stage in the protocol should there exist $\{\{\ldots\{v'\}_p\ldots\}_q\}_r$ in which $v \neq v'$. (This follows because if p is nonfaulty, it would not send out two different values v and v', and authentication prevents any other processor manufacturing such a value.) It is generally assumed that this requirement is satisfied if digital signatures are simply computed on and attached to the messages being relayed. This would be true if a valid message of the form $\{\{\ldots\{v\}_p\ldots\}_q\}_r$ could only arise once in the lifetime of the protocol. Theoretical examinations of these protocols normally consider only a single "run,"

but in practice they will be called repeatedly (e.g., to distribute sensor samples at the beginning of every process control cycle). It follows that processor r could save a valid message $\{\ldots\{v'\}_p\ldots\}_q$ from one run of the protocol and could then inject the correctly signed message $\{\{\ldots\{v'\}_p\ldots\}_q\}_r$ into a later run, which will cause any nonfaulty receiver to conclude that the original sender p must be faulty, and thereby defeat the protocol.

We do not need to postulate active, intelligent attacks to be concerned about this kind of problem: a hardware "off by one" fault that causes a message to be picked up from the wrong buffer when two agreement protocols are in operation simultaneously (as when all processors are exchanging sensor data) could produce this behavior. A solution to this particular problem is to include additional information under the digital signatures that will identify messages as "fresh" (Lamport, Shostak, and Pease suggest sequence numbers [16, page 400]), but this needs to be done carefully in order to distinguish *this* run of the protocol from others that may be active simultaneously.

In the rest of this section, we discuss this and a number of other issues requiring care in the implementation of authenticated Byzantine agreement protocols.

Signature permutation. The signature system must not be commutative. Otherwise, $\forall p, q, v, \{\{v\}_p\}_q = \{\{v\}_q\}_p$ and, if the session initiator is faulty, another faulty processor can falsely accuse a third, but correct, processor of being faulty in a several-round protocol.

Verifying signature sequences. Verifying a sequence of t signatures is not trivial. A recipient can try all possible sequences of t out of n signatures, but this requires an exponential amount of computation. Or the message can include a hint, such as the identity of the signer, in each stage of the signing, so the message may look like $\{q, \{p, v\}_p\}_q$. We can alternatively require that a list of hints is attached to each message outside the signatures. However, such hints will add $O(n \log n)$ bits to the message length (in an n-round protocol), thus exceeding the tight lower bound on message bits by Srikanth and Toueg [28, Theorem 1] by a factor of n. (In today's practice, a secure digital signature uses about 512 to 1024 bits.) Note that hints are necessary whether the signature system used is commutative or not. A third approach is to globally order the messages so that a recipient can deduce from the context which signature sequence should be used for verification.

Processors are assumed to know each others' signature keys. Borcherding [3] investigates the case where there is no central authority to distribute these keys, and proposes the notion of "local authentication" to achieve a weaker version of Byzantine agreement.

Distinguishing concurrent sessions. When multiple sessions can execute at the same time, it is vital to determine to which run a message belongs. Otherwise, suppose each processor maintains a different sensor and all processors are trying to agree on the values of all sensors, then a faulty processor may "borrow" a signed message from one run and use it in another. Even a benign processor can possibly make such a mistake, as we described previously. One solution is to attach a session identifier, possibly the identity of the session initiator, to the sensor value. This solution will increase the size of each message by $O(\log n)$ bits. This does not exceed the lower bound by Srikanth and Toueg [28] because they already allocate $O(\log n)$ bits for signatures.

Detecting replay attacks. Beside distinguishing concurrent sessions initiated by different processors, it is equally important to detect any attempt to reuse past messages (from the same initiator) in a new run. The initiator must securely attach a freshness identifier to the signed value. For example, the initiator can sign both the freshness identifier and the value in the same signature.

There are three types of freshness identifiers, each of which can be used in more than one way [13]. The first is a timestamp, if processors have synchronized clocks. In this case, the initiator attaches the reading from the local clock to the value before signing them. A recipient rejects any message with a timestamp that is outside an agreed time window relative to the recipient's local clock. A significant risk exists when a faulty processor can also have a faulty clock so that the processor sends out values signed with timestamps in the future. Even if this processor were to recover, another faulty processor could play back such a message when the correct time comes. The significance of this attack lies in the fact that there is no guarantee that any correct processor will know the existence of previously signed messages (with future timestamps). To invalidate such messages, a repaired processor can change its signature key during reintegration.

The second type of a freshness identifier is a random number, also known as a "nonce." Since the nonce must be generated by the processor that is checking for freshness, processors must exchange nonces with each other (thus adding one round to the protocol),

and the value must be signed with all $O(n)$ nonces, thus increasing the message length significantly.

The third type is a counter value. Each processor maintains a monotonic counter, increments the counter value before initiating a session, and then signs the value together with the current counter value. Each processor also maintains a vector timestamp, noting the last seen counter value from every other processor, and rejects any value signed with a past counter value. Similar to timestamps, a faulty processor may sign "future" counter values, so it is prudent to change to a new signature key after repair.

Repair and restart. When a processor fails, it may lose all its state information, including the current session and round numbers and freshness identifiers. If the failure is arbitrary, then the surviving state information may be wrong. For example, its clock or counters may be turned back or forward. Moreover, simply asking every processor to reset their counters to zero is vulnerable to replay attacks. Therefore, to restore the synchrony between processors after repair, a repaired processor must use challenge-response (with nonces) to obtain from other processors fresh replies containing the current state information. Given the additional need of assigning a new signature key to the restarting processor and notifying all other processors of the corresponding public key, restart can be costly.

Message redundancy. A message containing the value to be signed must contain sufficient redundancy to protect against forgery. For example, a faulty processor p may choose a random number x and broadcast it as $\{v\}_p$ for some value v. Because it is quite possible that there is a value v' such that $x = \{\{v'\}_q\}_p$, p may effectively forge a signature of value v' signed by q. Or the faulty processor p can simply copy $\{v'\}_q$ from a previous protocol run and broadcast $\{\{v'\}_q\}_p$. Any processor r who further signs $\{\{v'\}_q\}_p$ is also spoofed.

There are many ways to introduce redundancy into the messages. One is to attach a checksum of a sufficient length to the original value. The size of the message will thus increase, perhaps by 128 bits (the size of a typical one-way hash function output) or at least $O(\log n)$ bits. Note that including a unique identifier of the current run does not provide sufficient redundancy because a randomly selected value x can be of the form $\{id, v\}_q$, and if id is for a future run, an attack can still happen in the future.

3.1 Practical implications

We have shown that authentication using digital signatures needs to be managed very carefully if it is to be secure against attack. How significant are these threats? There are two main classes of applications for authenticated Byzantine agreement protocols: secure systems that must maintain coordination in the face of capture and active subversion of system components (e.g., the AT&T "Rampart" architecture [24]), and safety-critical embedded control systems (e.g., the MAFT architecture for aircraft flight control [15]). Sophisticated cryptographic and other attacks are a given in the first class of applications, so our concern about the security of authentication needs no further justification here (the literature is replete with broken cryptographic protocols [1, 21]).

Intelligent malicious attack is not considered a serious possibility in embedded systems, and the argument in these cases is a little different. Byzantine-resilient architectures are attractive in these contexts because they simplify the case for assurance and certification: instead of a collection of fault-tolerance mechanisms to counter specific failure modes, and for which it is necessary to provide evidence of coverage and noninterference, we have a single mechanism that can withstand *any* kind of fault, up to some number, and it is only necessary to provide evidence for correctness and for the estimated overall fault arrival rate. Written message protocols compromise the purity of this position: faulty processors can no longer do absolutely *anything*, but are constrained by certain assumptions. Real processors *can* do absolutely anything when faulty, and in implementations using signed messages, it is the authentication mechanism that constrains them within the assumed fault mode. For certification, it is therefore necessary to provide strong evidence that the authentication mechanism does accomplish this: broken authentication is not just another fault to be tolerated, it is a violation of the assumptions under which correctness of the protocol—and hence of the entire architecture—is established.

We have seen that cryptographically strong authenticated protocols require even small data messages to be encapsulated in large signature and freshness-indicating wrappers, and to carry various key-management indicators. Hence, embedded systems may prefer to dispense with truly secure authenticated protocols and to use short keyed checksums (Lamport, Pease, and Shostak suggest a suitable checksum algorithm [16, page 400]), with fixed keys and simple sequence-numbers to indicate freshness. The authentication assumptions may sometimes fail to hold in this arrangement. In the following sections we present and study protocols that take advantage of authentication if it is present, but that retain Byzantine resilience even when signatures may be

forged. Since checksums will only rarely be "forged" by random malfunctions, these protocols are very well suited to the needs of embedded systems.

The discussion has so far focussed on authentication failure in one direction: failure to adequately constrain the behavior of a faulty processor. Authentication can also fail in the other direction: causing good messages to be rejected as bad. There are two ways this can come about: the authentication mechanism may be algorithmically incorrect or nonrobust (e.g., vulnerable to loss of crypto-synch), or a hardware fault might damage a key. The issues enumerated earlier in this section are intended to help designers avoid the first of these dangers; the second is more likely, but less serious, because it *is* just another fault, and will be tolerated to the same extent as other faults.

4 Signed messages with hybrid faults

We have argued that great care in implementation is necessary in order to satisfy the assumptions of the authenticated protocols. This care would be justified if the authenticated protocols had significant advantages over oral message protocols. However, for the case of practical importance—that is, two-round protocols—there appears little to choose between the two classes of protocols: the signed message protocol SM(1) and the oral messages protocol OM(1) of Lamport, Pease, and Shostak [16] both require two rounds[2], and both tolerate only a single arbitrary fault. The difference is that OM(1) requires four processors, while SM(1) requires but three. However, a variation on OM(1) called OMH(1) [19] that operates under the Hybrid fault model can tolerate a arbitrary, s symmetric, and m manifest faults simultaneously, provided n, the number of processors, satisfies $n > 2a+2s+m+1$ and $a \leq 1$. Thus, OMH(1) appears to tolerate *more* faults than SM(1) under certain circumstances. Of course, this comparison is unfair because the analysis for OMH(1) considers the hybrid fault model, whereas that for SM(1) treats all faults as arbitrary. So one item that warrants examination is the behavior of SM(1) under the hybrid fault model.

The classical signed messages protocol, SM(r) proceeds as follows [16, p. 391]:

SM(r)

> The transmitter sends a signed message to each receiver. Each receiver adds its signature to the message and sends it to the other receivers who add their signatures and send it to the others, and so on for r rounds. When all the exchanges are completed, each receiver discards any improperly signed messages, extracts the values sent by the transmitter from those that remain and applies a deterministic choice function to those values.

Note that if the transmitter is not arbitrarily-faulty, the set of values considered in the choice will be a singleton. Lamport, Pease and Shostak show [16, Theorem 2] that SM(r) can tolerate up to r faulty processors, the optimal result [6, 11].

To extend SM(r) and its analysis to the hybrid fault model is straightforward: the hybrid protocol SMH(r) simply recognizes and discards manifest-faulty values. Authentication prevents symmetric-faulty receivers from injecting correctly signed new values, so these receivers either duplicate other messages (which is harmless), or they introduce incorrectly signed messages, which will be discarded. Thus, messages from both manifest- and symmetric-faulty receivers either duplicate existing values or are ignored; hence they play no part in the protocol and it is as if these processors were absent. It follows that only arbitrary-faulty processors need be counted in the fault-tolerance calculation. Thus, by direct analogy with the corresponding result (Theorem 2, page 393) in [16], we have the following result.

Theorem 1 *For any r, Protocol SMH(r) satisfies Validity and Agreement provided $r \geq a$, where a is the number of arbitrary-faulty processors.*

The result is somewhat vacuous unless there are at least two nonfaulty processors, so we also have $n > a + s + m + 1$, and $r \geq a$. This may be compared with OMH(r), where we have $n > 2a+2s+m+r$ and $r \geq a$.

It can be seen that OMH(r) and SMH(r) have the same fault tolerance with regard to rounds, but that SMH(r) requires considerably fewer processors than OMH(r) (or, equivalently, can tolerate more faults for a given number of processors). However, this increased fault tolerance is obtained at the cost of depending on authentication: if the authentication assumptions fail for any reason, then SMH(r) may fail altogether.

5 Combining authentication and oral messages

The idea of examining SM(r) under the hybrid fault model suggests the dual inquiry: examining oral message protocols in the presence of authentication. It turns out that this yields protocols that combine the advantages of the two classes of protocols with few

[2] The parameter r to these protocols starts at zero, so that the number of rounds is $r + 1$.

of their disadvantages. As noted in the discussion of SMH(r), authentication turns symmetric-faulty receivers into manifest-faulty ones: they can only generate messages that are improperly signed. In order to exploit this in an oral messages protocol, we need a protocol that has the capability to discard bad messages. The classical protocol OM(r) does not do this, but our hybrid protocol OMH(r) does. It therefore seems the most promising place to start.

The protocol OMH(r) [19] is our modified and formally verified [17] version of Thambidurai and Park's protocol Z(r) [29], which is in turn a modification of the $r+1$-round oral messages protocol OM(r) of Lamport, Shostak, and Pease [16]. The key idea in both Z(r) and OMH(r) is to introduce a distinguished value E to record receipt of manifest-faulty messages. E values are ignored in the majority vote that each processor uses to decide its final value. In Z(r), E is used to record both manifest-faulty messages and the *report* of such messages relayed by another processor. This leads to confusion when there is a manifest-faulty transmitter and an arbitrary- or symmetrically-faulty receiver; Z(1) can fail in this circumstance, and this leads to more complex failures in the $r > 1$ cases. OMH(r) repairs this problem by treating the report of manifest-faulty values differently than those values themselves: $R(E)$ indicates the report of E, $R(R(E))$ the report of a report, and so on. An inverse function UnR is used to "strip off" these Rs at a later stage in the protocol. Only E (not $R(E)$, $R(R(E))$, etc.) is ignored in the majority vote.

As noted in the previous section, OMH(r) is able to tolerate a arbitrary, s symmetric, and m manifest faults simultaneously, provided n, the number of processors, satisfies $n > 2a + 2s + m + r$ and $r \geq a$. This is optimal when only arbitrary faults are present (we have $a = r$, $s = m = 0$, so that $n > 3a$, satisfying the lower bound established by Pease, Shostak, and Lamport [23]). Separate analysis shows that the protocol is also optimal when only manifest faults are present, and the obtained bound is $n > m$ [18]. When only symmetric faults are present, however, the protocol is definitely suboptimal, in that additional rounds can *reduce* its resilience. For example, in OMH(0) (where receivers simply accept whatever value they obtain from the transmitter), the number of symmetric-faulty receivers is irrelevant. In OMH(1), however, where receivers relay information to each other and take the majority of the values obtained, one symmetric-faulty receiver can defeat the protocol unless $n \geq 4$.

Suppose now that we use digital signatures to add authentication to OMH(r), thereby creating a protocol we can call OMHA(r). First, as Lamport, Shostak, and Pease observe [16, p.393], there is no point authenticating the final step in the protocol (i.e., the OMH(0) round), because we have point-to-point communications and the communication port on which a message arrives serves to authenticate it (this is Assumption A2); thus OMHA(0) is the same as OMH(0). For the general case, we simply modify OMH(r) so that processors sign all messages that they send, and improperly signed messages are treated by their receivers as E.

Notice that as long as authentication does not *introduce* faults (i.e., as long as a properly signed message cannot be mistakenly considered improperly signed), then OMHA(r) must have at least the fault tolerance of OMH(r), and this is independent of the cryptographic strength of the signature scheme. However, if we make the usual assumptions about the strength of the signature scheme, then authentication reduces the severity of faults that can be introduced by receivers. In particular, a symmetric-faulty receiver cannot inject a completely false value into the exchanges: at worst, it can inject an E or $R(E)$ value; similarly, an arbitrary-faulty receiver can selectively inject E and $R(E)$, or can pass on the true value that it received. (Faulty processors cannot inject $R(R(E))$ etc., because this would require an $R(E)$ correctly signed by another processor.) Unfortunately, the residual ability to inject $R(E)$ is sufficient to limit the number and combination of faults that can be tolerated by OMHA(r) to be no better, in the worst case, than for OMH(r).

This disappointing result suggests consideration of a protocol ZA(r), derived from Thambidurai and Park's protocol Z(r) in the same way that OMHA(r) is derived from OMH(r). Since Z(r) and ZA(r) lack the E, $R(E)$ distinctions of OMH(r) and OMHA(r), it follows that symmetric-faulty receivers are reduced to manifest-faulty in ZA(r). Similarly, arbitrary-faulty receivers are reduced to manifest-faulty or "nonfaulty with communications link faults," which is a case considered in Section 6. Furthermore, authentication overcomes the bug in Z(r); this bug arises in Z(1) when an arbitrary- or symmetric-faulty receiver injects spurious values into the exchanges under a manifest-faulty transmitter: the E values from the transmitter, and those relayed by good receivers, are ignored in the majority votes, which are therefore won by the spurious values injected by the faulty receiver. ZA(r) eliminates this bug because it prevents the faulty receivers manufacturing the spurious values that other proces-

sors will incorporate in their majority votes. Protocol ZA(r) is defined as follows.

ZA(0)

1. The transmitter sends its value to every receiver.
2. Each receiver uses the value received from the transmitter, or uses the value E if a missing or manifestly erroneous value is received.

ZA(r), $r > 0$

1. The transmitter signs and sends its value to every receiver.
2. For each p, let v_p be the value receiver p obtains from the transmitter, or E if no value, or a manifestly bad value, or incorrectly signed value is received.

   Each receiver p acts as the transmitter in Protocol ZA($r-1$) to communicate the value v_p to the other $n-2$ receivers.
3. For each p and q, let v_q be the value receiver p received from receiver q in step (2) (using Protocol ZA($r-1$)), or else E if no such value, or a manifestly bad value, or incorrectly signed value was received. Each receiver p calculates the majority value among all non-E values v_q received; if no such majority exists, the receiver uses some arbitrary, but functionally determined value.

We have the following results, where a, s, and m are the numbers of arbitrary-, symmetric-, and manifest-faulty processors, respectively, and n is the total number of processors.

Lemma 1 *If signatures are secure, then for any a, s, m and r, Protocol ZA(r) satisfies Validity.*

Proof: In the first round, the transmitter signs and sends its value to all receivers. Validity assumes a nonfaulty transmitter, so all nonfaulty receivers will obtain the correct value in this round. The receivers exchange values in subsequent rounds, and faulty receivers may inject faulty values into this process. However, authentication prevents the injection of any correctly signed value other than that sent by the original transmitter. Thus the only values entering the majority vote will be this value and, possibly, E. Since all good receivers obtained at least one copy of the value v directly from the transmitter, and some combination of vs and Es from other receivers, the hybrid majority will always be v. □

Theorem 2 *If signatures are secure, then for any r, Protocol ZA(r) satisfies conditions Validity and Agreement if $r \geq a$.*

Proof: The proof is by induction on r. In the base case $r = 0$ there can be no arbitrary-faulty processors, since $r \geq a$. If there are no arbitrary-faulty processors then the previous lemma ensures that ZA(0) satisfies Agreement, and Validity follows. We therefore assume that the theorem is true for ZA($r-1$) and prove it for ZA(r), $r > 0$.

First consider the case in which the transmitter is not arbitrary-faulty. Then Validity is ensured by Lemma 1, and Agreement follows from Validity. Now consider the case where the transmitter is arbitrary-faulty. There are at most a arbitrary-faulty processors, and the transmitter is one of them, so at most $a-1$ of the receivers are arbitrary-faulty. At the next stage, we have one less round to perform, and one less arbitrary fault to tolerate. Since we assume $r \geq a$, we also know $r-1 \geq a-1$, and we may therefore apply the induction hypothesis to conclude that ZA($r-1$) satisfies conditions Agreement and Validity. Hence, for each q, any two nonfaulty receivers get the same value for v_q in step (3). (This follows from Validity if one of the two receivers is processor q, and from Agreement otherwise). Hence, any two nonfaulty receivers get the same vector of values $v_1, \ldots, v_{n-1}$, and therefore obtain the same value $\mathit{hybrid\text{-}majority}(v_1, \ldots, v_{n-1})$ in step (3) (since this value is functionally determined), thereby ensuring Agreement. □

Theorem 2 shows that ZA(r) has the same (optimal) fault tolerance as SMH(r) when signatures are secure; however, ZA(r) has the significant advantage that it is not totally broken if authentication fails. In the presence of authentication failure, ZA(r) reverts to, at worst, the fault tolerance of Z(r). To be sure, Z(r) is vulnerable to certain configurations of two faults no matter how many rounds and receivers are used (that is why we developed OMH(r)), but in the important case $r = 1$, its failure mode is very precisely characterized (manifest-faulty receiver and at least one symmetric-fault or arbitrary-faulty receiver—the latter is required to break Agreement). An alternative is to use the protocol OMHA(r), whose fallback, OMH(r) is fully robust against arbitrary and manifest faults, but whose resilience in the presence of working authentication is inferior to that of ZA(r). Table 1 compares the various protocols we have discussed in terms of worst-case bounds.

Protocol	Authentication Assumptions	
	Violated	Sound
SM(r)	a = s = 0, n > m+1	n > a+s+m+1, r ≥ a
SMH(r)	a = s = 0, n > m+1	n > a+s+m+1, r ≥ a
OM(r)	n > 2a+2s+2m+r, r ≥ a	n > 2a+2s+2m+r, r ≥ a (same)
OMH(r)	n > 2a+2s+m+r, r ≥ a	n > 2a+2s+m+r, r ≥ a (same)
OMHA(r)	n > 2a+2s+m+r, r ≥ a	n > 2a+2s+m+r, r ≥ a (same)
Z(r)	n > 2a+2s+m+r, r ≥ a†	n > 2a+2s+m+r, r ≥ a† (same)
ZA(r)	n > 2a+2s+m+r, r ≥ a†	n > a+s+m+1, r ≥ a

† Z(1) also fails with a manifest-faulty transmitter and one symmetric- or arbitrary-faulty receiver; Z(r), $r > 1$, fails in additional cases.

Table 1: Comparison of Byzantine Agreement Protocols

6 Link faults

Communications failures represent an important class of faults; we call them *link* faults, with the characterization that when a nonfaulty processor sends its value v to a nonfaulty recipient over a faulty link, the value received may be either v or E.

Because they arise frequently in practice (wires and connectors are prone to noise and breakage), it is desirable to tolerate link faults efficiently. Notice that a link fault is not attributed to a processor; thus, a processor at the receiving end of a faulty link may be nonfaulty and the protocol must ensure that it satisfies the Agreement and Validity conditions. The difficulty in extending Byzantine agreement protocols to link faults is due to the fact that these faults do introduce asymmetry and are therefore as expensive to tolerate as arbitrary failures in the worst case.

We can observe that ZA(r) achieves Validity in the presence of link faults and hybrid processor faults, provided that there is path of length $r + 1$ links or less from the transmitter to each nonfaulty receiver that passes through only nonfaulty processors and good links. SMH(r) has the same bounds on Validity as ZA(r), while that of OMHA(r) is worse and difficult to characterize. We can also observe that for Agreement, a link fault is as disruptive, in the worst case, as an arbitrary fault at either the sender or receiver on the link. Thus, if link faults are attributed to either their sender or receiver, and l is the minimum number of processors needed to account for all such faults, then ZA(r) will achieve Agreement provided $r \geq a+l$. Similar worst case bounds apply for Agreement in SMH(r), while OMHA(r) requires $n > 2a+2s+m+r+2l$ and $r \geq a+l$.

7 Examining fault tolerance using state-exploration techniques

The worst-case bounds given above are based on rather crude ways of counting faults: there are many scenarios for the behavior of a system with, say, one arbitrary-faulty and one manifest-faulty processor and two link faults, but the worst-case analyses treat them all alike. It is therefore interesting to enquire how well the protocols perform under more fine-grained analysis and, in particular, how they perform in regions beyond those characterized by the simple worst-case bounds.

Simulation could be used to sample the behavior of the protocols, but a more attractive alternative is to use a formal state-exploration tool to examine their behavior in specific configurations under *all* scenarios. The idea is to model the system as the composition of two concurrent processes: one that injects faults and one that tolerates or diagnoses them. A state-exploration tool will then systematically explore all possible scenarios for their interaction.

We have used the Murϕ (pronounced "Murphy") system from David Dill's group at Stanford [20] for this purpose. Essentially, we provided Murϕ programs for the OMH(1), OMHA(1), Z(1), ZA(1), and SMH(1) protocols in the $n = 5$ case, and caused Murϕ to nondeterministically perform a symbolic "fault injection" (of both link faults and hybrid processor faults) and then run the protocols. By exploring all different runs (there are over 20,000 of them), Murϕ essentially undertakes exhaustive fault injection on these protocols (the process takes a couple of minutes on a SPARC 10). Of course, it would be straightforward to write a program to do this, but we consider the use of formal state-exploration tools a very promising and general

technique for the examination of algorithms for fault tolerance and diagnosis.

Our experiments confirmed the worst-case bounds on fault tolerance claimed for the various protocols in the case $n = 5$ and $r = 1$, and rediscovered the known vulnerability of Z(1) to manifest-faulty transmitters [19]. That is to say, exhaustive search of all fault configurations satisfying the bounds claimed in Table 1 for the case of $n = 5$ and $r = 1$ found no violations of Validity nor of Agreement, except for the known cases in Z(1).

However, much more interesting results were obtained when we allowed fault-injection to continue beyond the simple characterizations of worst-case fault tolerance for the protocols concerned. For example, although no five-processor, two-round protocol can withstand two link faults in the worst case, we found ZA(1) does tolerate two such faults in most cases. We therefore used our Murϕ fault-injection system to count how many scenarios caused each protocol to fail with and without the assumption of secure authentication.

Protocol	Authentication Assumptions	
	Violated	Sound
OMH(1)	25	25
OMHA(1)	25	23
Z(1)	24	24
ZA(1)	24	12
SMH(1)	43	13

Table 2: Percentage of fault configurations in a 5-plex where each protocol fails

Table 2 compares the various protocols we have discussed, using exhaustive state exploration to calculate the percentage of fault configurations that caused the protocols to fail. Overall, it seems that ZA(1) is the most resilient of these protocols under the combination of hybrid and link faults, though more experiments are needed to confirm this.

Fault configurations consist of an assignment of fault class (good, manifest, symmetric, or arbitrary) to each processor, and an assignment of up to three faulty links between processors. We excluded configurations with link faults emanating from arbitrary or manifestly faulty transmitters, or arriving at faulty receivers (such link faults have no real impact on system behavior). For each configuration, we tested whether any scenario of messages by the faulty processors could cause good receivers to disagree or cause a good receiver to fail to agree with the transmitter. For each protocol, we then calculated the percentage of all fault configurations for which such failure was possible.

The newest release of the Murϕ system automatically detects and exploits symmetry in appropriately written specifications, reducing the search space dramatically. For example, the configuration where all processors are good except that the third receiver is manifest-faulty is isomorphic to the case when all processors are good except the second receiver, and Murϕ only explores one of these alternatives. Symmetries are used in the assignment of faulty links as well as in the assignment of behaviors to processors. Because of these symmetry reductions, not all configurations are counted individually, so the numbers in Table 2 should be taken to indicate relative, not absolute, performance. We further reduced the set of configurations to require at least one good receiver, since otherwise validity and agreement are trivially satisfied. We excluded symmetric-faulty processors sending manifestly bad (E) values, since this would amount to the same thing as a manifest fault, and we also excluded the case of a symmetric-faulty transmitter since there is very little difference between this case and that when the transmitter is good. However, we did allow an arbitrary-faulty transmitter to behave in any way, including the possibility of behaving as good, symmetric- or manifest-faulty, as well as sending various combinations of good, wrong, and E values.

For the authenticated protocols, faulty receivers were not allowed to send data values other than that received from the transmitter. Thus for algorithm ZA(1) arbitrary-faulty receivers are only able to send manifestly bad (E) values or the correct value. In algorithm OMHA(1), arbitrary-faulty receivers also have the opportunity to send $R(E)$ and, as discussed earlier, this is the main source of brittleness of OMHA(1). We further make the assumption in these experiments that authentication never leads to good processors discarding good messages. These factors, taken together, significantly reduce the total number of configurations that need to be considered, but do not effect the relative numbers of configurations where the various algorithms behave acceptably.

The table shows that the authenticated protocol ZA(1) wrings the maximum fault tolerance from a given amount of redundant hardware, and outperforms the classical Signed Messages protocol whether or not signatures are secure (dramatically so if signatures are insecure). ZA(1) is also superior in overall resilience to OMHA(1). This is not to say that ZA(1) is uniformly superior to OMHA(1). Consider a good

transmitter with link faults to all receivers except p, and p has a link fault to receiver q. Under ZA(1), q decides on E and all the other receivers decide on the value sent by the transmitter to p, thereby violating Agreement. Under OMHA(1) all receivers settle on E.

Note that we are testing the fault tolerance of these protocols well beyond their usually claimed fault tolerance: only approximately five percent of all fault configurations we studied fall within the worst-case bounds of the protocols. Thus, all these protocols are far more tolerant of faults than their simple worst-case bounds would suggest.

8 Conclusion

The assumptions required of the authentication mechanism in Byzantine agreement protocols that use "signed messages" are stronger than generally realized, and require that digital signatures are used with great care. Violation of these assumptions can cause the protocols to fail. We have presented new protocols that combine authentication with "oral messages" protocols so that additional resilience is obtained when the authentication assumptions are sound, but the resilience of the unauthenticated protocol is retained when authentication assumptions are violated.

When the authentication assumptions are sound, one of these new protocols, called ZA(r), matches the fault tolerance of the classical signed messages protocol under a hybrid fault model, and surpasses it when communications link faults are considered. ZA(r) also performs well overall when authentication assumptions are violated, but has an unfortunate "hole" in its worst-case bound (it is vulnerable when the transmitter is manifest-faulty). Another of the new protocols, OMHA(r) may be preferred if this case is considered important, though it is less resilient to link faults than ZA(r).

These new protocols are superior to other known protocols in properties and measures of practical interest, and we recommend them for general use. They are particularly attractive in security-critical systems where authentication may be subjected to sophisticated cryptographic attack, and in safety-critical embedded systems where maximum resilience is required but where only short or cryptographically weak signatures (e.g., checksums) may be feasible. Selection of the most suitable protocol for a given system must obviously depend on the expected modes and frequencies of faults, and the consequences of system failure.

Our use of the state-exploration system Murϕ to perform symbolic "fault injection" is, we believe, novel. It suggests a very promising new application area for this class of formal methods tools, and one that we intend to pursue in future work.

Acknowledgments

Our understanding of these topics has benefitted greatly from discussions with Chris Walter and Michele Hugue (both then with Allied Signal). Comments by the anonymous reviewers were also very helpful. Malte Borcherding of the University of Karlsruhe pointed out some errors in the original paper.

References

Papers by SRI authors can generally be retrieved from `http://www.csl.sri.com/fm.html`.

[1] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. In *Proceedings of the Symposium on Research in Security and Privacy*, pages 122–136, Oakland, CA, May 1994. IEEE Computer Society.

[2] Birgit Baum-Waidner. Byzantine agreement with a minimum number of messages both in the faultless and worst case. In *Fault Tolerant Computing Symposium 23* [14], pages 554–563.

[3] Malte Borcherding. Efficient failure discovery with limited authentication. In *15th International Conference on Distributed Computing Systems*, pages 78–82, Vancouver, Canada, May 1995. IEEE Computer Society.

[4] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–650, November 1976.

[5] D. Dolev and H. R. Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, November 1983.

[6] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for Byzantine agreement. *Journal of the ACM*, 32(1):191–204, January 1985.

[7] Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. Early stopping in Byzantine agreement. *Journal of the ACM*, 37(4):720–741, October 1990.

[8] Klaus Echtle. Fault masking with reduced redundant communication. In *Fault Tolerant Computing Symposium 16*, pages 178–183, Vienna, Austria, July 1986. IEEE Computer Society.

[9] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, July 1985.

[10] Paul D. Ezhilchelvan. Early stopping algorithms for distributed agreement under fail-stop, omission, and timing fault types. In *6th Symposium on Reliability in Distributed Software and Database Systems*, pages 201–212, Williamsburg, VA, March 1987. IEEE Computer Society.

[11] M. Fischer and N. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14:183–186, 1982.

[12] F. Di Giandomenico, M. L. Guidotti, F. Grandoni, and L. Simoncini. A graceful dependable algorithm for Byzantine agreement. In *6th Symposium on Reliability in Distributed Software and Database Systems*, pages 188–200, Williamsburg, VA, March 1987. IEEE Computer Society.

[13] L. Gong. Variations on the themes of message freshness and replay. In *Proceedings of the Computer Security Foundations Workshop VII*, pages 131–136, Franconia, NH, June 1993. IEEE Computer Society.

[14] *Fault Tolerant Computing Symposium 23*, Toulouse, France, June 1993. IEEE Computer Society.

[15] R. M. Kieckhafer, C. J. Walter, A. M. Finn, and P. M. Thambidurai. The MAFT architecture for distributed fault tolerance. *IEEE Transactions on Computers*, 37(4):398–405, April 1988.

[16] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[17] Patrick Lincoln and John Rushby. Formal verification of an algorithm for interactive consistency under a hybrid fault model. In Costas Courcoubetis, editor, *Computer-Aided Verification, CAV '93*, volume 697 of *Lecture Notes in Computer Science*, pages 292–304, Elounda, Greece, June/July 1993. Springer-Verlag.

[18] Patrick Lincoln and John Rushby. Formal verification of an algorithm for interactive consistency under a hybrid fault model. Technical Report SRI-CSL-93-2, Computer Science Laboratory, SRI International, Menlo Park, CA, March 1993. Also available as NASA Contractor Report 4527, July 1993.

[19] Patrick Lincoln and John Rushby. A formally verified algorithm for interactive consistency under a hybrid fault model. In *Fault Tolerant Computing Symposium 23* [14], pages 402–411.

[20] Ralph Melton and David L. Dill. *Murφ Annotated Reference Manual*. Computer Science Department, Stanford University, Stanford, CA, March 1993.

[21] Judy H. Moore. Protocol failures in cryptosystems. *Proceedings of the IEEE*, 76(5):594–602, May 1988.

[22] National Institute of Standards and Technology. The digital signature standard. *Communications of the ACM*, 37(7):36–40, July 1992.

[23] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[24] Michael Reiter. A secure group membership protocol. In *Proceedings of the Symposium on Research in Security and Privacy*, pages 176–189, Oakland, CA, May 1994. IEEE Computer Society.

[25] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[26] John Rushby. A formally verified algorithm for clock synchronization under a hybrid fault model. In *Thirteenth ACM Symposium on Principles of Distributed Computing*, pages 304–313, Los Angeles, CA, August 1994. Association for Computing Machinery.

[27] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

[28] T.K. Srikanth and S. Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, 1987.

[29] Philip Thambidurai and You-Keun Park. Interactive consistency with multiple failure modes. In *7th Symposium on Reliable Distributed Systems*, pages 93–100, Columbus, OH, October 1988. IEEE Computer Society.

[30] C. J. Walter, N. Suri, and M. M. Hugue. Continual online diagnosis of hybrid faults. In F. Cristian, G. Le Lann, and T. Lunt, editors, *Dependable Computing for Critical Applications—4*, volume 9 of *Dependable Computing and Fault-Tolerant Systems*, pages 233–249. Springer-Verlag, Vienna, Austria, January 1994.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.