Matrix Signatures: From MACs to Digital Signatures

Amitanand S. Aiyer †, Lorenzo Alvisi †, Rida A. Bazzi ‡, Allen Clement †
† Department of Computer Sciences
The University of Texas at Austin
‡ School of Computing and Informatics
Arizona State University
{anand, lorenzo, aclement}@cs.utexas.edu, bazzi@asu.edu

Abstract

We present the first general implementation to provide the properties of digital signature using MACs in a system consisting of any number of untrusted clients and n servers, up to f of which are Byzantine. At the heart of the implementation is a novel *matrix signature* that captures the collective knowledge of the servers about the authenticity of a message. Matrix signatures can be generated or verified by the servers in response to client requests and they can be transmitted and exchanged between clients independently of the servers. The implementation requires that no more than one third of the servers be faulty, which we show to be optimal. The implementation places no synchrony requirements on the communication and only require fair channels between clients and servers.

1 Introduction

Developing dependable distributed computing protocols is a complex task. Primitives that provide strong guarantees can help in dealing with this complexity, and often result in protocols that are simpler to design, reason about, and prove correct. Digital signatures are a case in point: by guaranteeing, for example, that the recipient of a signed message will be able to prove to a disinterested third party that the signer did indeed sign the message (*non repudiation*), they can discourage fraudulent behavior and hold malicious signers to their responsibilities. Weaker primitives such as message authentication codes (MACs) do not provide this desirable property.

MACs, however, offer other attractive theoretical and practical features that digital signatures lack. First, in a system in which no principal is trusted, it is possible to implement MACs that provide unconditional security—digital signatures instead are only secure under the assumption that one-way functions exist [13], which, in practical implementations, translates in turn to a series of unproven assumptions about the difficulty of factoring, the difficulty of computing discrete logarithms, or both. Second, certain MAC implementations (though not the ones that guarantee unconditional security!) can be three orders of magnitude faster to generate and verify than digital signatures [3].

Given these rather dramatic tradeoffs, it is natural to wonder whether, by introducing stronger assumptions about some of the principals, it is possible to get the best of both worlds: a MAC-based implementation of digital signatures.

The few successful attempts to date at addressing this challenge [3, 4, 5, 8, 15] have produced solutions specific only to the particular protocols for which the implementation was being sought—these MAC-based, signature-free protocols do not offer, nor seek to offer, a generic mechanism for transforming an arbitrary protocol based on digital signatures into one that uses MACs. Further, these new protocols are significantly less intuitive than their signature-based counterparts, so much so that their presentation tends to be confined to obscure technical reports [3, 9].

In this paper, we present the first general implementation to provide the properties of digital signature using MACs in a system consisting of any number of untrusted clients and n servers, up to f of which are Byzantine. At the heart of the implementation is a novel *matrix signature* that captures the collective knowledge of the servers about the authenticity of a message. Matrix signatures can be generated or verified by the servers in response to client requests and they can be transmitted and exchanged between clients independently of the servers. The implementation requires that no more than one third of the servers be faulty, which we show to be optimal. The implementation places no synchrony requirements on the communication and only require fair channels between clients and servers.

In summary, we make three main contributions:

- We introduce matrix signatures, a general, protocol-agnostic MAC-based signature scheme that provides properties, such as non-repudiation, that so far have been reserved to digital signatures.
- We present an implementation of a signing and verification service for matrix signatures. We prove its correctness under fairly weak system assumptions (asynchronous communication and fair channels) as long as at most one third of the servers are arbitrarily faulty.
- We show that no MAC based signature and verification service can be implemented using fewer servers, even under stronger assumptions (synchronous communication and reliable channels).

2 Related Work

Matrix signatures differ fundamentally from earlier attempts at using MACs in lieu of signatures by offering a general, protocol-agnostic translation mechanism.

Recent work on practical BFT state machine replication [4, 3, 5, 8] has highlighted the performance opportunities offered by substituting MACs for digital signatures. These papers follow a similar pattern: they first present a relatively simple protocol based on digital signatures and then remove them in favor

of MACs to achieve the desired performance. These translations, however, are protocol specific, produce protocols that are significantly different from the original—with proofs of correctness that require understanding on their own— and are often incomplete. Of these efforts, only [3] succeeds in completely replacing signatures with MACs by replacing the signature generated by a set of replicas with an (implicit) broadcast-like primitive work of Srikanth and Toueg [15] (discussed below).

Aiyer et al. [1] present a wait-free atomic register implementation that tolerates Byzantine readers and up to one third Byzantine servers without relying on cryptographic primitives. Their protocol entirely avoids digital signatures, but the secret sharing mechanism they use depends on assuming a non-malicious writer.

Srikanth and Toueg [15] consider the problem of implementing *authenticated broadcast* in a system where processes are subject to Byzantine failures. Their implementation, which provides unconditional guarantees, is applicable only to a closed system of $n > 3f$ processes, with authenticated pairwise communication between them. They do not consider the general problem of implementing signatures: given a message one cannot tell if it was "signed" unless one goes through the history of all messages ever received and determines that at some point the message was broadcast. This renders their approach unusable if the signed messages are persistent.

In contrast, we provide signing and verification primitives for an open asynchronous system with any number of clients. While our approach can be used with unconditionally secure MACs, using practical implementations of MACs (without the unconditional guarantees) requires only bounded memory at each server.

Mechanisms based on unproven number theoretic assumptions, are known to implement digital signatures using local computation without requiring any trusted principal [6, 12]. It is also known that implementing signatures using authenticated channels (or MACs) is possible if clients communicate with a known trusted principal in the system [14]. In the absence of a known trusted principal, implementing digital signatures locally requires one-way functions [13]. Our results show that with partial trust in the system, implementing digital signatures is still possible without requiring one-way functions.

3 MACs and Digital Signature Schemes

Digital signatures provide the following properties, analogous to those of their physical counterparts [14]:

- **Authenticity** The signature convinces the recipient that the signer has signed the message.
- **Unforgeability** Only an authorized signer can generate a signature that successfully passes the verification.
- **Non-Reusability** The signature of a message (that has not been signed) cannot be derived from the signature(s) of a different message(s).
- **Non-repudiation** Upon successful verification the verifier can convince any other participant that the signer has signed the message.
- **Integrity** The signature generated for one message can not be used as a signature for a different message.

Collectively these properties allow recipients to authenticate both the origin and the content of a message in a manner that is provable to a disinterested third party [2]—a desirable ability in distributed systems where processes, the network, or both can fail in unpredictable ways. These properties, however, are informal descriptions of physical signatures. In the rest of this section we formalize these properties for digital signatures.

3.1 Digital Signatures

Digital signatures are implemented by signature schemes. In general, a signature scheme defined over a set of signers S and a set of verifiers V (not necessarily disjoint) consists of a signing procedure $\mathcal{S}$ and a verification procedure $\mathcal{V}$:

$$\mathcal{S}_{S,V} \quad : \quad \Sigma^* \mapsto \Sigma^* \tag{1}$$

$$\mathcal{V}_{S,V} \quad : \quad \Sigma^* \times \Sigma^* \mapsto \{\textbf{true}, \textbf{false}\} \tag{2}$$

The signing procedure $\mathcal{S}$ takes as arguments the message to be signed and returns a signature. The verification procedure takes as arguments a message and a signature and returns a boolean value. When a verification procedure returns true, we say that the signature passes the verification procedure for the message; otherwise it fails.

The set of signing processes S contains all the processes that can invoke the signing procedure. To simplify the exposition, we assume that this set contains, a priori, any processes with which signers might want to collude and we assume that once S is specified only processes in S can invoke the signing procedure. V contains all the processes that can verify a signature for the signature scheme.

Formal definitions that aim to capture mathematical constructions of signature schemes explicitly include signing and verification keys in the definition. In this work, we are only interested in using signatures schemes as black boxes, so we omit the keys for simplicity and assume they are implicitly captured in S and V. In practice, S would be the set of processes that has the necessary information (keys) to sign and V would be the set of processes that has the necessary information (keys) to verify the signatures. For example, with public key cryptography, the private key which is necessary to sign is known only to one participant – namely the signer, and the public key which is necessary to verify the signature is known to all. S would be a singleton and V consists of all the participants in the system.

We formalize the following requirements of a digital signature scheme:

- (**Consistency**) If a non-faulty sender $s \in S$ invokes $\mathcal{S}_{S,V}(s, msg)$ to generate msg_{sig} and $v \in V$ is a non-faulty verifier, then $\mathcal{V}_{S,V}(msg, msg_{sig})$ invoked by v always returns **true**.

- (**Validity**) A signature msg_{sig} can pass the verification function for a message msg at a non-faulty verifier $v \in V$

  $$\mathcal{V}_{S,V}(msg, msg_{sig}) = \textbf{true}$$

  only if a sender $s \in S$ has invoked the signing function $\mathcal{S}_{S,V}(s, msg)$ on the same message msg.

- (**Verifiability**) If a non-faulty verifier $v \in V$ returns **true** on invoking the verification procedure $\mathcal{V}_{S,V}(msg, msg_{sig})$ then it can generate a proof msg_{proof} to convince any (non-faulty) disinterested third party that a sender $s \in S$ has executed $\mathcal{S}_{S,V}(s, msg)$.

Digital Signatures typically achieve verifiability by allowing all processes to be able to invoke the verification procedure; and by ensuring that if $\mathcal{V}_{S,V}(msg, msg_{sig})$ returns **true**, the verifier can produce a proof msg_{proof} such that $\mathcal{V}_{S,V}(msg, msg_{proof})$ will always succeed. Signature schemes with a deterministic verifying procedure, such as [6, 12], have $msg_{proof} = msg_{sig}$.

Any digital signature scheme that meets these requirements provides the general properties expected of signatures. Consistency provides authentication; validity provides unforgeability, non-reusability and integrity; and verifiability provides non-repudiation since the set of verifiers is typically considered to be all processes.

3.2 Message Authentication Codes

MACs are used to implement reliable communication between a particular sender and a particular receiver. They can be formalized in terms of the signing and verifying procedures as above, but are constrained to have $|S| = |V| = 1$.

MACs are defined with respect to a particular signer s and a particular verifier v. It consists of a signing procedure $\mathcal{S}$ and a verification procedure $\mathcal{V}$:

$$\mathcal{S}_{s,v} \quad : \quad \Sigma^* \mapsto \Sigma^* \tag{3}$$

$$\mathcal{V}_{s,v} \quad : \quad \Sigma^* \times \Sigma^* \mapsto \{\textbf{true}, \textbf{false}\} \tag{4}$$

The signing procedure takes a message and generates a MAC as an output. The verification procedure takes a message and a MAC and checks if the MAC is correct.

MACs are required to ensure authentication, unforgeability, non-reusability and integrity. They are do not provide non-repudiation. So, formally, they are only required to satisfy the consistency and validity requirements.

- (**Consistency**) If the non-faulty sender s invokes $\mathcal{S}_{s,v}(s, msg)$ to generate msg_{mac} and $\mathcal{V}_{s,v}(msg, msg_{mac})$ invoked by a non-faulty verifier v always returns **true**.
- (**Validity**) A MAC msg_{mac} can pass the verification procedure invoked by a non-faulty verifier v for a message msg

  $$\mathcal{V}_{s,v}(msg, msg_{mac}) = \textbf{true}$$

  only if the sender s has invoked the signing function $\mathcal{S}_{s,v}(s, msg)$ to generate the MAC msg_{mac}.

Typically, MACs are implemented based on symmetric key cryptography, where both the signer and verifier have access to all the keys used. The requirements restrict the behavior of a non-faulty verifier, so that it does not generate the MACs by itself (except as part of the verification procedure).

3.3 Discussion

Formalisms for MACs and digital signatures typically express their properties in terms of probabilities that the schemes can fail. For schemes that rely on unproven assumptions, restrictions are placed on the computational powers of the adversary. In this paper we are only interested in implementing signature using a finite number of black box MAC implementations. We state our requirement in terms of properties of the executions that always hold without reference to probabilities or adversary powers. This does not affect the results, but leads to a simpler exposition.[1].

4 Model

The system consists of two sets of processes: a set of n server processes (also known as replicas) and a finite set of client processes (signers and verifiers). The set of clients and servers is called the set of *participants*. The identifiers of participants are elements of a completely ordered set.

An execution of a participant consists of a sequence of events. An event can be an internal event, a message send event or a message receive event. Two particular internal events are of special interest to us. A *message signing event* invokes a signing function that is guaranteed to terminate. A *message verification event* is associated with the invocation of a verification procedure. In our implementations of signature schemes we only consider communication between clients and servers to implement the signing and the verification functions.

Clients communicate with the servers over authenticated point-to-point channels. Inter-server communication is not required. The network is asynchronous and fair—but, for simplicity, our algorithms are described in terms of reliable FIFO channels, which can be easily implemented over fair channels between correct nodes.

[1]Our implementations use only finitely many MACs, consequently the probability of breaking our implementation can be made arbitrarily small if the probability of braking the MAC implementations can be made arbitrarily small. Also, our requirements restrict the set of allowable executions, which in essence place a restriction on the computational power of the verifiers. In particular, they do not allow a verifier to break the signature scheme by enumerating all possible signatures and verifying them

To send a message, a process uses the *send* function that takes an intended recipient and a message *content* as parameters. We identify a sent message as a triplet (s, r, c), where s is the sender, r is the recipient and c is the content of the message. Every message sent by a non-faulty participant to another non-faulty participant is guaranteed to be delivered, but there is no bound on the time elapsed between the time a message is sent and the time it is delivered. A message delivered to p is of the form (s, c), where (s, p, c) is a message sent at an earlier time. To receive messages, a process p uses the receive function which returns the set of messages that have been delivered to p since the last time p called the receive function.

Each process has an internal state and follows a protocol that specifies its initial states, the state changes, and the messages to send in response to messages received from other processes. An arbitrary number of client processes and up to f of the server processes can exhibit arbitrary (Byzantine) faulty behavior [10]. Remaining processes follow the protocol specification.

5 Signatures using MACs

We now show how to achieve non-repudiation using a construct based on MACs. In a distributed setting with $n \geq 3f + 1$, our construction of *matrix-signatures* satisfies *consistency*, *validity* and *verifiability*, thus providing all the properties required of digital signatures.

We first present the high level idea assuming the existence of two trusted entities in the system. We later relax this assumption in Section 6, where we present a general construction that does not require any trusted entities.

5.1 Signatures with Trusted Witnesses

Suppose we have two trusted entities in the system. One trusted entity can act as a signing-witness, and another acts as a verifying-witness. These two witnesses share a secret-key $\mathcal{K}$ that is used to generate and verify MACs.

Signing a message Whenever a signer wants to sign a message, it delegates the signing witness to generate a signature. This signature, henceforth called a MAC-signature, certifies that the signer s wants to sign the message m. MAC-signature is essentially a MAC computed using the secret key $\mathcal{K}$ that is known only to the two witnesses.

On receiving a request from s to sign a message m, the signing witness generates the MAC-signature and gives it to s. The MAC-signature can be used by the verifiers to validate that s has signed the message.

Verifying a signature To verify that a MAC-signature is valid, the verifiers (clients) need to contact the verifying-witness. The verifying-witness uses the message and the signer information to compute the MAC-signature and then tells the verifiers (clients) whether the two MAC-signatures match or not. If so, then the verifier (client) concludes that the MAC-signature is correct; otherwise, it concludes that the MAC-signature is wrong.

5.2 Signature Properties

Consistency is ensured because the signing and verifying witnesses use the same key $\mathcal{K}$ to create the MAC-values. *Validity* is ensured because the MAC-value is only generated by the trusted signing witness when it receives a message from the signer over an authenticated channel. *Verifiability* is guaranteed because the verifying witness is trusted and will either always accept a MAC-signature for a given message, or will always reject it.

6 A Distributed Signature Implementation

In the absence of any known trusted witness, we can utilize $n > 3f$ servers to implement the same service in a fault tolerant manner.

We assume $n \geq 3f + 1$ witness servers in the system, of which no more than f are faulty. These witness servers share pairwise secret keys that can be used to generate pairwise MACs. Each witness

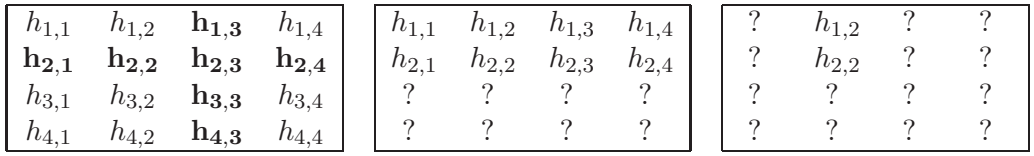

Figure 1: Example Matrix-signatures

replica functions both as a signing-witness-server to implement the signing witness, and as a verifying-witness-server to implement the verifying witness.

Clients can sign or verify a signature by contacting all the signing witness (or, respectively, verifying-witness) servers. The key difference with the protocol described in the previous section is that the signatures being generated or verified can no longer be implemented as a single MAC-value, but instead as a matrix of $n \times n$ MAC-values, called a *matrix signature*.

6.1 Matrix signatures

A matrix signature consists of n^2 MACs arranged into n rows and n columns which, together, capture the collective knowledge of the servers about a message authenticity.

Each MAC is based on a secret key ($\mathcal{K}_{i,j}$) shared between a signing-witness-server and a verifying-witness-server[2].

Every row of the matrix-signature corresponds to a signing-witness-server and every column corresponds to a verifying-witness-server. The i^{th} row of the matrix-signature comprises of the MACs generated *by* the i^{th} signing-witness-server. The j^{th} column of the matrix-signature comprises of the MACs generated *for* the j^{th} verifying-witness-server. In Figure 1, the row in bold font is generated by the 2^{nd} signing-witness-server, and the column in bold is generated for the 3^{rd} verifying-witness-server.

Definition 1 (Valid)**.** *A matrix-signature is valid if it has at least* $(f + 1)$ *correct MAC values in every column.*

Definition 2 (Admissible)**.** *A matrix-signature is said to be admissible if it has at least one column corresponding to a non-faulty server that contains at least* $(f + 1)$ *correct MAC values.*

An *admissible* matrix-signature captures the minimum requirement for it to be successfully verified by a non-faulty verifier. A *valid* matrix-signature captures the minimum requirement for being *guaranteed* to be always successfully verified by any non-faulty verifier. Thus, every *valid* signature is *admissible*, but the converse does not hold.

6.2 Protocol Description

The protocols for generating and verifying matrix-signatures is shown in Figure 2.

6.2.1 Generating a Signature

To generate a matrix-signature, the signer s sends the message Msg to be signed, along with its identity, to all the signing-witness-servers over *authenticated channels*. Each signing-witness-server generates a row of MACs, attesting that s signs Msg, and responds to the signer. The signer waits to collect the MAC-rows from at least $(2f + 1)$ signing-witness-servers to form the matrix-signature.

Note that the matrix-signature thus formed may contain some empty rows corresponding to the unresponsive/slow servers. It may also contain up to f rows with incorrect MAC values corresponding to faulty servers.

[2]Although a signing-witness-server and a verifying-witness-server can both be mapped to a single witness server, for the time being it is useful to think of them as separate entities.

6.2.2 Verifying a Signature

To verify a matrix-signature the verifier sends (a) the matrix-signature, (b) the message, and (c) the identity of the client claiming to be the signer to the verifying-witness-servers. A verifying-witness-server admits the signature only if at least $(f+1)$ MAC-values in the server's column are correct; otherwise, it rejects. Note that a non-faulty server will never reject a valid matrix-signature.

The verifier collects responses from the servers until it either receives $(2f+1)$ $\langle \text{ADMIT}, \ldots \rangle$ responses to accept the signature, or it receives $(f+1)$ $\langle \text{REJECT} \rangle$ responses to reject the signature as not *valid*.

Regenerating a valid signature : Receiving $(2f+1)$ $\langle \text{ADMIT}, \ldots \rangle$ responses does not guarantee that the signature is valid–only that it is admissible, hence leaving open the possibility that a signature accepted by a correct verifying client may not be later verifiable by a different correct verifying client. To see how this may happen, suppose a faulty client sends a matrix signature in which the column corresponding to a $(f+1)$ non-faulty verifying-witness servers is correct while all other columns contain incorrect MACs. The f faulty verifying witness server can then answer affirmatively to some verifying clients and reject the signature when other clients attempt the verification.

To satisfy *verifiability* and provide *non-repudiation*, we ensure that the verifier always gathers a valid matrix-signature as a proof whenever any signature (valid or just admissible) is accepted by the verification procedure.

This is accomplished by having a verifying-witness-server and signing-witness-server run on the same physical server. When the witness-server admits a matrix-signature (as a verifying-witness-server), as part of the $\langle \text{ADMIT}, \ldots \rangle$ message it generates and sends the (signing-witness-server's) corresponding row of MAC-values. Thus, if a verifier collects $(2f+1)$ $\langle \text{ADMIT}, \ldots \rangle$ responses, it can form a new regenerated matrix-signature with the received MAC-rows that is *valid*.

Ensuring termination : In an asynchronous system, a client cannot expect responses from more than $(n-f)$ non-faulty servers. If $n \geq 4f+1$, the verifier can always decide either to accept the signature, or reject it on receiving $(n-f) \geq (3f+1)$ responses. However, if $n \leq 4f$ then the verifier may not be able to make a decision on receiving just $(n-f)$ responses if some (f or fewer) of them are $\langle \text{REJECT} \rangle$ and some ($2f$ or fewer) of them are $\langle \text{ADMIT}, \ldots \rangle$.

The verifier can wait for additional responses only if it knows that it is yet to hear from a non-faulty server. If the original signature being verified is *valid* then it can wait for additional responses. However, it is also possible that all the non-faulty servers have already responded, where waiting for additional responses may not be a good choice.

Fortunately, these two scenarios can be distinguished in terms of the newly regenerated signature. If all non-faulty servers have already responded, and the verifier has less than f $\langle \text{REJECT} \rangle$ responses then the newly regenerated signature would be valid. Then, even the non-faulty servers that have rejected the initial signature should admit the regenerated signature, giving the verifier the required $(2f+1)$ admit responses. Otherwise, the verifier is yet to hear from at least one non-faulty server and waits for the additional responses.

Eventually, when all the non-faulty servers respond (to the original signature, and the newly generated signatures) the verifier will have enough $\langle \text{ADMIT}, \ldots \rangle$ responses or $\langle \text{REJECT} \rangle$ responses to accept or reject the signature.

6.3 Correctness

We now show that matrix-signatures satisfy all the requirements of digital signatures and ensure that the signing/verification procedures always terminate for $n \geq 3f+1$.

Lemma 1. *Every valid signature is admissible.*

Lemma 2. *The matrix-signatures generated by a non-faulty signer (Figure 2) is valid.*

Proof. A non-faulty signer has to collect MAC-rows from at least $(2f+1)$ servers to generate a matrix-signature. At least $(f+1)$ of these rows are from non-faulty servers and consist only of correct MAC

```
Signature Client-Sign (Msg Msg) {
    ∀i : σ_{Msg,S}[i][] :=⊥
    send  ⟨SIGN, Msg, S⟩ to all
    do
        (* collect MAC-rows from the servers *)
        rcv   ⟨σ_i[1...n]⟩ from server i
        σ_{Msg,S}[i][1...n] := σ_i[1...n]
    until rcvd from ≥ 2f + 1 servers
    return σ_{Msg,S}
}

bool Client-Verify(Msg Msg, Signer S, ref Signature σ) {
    (* σ_new collects the regenerated signature proof *)
    ∀i : σ_new[i][] :=⊥
    ∀i : resp[i] :=⊥
    send  ⟨VERIFY, Msg, S, σ[][]⟩ to all
    do
        rcv   ⟨ADMIT, σ_i[1...n]⟩ or ⟨REJECT⟩ from server i
        if received ⟨ADMIT, σ_i[1...n]⟩
            σ_new[i][1...n] := σ_i[1...n]
            resp[i] := ADMIT
            (* Accept: If enough servers admit it *)
            if ( Count(resp, ADMIT) ≥ 2f + 1 )
                (* update the signature (by reference) *)
                σ := σ_new
                return TRUE;
        else
            resp[i] := REJECT
            (* Reject: If at least one non-faulty server rejects it *)
            if ( Count(resp, REJECT) ≥ f + 1 )
                return FALSE;

        (*If No decision even after (n − f) responses *)
        if ( Count(resp, ADMIT) + Count(resp, REJECT) ≥ (n − f) )
            (* retry with the new signature *)
            send  ⟨VERIFY, Msg, S, σ_new[][]⟩ to { r : resp[r] ≠ ADMIT}
    until  (false)
}
```

```
void Signing-Witness-Server(Id i) {
    while(true) {
        rcv ⟨SIGN, Msg, S⟩ from S
        ∀j : compute σ_i[j] := MAC(K_{i,j}, S : Msg)
        send ⟨σ_i[1...n]⟩ to S
    }
}

void Verifying-Witness-Server(Id j) {
    while(true) {
        rcv ⟨VERIFY, Msg, S, σ⟩ from V
        correct_cnt := |{i : σ[i][j] ==
                              MAC(K_{i,j}, S : Msg)}|
        if (correct_cnt ≥ f + 1)
            ∀l : compute σ_j[l] := MAC(K_{j,l}, S : Msg)
            send ⟨ADMIT, σ_j[1...n]⟩ to V
        else
            send ⟨REJECT⟩ to V
    }
}
```

Figure 2: Signing and Verifying procedures for Matrix-signatures

values. □

Lemma 3. *A Valid signature always passes the verification procedure for a non-faulty verifier.*

Proof. A valid signature consists of all correct MAC-values in at least $(f + 1)$ rows. So, no non-faulty server will send a ⟨REJECT⟩ message. All non-faulty servers will accept the signature, and respond with the ⟨ADMIT, ...⟩ message. Since there are at least $(n - f) \geq 2f + 1$ non-faulty servers the verification procedure will pass on receiving these messages. □

Lemma 4. *If a matrix-signature is not-admissible, then it will fail the verification procedure for any non-faulty verifier.*

Proof. If a matrix-signature is not admissible, then all non-faulty servers will reject it by sending the ⟨REJECT⟩ message. On receiving $(n - f)$ responses, the verifier will have $(n - 2f) \geq (f + 1)$ ⟨REJECT⟩ messages causing the verification procedure to fail. □

Lemma 5. *If a matrix-signature passes the verification procedure for a non-faulty verifier, then it is admissible.* □

Lemma 6. *An adversary cannot generate an admissible signature for a message Msg, for which the signer did not initiate the signing procedure.*

Proof. For a signature to be admissible, a column corresponding to a non-faulty server (say j) should contain at least $(f + 1)$ correct MAC-values. Among the $(f + 1)$ correct MAC-values in the column, at least one correct MAC-value will correspond to a non-faulty server (say i).

$\mathcal{K}_{i,j}$ is only known to the non-faulty servers i and j. Only server i will generate the correct MAC for the clients.

In the verification procedure, Replica i would have generated the MAC-value for the clients only if it has received a signature that has $(f+1)$ correct MAC entries in its column. This could only happen if the signature previously verified was already an admissible signature generated.

In the signing procedure, server i will generate the MAC-value for the clients only if it receives a request from the signer to sign the document, as part of the signing procedure. □

Lemma 7. *If a signature passes the verification procedure for a non-faulty verifier, then the newly reconstructed matrix-signature is valid.*

Proof. For a signature to pass the verification procedure, the verifier must receive at least $(2f + 1)$ $\langle \text{ADMIT}, \ldots \rangle$ responses. At least $(f + 1)$ of these are from non-faulty servers and include a correct MAC-row along with the response. Thus the reconstructed matrix-signature consists of at least $(f + 1)$ correct rows. □

Lemma 8. *If a non-faulty client accepts that S has signed Msg after the matrix-signature passes the verification procedure, then it can convince every other non-faulty client that S has signed Msg.*

Proof. If a non-faulty client accepts that S has signed Msg after the matrix-signature passes the verification procedure, then the client would be able to gather a regenerated *valid* matrix-signature to that effect. Since a *valid* matrix-signature always passes the verification procedure, the client can present this valid matrix-signature to convince other clients that S has signed Msg. □

Theorem 1. *The Matrix-signature scheme presented in Figure 2 satisfies consistency, validity and verifiability.*

Proof. Consistency follows from lemmas 2 and 3. Validity follows from lemmas 5 and 6. Verifiability follows from lemmas 7 and 3. □

Theorem 2. *If $n \geq 3f + 1$ the signing procedure always terminates for any non-faulty signer.*

Proof. There are at least $(n - f)$ non-faulty servers that will respond to the signer. Thus eventually, it will get $(n - f) \geq (2f + 1)$ responses. □

Theorem 3. *If $n \geq 3f + 1$ the verification procedure always terminates for any non-faulty verifier.*

Proof. All non-faulty servers will eventually respond to the verifier. If the verifier already receives either $(f+1)$ $\langle \text{REJECT} \rangle$ responses, or receives $(2f+1)$ $\langle \text{ADMIT}, \ldots \rangle$ responses, the verification procedure will clearly terminate.

Suppose that the verifier does not terminate. Then it cannot have received more than f $\langle \text{REJECT} \rangle$ responses. Thus, it would have received at least $(f+1)$ $\langle \text{ADMIT}, \ldots \rangle$ responses from the non-faulty servers that is accompanied with the correct row of MACs. These $(f + 1)$ rows of correct MACs will ensure that the new signature σ_{new} is Valid.

Thus all non-faulty servers that have not sent a $\langle \text{ADMIT}, \ldots \rangle$ response will do so when they receive σ_{new}. The verifier will eventually have $(n - f) \geq (2f + 1)$ $\langle \text{ADMIT}, \ldots \rangle$ responses thus enabling the verification procedure to terminate. □

7 Discussion

Our distributed implementation of digital signatures is based on an underlying implementation of MACs. We make no additional computational assumptions to implement the digital signatures. However, if the underlying MAC implementation relies on some computational assumptions (e.g. collision resistant hash functions) then the signature scheme realized will be secure only as long as those assumptions hold.

7.1 Plausible Alternative Approaches

Section 5 demonstrates a known concept that it is feasible to implement a signature scheme using a known trusted entity. One might wonder if we can use a distributed implementation of either a replicated state machine, or a quorum system to build such a known trusted entity. We argue that while these approaches seem plausible, they are not feasible in a distributed setting where the network is asynchronous and channels are fair (i.e may drop messages).

7.1.1 State Machine Approaches

Implementing a replicated state machine requires making synchrony assumptions about the network. In an asynchronous setting a replicated state machine implementations cannot provide both safety and liveness [7]. Replicated state machine implementations require periods of synchrony to make any progress[4]. Thus such a system cannot guarantee that the signing procedure or the verifying procedure will terminate.

Even if the network were synchronous and implementing a replicated state machine was possible, it is not useful to implement the notion of a trusted witness as required in section 5. Replicated state machines are only useful to implement a service that do not have any confidentiality requirements. If the secret keys required by the trusted witness are replicated at all the replicas, then compromising even a single replica gives the adversary the power to generate signatures for any message.

7.1.2 Quorum based Approaches

Quorum Systems are also useful to build dependable distributed services in both synchronous and asynchronous setting. They rely on the intersection property of quorum sets to tolerate faults. Without signatures, masking f Byzantine faults require the intersection to be at least $(2f+1)$. Hence, such systems cannot be realized in an asynchronous setting unless the total number of servers in the system $n \geq 4f+1$. So, these approaches will not be helpful for protocols that only assume $n \geq 3f+1$ servers.

Our solution, presented in section 6, is applicable to all distributed protocols that satisfy $n \geq 3f+1$ even if the network is asynchronous and the channels are fair. We now show that any scheme that provides a general implementation of signatures using MACs will require $n \geq 3f+1$.

8 Lower Bound

We show that a generalized scheme to implement the properties of signatures using MACs cannot be done with $n \not> 3f$ replicas. The lower bound holds even in a much stronger model where the network is synchronous and the point-to-point channels between the processes are authenticated and reliable.

8.1 A Stronger Model

We assume for the lower bound purposes that the network is synchronous. All processes can communicate with each other over authenticated point-to-point channels that are reliable. We also assume that processes can maintain the complete history of all messages sent and received over these authenticated channels.

Note that this model here is strictly stronger than the model described in Section 4. Lowerbound that holds in this strong network model automatically holds in the weaker model (from Section 4) where the network is asynchronous and the channels are fair.

8.2 High Level Idea

We first show that in this model, the authenticated point-to-point channels can be used instead of MACs. If there exists a protocol to implement signatures using MACs with $n = m$ replicas, then we can implement a reliable broadcast channel using $max\{m, f+1\}$ replicas.

Existing results show that it is impossible to build a reliable broadcast channel in this model with $n \leq 3f$ replicas. We can thus conclude that it is impossible to provide a generalized construction for implementing signatures using MACs with fewer than $3f + 1$ replicas.

8.2.1 MACs and Authenticated Channels

In this synchronous model, MACs can be trivially implemented using authenticated and reliable point-to-point channels. To sign a message, the sender sends the message to the verifier over the authenticated point-to-point channel. To verify that a message is signed, the verifier looks into the history of messages received over the authenticated channel. The message is deemed to have been signed if and only if it was received on the authenticated channel from the signer.

Note that, because the system is synchronous, whenever a message is signed (i.e. sent to the receiver) the message can be successfully verified from the next instance onwards (by which it is guaranteed to be received over the channel).

It is well known that implementing a reliable-broadcast channel in a synchronous setting using authenticated point-to-point channels, without signatures, requires at least $n \geq 3f + 1$ replicas [11].

Theorem 4. *Any construction that implements a generalized signature scheme using MACs require at least $n \geq 3f + 1$ replicas.*

Proof. It is well known that implementing a reliable-broadcast channel in a synchronous setting, without signatures, requires $n \geq 3f + 1$ replicas in the system [11]. However, using signatures it is possible to implement a reliable-broadcast channel with just $n \geq f + 1$ replicas [11].

Suppose that there exists a generalized scheme to implement a signature scheme using a MACs with $n = m$ replicas. Then it is possible to combine these technique with [11] to implement a reliable-broadcast channel using $n \geq max\{m, f + 1\}$ replicas. From [11] it follows that

$$\begin{array}{rrcl} & max\{m, f+1\} & \geq & 3f+1 \\ \Rightarrow & m & \geq & 3f+1 \end{array}$$

□

References

[1] A. Aiyer, L. Alvisi, and R. A. Bazzi. Bounded wait-free implementation of optimally resilient byzantine storage without (unproven) cryptographic assumptions. In DISC '07, pages 443–458, London, UK, sep 2007. Springer-Verlag.

[2] M. Bishop. Computer Security. Addison-Wesley, 2002.

[3] M. Castro. Practical Byzantine Fault Tolerance. PhD thesis, Jan. 2001.

[4] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst., 20(4):398–461, Nov. 2002.

[5] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira. HQ replication: A hybrid quorum protocol for Byzantine fault tolerance. In Proc. 7th OSDI, Nov. 2006.

[6] W. Diffie and M. Hellman. New directions in cryptography. IEEE Trans. on Info Theory, 22(6):644–654, 1976.

[7] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. J. ACM, 32(2):374–382, 1985.

[8] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative byzantine fault tolerance. In Proc. 21st SOSP, 2007.

[9] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative byzantine fault tolerance. Technical Report TR-07-40, University of Texas at Austin, 2007.

[10] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. ACM Trans. Program. Lang. Syst., 4(3):382–401, 1982.

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

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

[13] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing, pages 387–394, New York, NY, USA, 1990. ACM.

[14] B. Schneier. Applied cryptography (2nd ed.): protocols, algorithms, and source code in C. John Wiley & Sons, Inc., New York, NY, USA, 1995.

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