Matrix Signatures: From MACs to Digital Signatures in Distributed Systems

Amitanand S. Aiyer †, Lorenzo Alvisi [1] †, 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]This work was supported in part by NSF grant CSR-PDOS 0720649

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 [15], 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, under different assumptions about the principals, it is possible to get the best of both worlds: a MAC-based implementation of digital-signatures. It is relatively easy to show that such an implementation exists in systems with a specific trusted entity [16]—in the absence of a specific trusted entity, however, the answer is unknown.

The few successful attempts to date at replacing digital signatures with MACs [3, 4, 5, 10, 17] 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 tend to be significantly less intuitive than their signature-based counterparts, so much so that their presentation is often confined to obscure technical reports [3, 11].

In this paper we study the possibility of implementing digital signatures using MACs in a system consisting of any number of untrusted clients and n servers, up to f of which can be Byzantine. We show that, when $n > 3f$, there exists a general implementation of digital signatures using MACs for asynchronous systems with fair channels. We also show that such an implementation is not possible if $n \leq 3f$—even if the network is synchronous and reliable.

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.

Matrix signatures do not qualify as *unique signature schemes* [8]. Depending on the behavior of the Byzantine servers and message delivery delays, the same message signed at different times can produce different signatures, all of which are admissible. Unique signature schemes have a stronger requirement: for every message, there is a unique admissible signature. We show that unique signature schemes can also be implemented using MACs, but that any such implementation requires an exponential number of MACs if f is a constant fraction of n.

In summary, we make four 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 optimally resilient 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).
- We provide an implementation of unique signatures, and show a bound on the number of MACs required to implement them.

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 Byzantine fault tolerant (BFT) state machine replication [3, 4, 5, 10] 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, with the exception of [3], are incomplete.

[1] addresses the problem of allowing Byzantine readers to perform a write back without using digital signatures; however, it uses secret-sharing and relies on having a trusted writer.

Srikanth and Toueg [17] consider the problem of implementing *authenticated broadcast* in a system where processes are subject to Byzantine failures. Their implementation 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: in their protocol, given a message one cannot tell if it was "signed" unless one goes through the history of all messages ever received to determine whether the message was broadcast—an impractical approach if signed messages are persistent and storage is limited. In contrast, we provide signing and verification primitives for an open asynchronous system with any number of clients.

Mechanisms based on unproven number theoretic assumptions, are known to implement digital signatures using local computation without requiring any communication steps [6, 14]. Some also provide unconditional security[9]; but, they bound the number of possible verifiers and allow for a small probability that a verifier may be unable to convince other verifiers that the message was signed.

If there is a trusted entity in the system signatures can be implemented over authenticated channels (or MACs) [16]. In the absence of a known trusted principal, implementing digital signatures locally requires one-way functions [15]. Our results show that even with partial trust in the system, implementing digital signatures is possible without requiring one-way functions.

3 MACs and Digital Signatures

Digital Signatures and MACs allow a message recipient to establish the authenticity of a message. Unlike MACs, digital signatures also allow a message recipient to prove this authenticity to a disinterested third party [2]—non repudiation.

3.1 Digital Signatures

A signature scheme over a set of signers S and a set of verifiers V consists of a signing procedure $\mathcal{S}_{S,V}$ and a verification procedure $\mathcal{V}_{S,V}$:

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

$$\mathcal{V}_{S,V} \quad : \quad \Sigma^* \times \Sigma^* \mapsto \textbf{Boolean} \times \Sigma^* \qquad (2)$$

The signing procedure $\mathcal{S}_{S,V}$ is used to sign a message. It outputs a signature, which can convince the verifier that the message was signed.

The set S contains all the processes that can invoke the signing procedure. The set V contains all processes that may verify a signature in the signature scheme.

The verification procedure, $\mathcal{V}_{S,V}$, takes as input a message and a signature and outputs two values.

The first value is a boolean and indicates whether the verification procedure *accepts* or *rejects* the signature. The second value is a signature, whose role needs some explaining.

The signature schemes we define guarantee that (i) a verifier always accepts signatures that are generated by invoking the signing procedure and that (ii) any message whose signature is accepted was, at some point, signed by a member of S by invoking the signing procedure *although the signature that the verifier accepts may not be the one produced by the signing procedure*. We call these second type of signatures *derivative*.

In traditional, non-distributed, implementations of signatures, one does not expect that a verifier be presented with a *derivative* signature that was not explicitly generated by the signing procedure. In a distributed implementation, and for reasons that will become clear in Section 6, when we discuss the actions that Byzantine nodes can take to disrupt a MAC-based signature scheme, the existence of derivative signatures is the norm rather than the exception, and one needs to allow them in a definition of signature schemes. Furthermore, because the nondeterministic delays and Byzantine behavior of faulty servers, there exist derivative signatures that may nondeterministically be accepted or rejected by the verification procedure. It may then be impossible for a verifier who accepted the signature to prove to another the authenticity of a message.

So, from the perspective of ensuring non repudiation, derivative signatures present a challenge. To address this challenge, we require the verification procedure, every time a verifier v accepts a signed message m, to produce as output a new derivative signature that, by construction, is guaranteed to be accepted by all verifiers. This new signature can then be used by v to authenticate the sender of m to all other verifiers. Note that, if the first output value produced by the verification procedure is **false**, the second output value is irrelevant.

Digital signature schemes are required to satisfy the following properties:
(**Consistency**) A signature produced by the signing procedure is accepted by the verification procedure.

$$\mathcal{S}_{S,V}(msg) = \sigma \Rightarrow \mathcal{V}_{S,V}(msg, \sigma) = (true, \sigma')$$

(**Validity**) A signature for a message m that is accepted by the verification procedure cannot be generated unless a member of S has invoked the signing procedure.

$$\mathcal{V}_{S,V}(msg, \sigma) = (true, \sigma') \Rightarrow \mathcal{S}_{S,V}(msg) \text{ was invoked}$$

(**Verifiability**) If a signature is accepted by the verification procedure for a message m, then the verifier can produce a signature for m that is guaranteed to be accepted by the verification procedure.

$$\mathcal{V}_{S,V}(msg, \sigma) = (true, \sigma') \Rightarrow \mathcal{V}_{S,V}(msg, \sigma') = (true, \sigma")$$

Verifiability is recursively defined; it ensures non-repudiation. If the verification procedure accepts a signature for a given message, then it outputs a signature that is accepted by the verification procedure for the same message. In turn, the output signature can be used to obtain another signature that will be accepted by the verification procedure and so on.

Any digital signature scheme that meets these requirements provides the general properties expected of signatures. Consistency and validity provide authentication; verifiability provides non-repudiation.

3.1.1 Unique Signature Schemes

Unique signature schemes are signature schemes for which only one signature can be accepted by the verification procedure for a given message. If $(\mathcal{S}_{S,V}, \mathcal{V}_{S,V})$ is a unique signature scheme, then, in addition to *consistency, validity and verifiability*, it satisfies:
$\mathcal{V}(msg, \sigma) = (true, \sigma_{proof}) \wedge \mathcal{V}(msg, \sigma') = (true, \sigma'_{proof})$
$\Rightarrow \sigma = \sigma'$

It follows from the definition that $\sigma_{proof} = \sigma'_{proof} = \sigma = \sigma'$, implying that, for unique signatures, the signature produced in output by the verification procedure is redundant. It also follows from the definition and the consistency requirement that unique signatures have deterministic signing procedures.

3.2 Message Authentication Codes

MACs are used to implement authentication between processes. A message authentication scheme consists of a signing procedure $\mathcal{S}_U$ and a verifying procedure $\mathcal{V}_U$.

$$\mathcal{S}_U \quad : \quad \Sigma^* \mapsto \Sigma^* \qquad (3)$$
$$\mathcal{V}_U \quad : \quad \Sigma^* \times \Sigma^* \mapsto \{\textbf{true}, \textbf{false}\} \qquad (4)$$

The signing procedure $\mathcal{S}_U$ takes a message and generates a MAC for that message. The verification procedure $\mathcal{V}_U$ takes a message along with a MAC and checks if the MAC is valid for that message.

For a given MAC scheme, the set U contains all processes that can generate and verify MACs for the scheme.

MACs are required to ensure authentication, but not non-repudiation. Formally, they are required to satisfy:
(**Consistency**) A MAC generated by the signing procedure will be accepted by the verifying procedure.

$$\mathcal{S}_U(msg) = \mu \Rightarrow \mathcal{V}_U(msg, \mu) = \mathbf{true}$$

(**Validity**) A MAC for a message m that is accepted by the verification procedure cannot be generated unless a member of U has invoked the signing procedure.

$$\mathcal{V}_U(msg, \mu) = \mathbf{true} \Rightarrow \mathcal{S}_U(msg) \text{ was invoked}$$

3.3 Discussion

Keys, Signatures, and MACs Formal definitions of signature schemes typically include signing and verification keys. In our work we omit the keys for simplicity and assume they are implicitly captured in $\mathcal{S}_{S,V}$ and $\mathcal{V}_{S,V}$. In our setting, S is be the set of processes that know the key needed to sign and V is the set of processes that know the key needed to verify.

MACs are also typically defined with reference to a symmetric secret key K that is used to generate and verify MACs. In our setting, processes that know K are members of the set U of signers and verifiers. In proving a lower bound on the number of MAC schemes needed to implement unique signatures, we find it convenient to identify a MAC scheme with the key K it uses. In this case, we distinguish between the name of the key, K, and the value of the key k as different schemes might use the same key value.

Signers and Verifiers Since we will be considering Byzantine failures of servers and clients (participants), the composition of the sets S or U for a given scheme might change because a participant can give the secret signing key to another participant. To simplify the exposition, we assume that the sets of signers (verifiers) include any participant that can at some point sign (verify) a message according to the scheme.

Semantics 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.[2]

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 procedure. 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 procedures.

Clients communicate with the servers over authenticated point-to-point channels. Inter-server

[2]Our implementations use only finitely many MACs, consequently the probability of breaking our implementation can be made arbitrarily small if the probability of breaking the underlying 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.

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.

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 [12]. Remaining processes follow the protocol specification.

5 Signatures using MACs

We first present the high level idea assuming two trusted entities in the system. One trusted entity acts as a signing-witness and one acts as a verifying-witness. The two witnesses share a secret-key $\mathcal{K}$ that is used to generate and verify MACs.

Signing a message A signer delegates to the signing witness the task of signing a message. This signing witness generates, using the secret key K, a MAC value for the message m to be signed and sends the MAC value to the signer. This MAC-signature certifies that the signer s wants to sign m. It can be presented by a verifier to the verifying-witness to validate that s has signed m.

Verifying a signature To verify that a MAC-signature is valid, a verifier (client) delegates the verification task to the verifying witness. The verifying witness computes, using the secret key K, the MAC for the message and verifies that it is equal to the MAC presented by the verifier. If it is, the signature is accepted otherwise, it is rejected.

Since the two witnesses are trusted and only they know the secret key K, this scheme satisfies *consistency*, *validity* and *verifiability*.

6 A Distributed Signature Implementation

In an asynchronous system with $n \geq 3f + 1$ servers, it is possible to delegate the tasks of the signing witness and the verifying witness to the servers. However, achieving non-repudiation is tricky.

6.1 An Illustrative Example

Consider a scheme, for $n = 3f + 1$, where each server i has a secret key K_i is used to generate/verify MACs. The "signature" is a vector of n MACs, one for each server.

To sign a message, the signer contacts the servers to collect the MACs. However, due to asynchrony, it cannot collect more than $(n-f) = (2f + 1)$ MACs.

To verify a signature, the verifier contacts the servers to determine which MACs are correct. Since, up to f Byzantine nodes could have sent wrong values to the signer, ensuring *consistency* requires that the verifier accept the "signature" even if only $f+1$ MAC are accepted by the servers.

This allows the adversary to fool a non-faulty verifier into accepting a vector that contains only one correct MAC value. If that happens, the verifier will not be able to convince other verifiers that the message was signed.

6.2 Matrix Signatures

To deal with the difficulties raised in the illustrative example, we propose *matrix signatures*. A matrix signature consists of n^2 MAC values arranged in n rows and n columns, which together captures the servers' collective knowledge about the authenticity of a message.

We have $n \geq 3f + 1$ witness servers which share pairwise secret keys that are used to generate/verify MACs. Each witness server 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 signature being used is a matrix of $n \times n$ MACs as opposed to a single MAC value. Each MAC value in the matrix is calculated using a secret key $K_{i,j}$ shared between a signing-witness-server i and a verifying-witness-server j.[3]

The i^{th} row of the matrix-signature consists of the MACs generated *by* the i^{th} signing-witness-server. The j^{th} column of the matrix-signature consists 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.

We distinguish between *valid* and *admissible* matrix signatures:

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.

6.3 Protocol Description

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

6.3.1 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.

$$\begin{array}{|cccc|}\hline h_{1,1} & h_{1,2} & \mathbf{h_{1,3}} & h_{1,4} \\ \mathbf{h_{2,1}} & \mathbf{h_{2,2}} & \mathbf{h_{2,3}} & \mathbf{h_{2,4}} \\ h_{3,1} & h_{3,2} & \mathbf{h_{3,3}} & h_{3,4} \\ h_{4,1} & h_{4,2} & \mathbf{h_{4,3}} & h_{4,4} \\ \hline \end{array}$$

(a) A Matrix-signature

$$\begin{array}{|cccc|}\hline h_{1,1} & h_{1,2} & h_{1,3} & h_{1,4} \\ h_{2,1} & h_{2,2} & h_{2,3} & h_{2,4} \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ \hline \end{array}$$

(b) A Valid Signature

$$\begin{array}{|cccc|}\hline ? & h_{1,2} & ? & ? \\ ? & h_{2,2} & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ \hline \end{array}$$

(c) An Admissible Signature

Figure 1: Example Matrix-signatures

The matrix-signature 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 the faulty servers.

6.3.2 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 $(2f+1)$ $\langle$ADMIT$, \ldots\rangle$ responses to accept the signature, or it receives $(f+1)$ $\langle$REJECT$\rangle$ responses to reject the signature as not *valid*.

Regenerating a valid signature : Receiving $(2f+1)$ $\langle$ADMIT$, \ldots\rangle$ responses does not guarantee that the signature being verified is *valid*. If some

[3]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.

```
Signature Client-Sign (Msg $Msg$) {
    $\forall i : \sigma_{Msg,S}[i][] := \perp$
    send   $\langle$SIGN, $Msg, S\rangle$ to all
    do {
        // Collect MAC-rows from the servers
        rcv    $\langle\sigma_i[1 \ldots n]\rangle$ from server $i$
        $\sigma_{Msg,S}[i][1 \ldots n] := \sigma_i[1 \ldots n]$
    } until  received from $\geq 2f+1$ servers
    return $\sigma_{Msg,S}$
}

(bool, Signature) Client-Verify(Msg $Msg$, Signer $S$,
                                        Signature $\sigma$) {
    $\forall i : \sigma_{new}[i][] := \perp$
    $\forall i :$ resp$[i] := \perp$
    send   $\langle$VERIFY, $Msg, S, \sigma[][]\rangle$ to all
    do {
        rcv $\langle$ADMIT, $\sigma_i[1 \ldots n]\rangle$ or $\langle$REJECT$\rangle$ from server $i$
        if received $\langle$ADMIT, $\sigma_i[1 \ldots n]\rangle$ {
            $\sigma_{new}[i][1 \ldots n] := \sigma_i[1 \ldots n]$
            resp$[i]$ := ADMIT
            // Accept: If $(2f+1)$ admit it
            if ( Count(resp, ADMIT) $\geq 2f+1$ )
                return (true, $\sigma_{new}$);
        } else {
            if (resp$[i] = \perp$)
                resp$[i]$ := REJECT
            // Reject: If at least one non-faulty server rejects
            if ( Count(resp, REJECT) $\geq f+1$ )
                return (false, $\perp$);
        }

        // If cannot accept or reject: retry with $\sigma_{new}$
        if ( Count(resp, ADMIT)
                + Count(resp, REJECT) $\geq (n-f)$ ) {
            send   $\langle$VERIFY, $Msg, S, \sigma_{new}[][]\rangle$ to
                                        { $r$ : resp$[r] \neq$ ADMIT}
        }
    } until (false)
}

void Signing-Witness-Server(Id $i$) {
    while(true) {
        rcv $\langle$SIGN, $Msg, S\rangle$ from $S$
        $\forall j$ : compute $\sigma_i[j]$ := MAC($\mathcal{K}_{i,j}$, $S : Msg$)
        send $\langle\sigma_i[1 \ldots n]\rangle$ to $S$
    }
}

void Verifying-Witness-Server(Id $j$) {
    while(true) {
        rcv $\langle$VERIFY, $Msg, S, \sigma\rangle$ from $V$
        correct_cnt := $|\{i : \sigma[i][j] ==$
                        MAC($\mathcal{K}_{i,j}, S : Msg$)$\}|$
        if (correct_cnt $\geq f+1$)
            $\forall l$ : compute $\sigma_j[l]$ := MAC($\mathcal{K}_{j,l}$, $S : Msg$)
            send $\langle$ADMIT, $\sigma_j[1 \ldots n]\rangle$ to $V$
        else
            send $\langle$REJECT$\rangle$ to $V$
    }
}
```

Figure 2: Matrix-signatures

of these responses are from Byzantine nodes, the same signature could later fail the verification if the Byzantine nodes respond differently.

Verifiability requires that that if a signature passes the verification procedure, then the verifier gets a signature that will always pass the verification procedure. This is accomplished by constructing a new signature, that is a *valid* signature.

Each witness-server acts both as a verifying-witness-server and a signing-witness-server. Thus, when a witness-server admits a signature (as a verifying-witness-server), it also re-generates the corresponding row of MAC-values (as a signing-witness-server) and includes that in the response. Thus, if a verifier collects $(2f+1)$ $\langle$ADMIT$, \ldots\rangle$ responses, it receives $(2f+1)$ rows of MAC-values, which forms a *valid* signature.

Ensuring termination : The verifier may receive $(n-f)$ responses, and still not have enough admit responses or enough reject responses, to decide. This can happen if the signature being verified, σ, is maliciously constructed such that some of the columns are bad. This can also happen if the signature σ is *valid*, but some non-faulty servers are slow and Byzantine servers, who respond faster, reject it.

To ensure that the verifier gets $(2f+1)$ $\langle$ADMIT$, \ldots\rangle$ responses it retries by sending σ_{new}, each time σ_{new} is updated, to all the servers that have not sent an $\langle$ADMIT$, \ldots\rangle$ response. Eventually, it either receives $(f+1)$ $\langle$REJECT$\rangle$ responses from different servers (which guarantees that σ was not *valid*), or it receives $(2f+1)$ $\langle$ADMIT$, \ldots\rangle$ responses (which ensures that the regenerated signature, σ_{new} is *valid*).

6.4 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.*

Proof. A valid signature by definition has at least $(f+1)$ rows of all correct MAC-values. Every column has at least $(f+1)$ correct MAC values. □

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 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 $\langle$REJECT$\rangle$ message. When all non-faulty servers respond, the verifier will have $(2f + 1)$ $\langle$ADMIT$, \ldots\rangle$ 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 $\langle$REJECT$\rangle$ message. On receiving $(n - f)$ responses, the verifier will have $(n - 2f) \geq (f + 1)$ $\langle$REJECT$\rangle$ 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. Consider the first admissible signature, σ, that is generated for Msg. By the definition, it should have at least $(f + 1)$ correct MAC values in a column corresponding to a non-faulty server (say j). At least one of these MAC values is in a row that corresponds to a non-faulty server (say i). The key $K_{i,j}$ is only known to the non-faulty servers i and j.

Only server i might generate the MAC value and give it to a client. If the message was not signed by the signer, then the MAC must have been generated as part of the verification procedure. In the verification procedure, a non-faulty server i only generates the MAC value, if it has already received an admissible matrix-signature that has $(f + 1)$ correct MAC values in column i. This is not possible because σ is the first admissible signature generated. □

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 $(2f + 1)$ $\langle$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 verifier accepts that S has signed Msg, then it can convince every other non-faulty verifier that S has signed Msg.*

Proof. A non-faulty verifier, v_1, accepts that a message is signed, only if it passes the verification procedure. The newly generated signature that it gathers, σ_{new}, is *valid* and will pass the verification for any non-faulty verifier v_2, convincing the verifier (v_2) that the message was signed. □

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 3 and 7. □

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. Suppose that the verifier does not terminate even when it gets the responses from all the non-faulty servers. It cannot have received more than f $\langle$REJECT$\rangle$ responses. Thus, it would have received at least $(f + 1)$ $\langle$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 the verifier retries with 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. □

6.5 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, or assumptions about a non-adaptive adversary) then the signature scheme realized will be secure only as long as those assumptions hold.

7 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 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. Preventing such a misuse will require the use of threshold signature schemes, that are at least as costly as digital signatures.

7.2 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 $(2f+1)$.

Such systems cannot be realized in an asynchronous setting unless the total number of servers in the system $n \geq 4f+1$.

Even under those circumstances, this approach may not provide the recursive non-repudiation required by digital signatures.

7.3 Vector of n MACs

Consider the following scheme, for illustration, when $n = 4f+1$:

Each server i has a secret key K_i that is only known to server i. The signature consists of a (possibly incomplete) vector of n MACs, one for each K_i.

To sign a message, the signer contacts the servers and collects $(n-f)$ individual MAC values to form the vector.

To verify the vector, the verifier contacts the individual servers to verify if each of the MAC values is correct or not. The verifier accepts the "signature" if more than a threshold of $(n-2f) = (2f+1)$ servers respond that the MAC is correct.

Analysis Consider a verifier (say v_1) who accepts the "signature" because at least $(n-2f) = (2f+1)$ servers respond that their MAC is correct. If f of these responses were from faulty servers, then v_1 may not be able to convince a second verifier (say v_2) that the message was signed.

The second verifier, v_2 can only get up to $(n-3f) = (f+1)$ responses that the MAC is correct. Even if the second verifier is willing to accept (based on $f+1$ responses) that the message was signed, he will not be able to convince a third verifier about the same. This is because if f of the servers responding to the second verifier were faulty, then the third may not get any more than 1 response that the MAC is correct.

Thus this approach also does not provide the recursive non-repudiation as required by digital signatures.

8 The $n \leq 3f$ Case

We show that a generalized scheme to implement the properties of signatures using MACs is not possible if $n \leq 3f$. The lower bound holds for a much stronger system 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 that the network is synchronous and processes communicate with each other over authenticated and reliable point-to-point channels. We also assume that processes can maintain the complete history of all messages sent and received over these authenticated channels.

This model, without any further set-up assumptions, is strictly stronger than the model described in Section 4. A lowerbound that holds in this stronger model automatically holds in the weaker model (from Section 4) where the network is asynchronous and the channels are only guaranteed to be fair.

In this model, we show that it is possible to implement MACs over authenticated channels. If, in this model, signatures can be implemented using MACs with $n \leq 3f$, then they can also be implemented over authenticated channels with $n \leq 3f$. Using signatures, it is possible to implement a reliable-broadcast channel with just $n \geq f + 1$ replicas [13]. So, it would be possible to implement a reliable-broadcast channel assuming a MAC-based implementation of signatures with n servers, where $(f + 1) \leq n \leq 3f$.

But, it is well known that implementing a reliable-broadcast channel in a synchronous setting over authenticated point-to-point channels, without signatures, requires $n \geq 3f + 1$ [13]. We conclude that implementing signatures with MACs is not possible if $n \leq 3f$.

It only remains to show that MACs can be implemented in the strong model.

Lemma 9. *In the strong system model, MACs can be implemented amongst any set of servers, U, using authenticated point-to-point channels between them.*

Proof. (outline) To sign a message, the sender sends the message, tagged with the identity of set U, to all the servers in U over the authenticated point-to-point channels. Since the network is synchronous, these messages will be delivered to all the servers in U within the next time instance. To verify that a message is signed, a 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. $\square$

9 Unique Signatures

We provide an implementation of unique signatures when $n > 3f$. By Lemma 9, it follows that the implementation is optimally resilient. Our implementation requires an exponential number of MAC values. We show that any implementation of unique signatures requires that an exponential number of MAC values be generated if f is a constant fraction of n. The implementation is optimal if $n = 3f + 1$; the number of MAC values it requires exactly matches the lower bound when $n = 3f + 1$.

Our implementation uses *unique MAC schemes*. These are schemes for which only one MAC value passes the verification procedure for a given message and that always generate the same MAC value for a given message. Many widely used MAC schemes are unique MAC schemes, including those that provide unconditional security.[4]

[4]For many MAC schemes the verification procedure consists of running the MAC generation (signing) procedure on the message and comparing the resulting MAC value with the MAC value to be verified. Since the signing procedure is typically deterministic, only one value can pass the verification procedure.

9.1 A Unique Signature Implementation

We give an overview of the implementation. The detailed protocol presented in Figure 3.

In our unique signature scheme, the signing procedure generates signatures which are vectors of $N = \binom{n}{2f+1}$ MAC values, one for each subset of $2f+1$ servers. The i'th entry in the vector of signatures can be generated (and verified) with a key K_i that is shared by all elements of the i'th subset G_i of $2f+1$ servers, $1 \leq i \leq \binom{n}{2f+1}$. For each K_i, the MAC scheme used to generate MAC values is common knowledge, but K_i is secret (unless divulged by some faulty server in G_i).

To sign a message, the signer sends a request to all the servers. A server generates the MAC values for each group G_i that it belongs to and sends these values to the signer. The signer collects responses until it receives $(f+1)$ identical MAC values for every group G_i. Receiving $f+1$ identical responses for every G_i is guaranteed because each G_i contains at least $f+1$ correct servers. Also, receiving $(f+1)$ identical MAC values guarantees that the MAC value is correct because one of the values must be from a non-faulty server.

To verify a unique signature, the verifier sends the vector of N MACs to all the servers. The i'th entry M_i is verified by server p if $p \in G_i$ and M_i is the correct MAC value generated using K_i. A verifier accepts the signature if each entry in the vector is correctly verified by $f+1$ servers. The verifier rejects a signature if one of its entries is rejected by $f+1$ servers. Since the underlying MAC schemes are unique and each G_i contains $2f+1$ servers, a signature is accepted by one correct verifier if an only if it is accepted by every other correct verifier and no other signature is accepted for a given message.

9.2 Correctness

We now show that this algorithm implements a Unique Signature Scheme.

Lemma 10 (Uniqueness)**.** *The signature procedure, invoked by a non-faulty signer, always produces the same signature for the same message.*

Proof. Each MAC implementation results in a unique MAC value for the message, thus the vector of N MACs is unique for every message. □

Lemma 11. *The signature generated by the signature procedure in Figure 3 contains the correct MAC values for every position in the vector.*

Proof. The signer has to wait for $(f+1)$ matching MAC values for each of the N positions in the vector of MACs. Since at least one of the $(f+1)$ responses is from a non-faulty server, each of the MAC in the vector is correct. □

Lemma 12 (Consistency)**.** *Signature consisting of correct MAC values at each of the N positions will pass the verification procedure.*

Proof. A non-faulty server will never send WRONG if the MAC value is correct. Thus, the verification procedure can get no more than f WRONG responses (from the faulty servers) for any of the N MACs in the vector.

Each MAC in the vector can be verified by $(2f+1)$ servers, at least $(f+1)$ of which are non-faulty. These non-faulty servers will send a response that the MAC value is CORRECT. Thus, on receiving the responses from all non-faulty servers, the verification procedure will accept the signature as it will have $(f+1)$ CORRECT responses for each of the N positions in the vector. □

Lemma 13 (Validity)**.** *A vector containing all N correct MACs cannot be generated unless the signer invokes the signing procedure.*

Proof. For any set of f faulty servers, there is a group of $(2f+1)$ nodes that contains only non-faulty servers in it (because $n \geq 3f+1$). The MAC value corresponding to this group can only be generated correctly by one of these non-faulty servers, which will only do so if they receive a SIGN message from the signer. □

Lemma 14 (Verifiability)**.** *If a signature passes the verification procedure in Figure 3 then the signature contains the correct MAC value for every position in the vector.*

Proof. The verification procedure has to waits for $(f+1)$ CORRECT responses for each of the N MACs in the vector. Since at least one of the $(f+1)$ responses is from a non-faulty server, and it would send a CORRECT response only if the MAC is correct, each of the N MACs in the vector is correct. $\square$

Theorem 4. *The algorithm presented in Figure 3 implements a Unique Signature Scheme.*

9.3 Complexity of unique signature implementations

We derive a lower bound on the number of MAC schemes needed to implement a MAC-based unique signature scheme using n servers, f of which are Byzantine. We identify the MAC schemes used in the implementation with their secret keys and, in what follows, we refer to K_i instead of *the MAC scheme that uses* K_i. We consider a general implementation that uses M secret keys. Every key K_i is shared by a subset of the servers; this is the set of servers that can generate and verify MAC values using K_i. We do not make any assumptions on how a signature looks. We simply assume that the signing procedure can be expressed as a deterministic function $\mathcal{S}(msg, k_1, k_2, \ldots, k_M)$ of the message to be signed (msg), where $k_1, \ldots, k_M$ are the values of the keys $K_1, \ldots, K_M$ used in the underlying MAC schemes.

The lower bound proof relies on two main lemmas which establish that (1) every key value must be known by at least $2f+1$ servers, and (2) for any set of f servers, there must exist a key value that is not known by any element of the set. Then, we use a combinatorial argument to derive a lower bound on the number of keys.

Since we are proving a lower bound on the number of keys, we assume that the signature scheme uses the minimum possible number of keys. It follows, as shown in the following lemma, that no key is redundant. That is, for every key K_i, the value of the signature depends on the value of K_i for some message and for some combination of the values of the other keys.

Lemma 15 (No key is redundant). *$\forall i$, $\exists msg$, $k_1, k_2, \ldots k_{i-1}, k_i^\alpha, k_i^\beta, k_{i+1}, k_{i+2}, \ldots, k_M$:*
$\mathcal{S}(msg, k_1, k_2, \ldots, k_{i-1}, k_i^\alpha, k_{i+1}, \ldots, k_M) = \sigma_1$, $\mathcal{S}(msg, k_1, k_2, \ldots, k_{i-1}, k_i^\beta, k_{i+1}, \ldots, k_M) = \sigma_2$,
and $\sigma_1 \neq \sigma_2$

Proof. (Outline) If the signature produced for a message is always independent of the key K_i, for every combination of the other keys. Then, we can get a smaller signature implementation, by using a constant value for K_i, without affecting the resulting signature. $\square$

Now, we can establish the two main lemmas.

Lemma 16 ($2f+1$ servers know every key). *At least $(2f+1)$ servers know the value of K_i.*

Proof. We show by contradiction that if K_i is only known by a group G, $|G| \leq 2f$, servers. the signature scheme is not unique. If $|G| \leq 2f$, G is the union of two disjoint sets A and B of size less than $f+1$ each. From Lemma 15, $\exists msg$, $k_1, k_2, \ldots k_{i-1}, k_i^\alpha, k_i^\beta, k_{i+1}, k_{i+2}, \ldots, k_M$:
$\mathcal{S}(msg, k_1, k_2, \ldots, k_{i-1}, k_i^\alpha, k_{i+1}, \ldots, k_M) = \sigma_1$, $\mathcal{S}(msg, k_1, k_2, \ldots, k_{i-1}, k_i^\beta, k_{i+1}, \ldots, k_M) = \sigma_2$,
and $\sigma_1 \neq \sigma_2$

Consider the following executions, where message msg is being signed. In all executions, the value of K_j is k_j for $j \neq i$.

- (Exec α) The symmetric key value for K_i is k_i^α. All servers behave correctly. The resulting signature value is σ_1.
- (Exec α') The symmetric key value for K_i is k_i^α. Servers not in B behave correctly. Servers in B set the value of K_i to be k_i^β instead of k_i^α. The resulting signature value is also σ_1 because the signature scheme is unique and tolerates up to f Byzantine failures and $|B| \leq f$.
- (Exec β) The symmetric key value for K_i is k_i^β. All servers behave correctly. The resulting signature value is σ_2.
- (Exec β') The symmetric key value for K_i is k_i^β. Servers not in A behave correctly. Servers in A set the value of K_i to be k_i^α

instead of k_i^β. The resulting signature value is also σ_2 because the signature scheme is unique and tolerates up to f Byzantine failures and $|A| \leq f$.

Executions α' and β' only differ in the identities of the faulty servers and are otherwise indistinguishable to servers not in G and to clients. It follows that the same signature value should be calculated in both cases, contradicting the fact that $\sigma_1 \neq \sigma_2$. □

Lemma 17 (Faulty servers do not know some key). *For every set of f servers, there exists a secret key K_i that no server in the set knows.*

Proof. If a given set of f servers has access to all the M secret keys, then, if all the elements of the set are faulty, they can generate signatures for messages that were not signed by the signer, violating validity. □

We now show establish a lower bound on the number of keys required by a MAC-based unique signature implementation.

Theorem 5. *The number of keys used by any MAC-based implementation of a unique signature scheme is $\geq \binom{n}{f} / \binom{n-(2f+1)}{f}$*

Proof. Let $G = (\mathcal{K}, R, E)$ be a bipartite graph over the set $\mathcal{K}$ of M keys and the set of servers R with an edge between K_i and server j if and only if j is not assigned the key K_i by the unique signature scheme (that is j is not given the value of K_i as part of the scheme). Since every K_i is known by at least $(2f+1)$ servers (Lemma 16) it follows that, in G, the degree of vertex K_i is at most $\big(n-(2f+1)\big)$.

Let $G' = (\mathcal{K}, \mathcal{A}, E')$ be the bipartite graph over $\mathcal{K}$ and the set $\mathcal{A}$ of f-subsets of R. The set $\mathcal{A}$ consists of those subsets of R that could be controlled by the adversary. An edge (K, A) belongs to E' if and only if (K, a) belongs to E for every $a \in A$.

$$(K, A) \in E' \iff \forall a \in A : (K, a) \in E$$

If a node $K_i \in \mathcal{K}$ has degree d_i in G, K_i would have, in G', degree $d'_i = \binom{d_i}{f} \leq \binom{n-(2f+1)}{f}$.

From Lemma 17, it follows that for each $A \in \mathcal{A}$, $d'_A \geq 1$ in G'.

In any graph, the sum of out-degree must be equal to the sum of in-degree. From graph G', it follows that:

$$\begin{aligned} \sum_{k_i \in \mathcal{K}} d'_i &= \sum_{A \in \mathcal{A}} d'_A \\ \Rightarrow \quad \sum_{k_i \in \mathcal{K}} \binom{n-(2f+1)}{f} &\geq \sum_{A \in \mathcal{A}} 1 \\ \Rightarrow \quad M &\geq \binom{n}{f} / \binom{n-(2f+1)}{f} \end{aligned}$$

□

It follows that for $n = 3f+1$, the unique signature implementation described in Section 9.1 is optimal. In general, if the fraction of faulty nodes $\frac{f}{n} > \frac{1}{k}$, for $k \geq 3$, then the number of MACs required is at least $(\frac{k}{k-2})^f$.

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, MIT, 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.

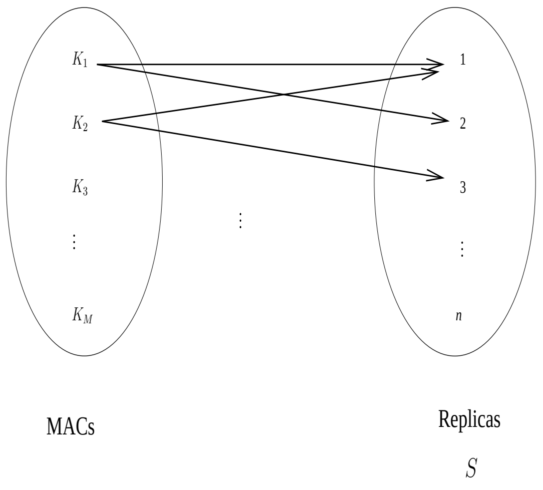
[6] W. Diffie and M. Hellman. New directions in cryptography. IEEE Trans. on Info Theory, 22(6):644–654, 1976.

[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.

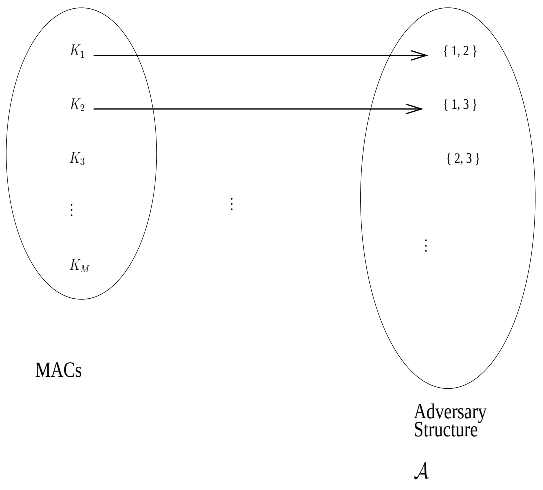
[8] O. Goldreich. Foundations of Cryptography, volume Basic Tools. Cambridge University Press, 2001.

[9] G. Hanaoka, J. Shikata, Y. Zheng, and H. Imai. Unconditionally secure digital signature schemes admitting transferability. In ASIACRYPT '00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security, pages 130–142, London, UK, 2000. Springer-Verlag.

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



(a) Graph G



(b) Graph G'

Figure 4: Example Bipartite Graphs G and G'

[11] 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.

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

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

[14] 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.

[15] 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.

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

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

```
Signature Client-Sign (Msg $Msg$) {
    $\forall s : 1 \leq s \leq n, 1 \leq g \leq N,$ MACS$[s][g] :=\perp$
    $\forall g : 1 \leq g \leq N, signature[g] :=\perp$

    send $\langle$SIGN$, Msg, signer\rangle$ to all.

    // gather responses until we have $f+1$ matching
    // MAC values for each of the $N$ macs.
    while ( $\exists g : signature[g] =\perp$ )
    {
        on receive $\langle$MACS$, macs_i[1 \ldots N]\rangle$ from server $i$.
            foreach $g = 1 \ldots N$
                if ($i \in G_g$)
                    $MACS[i][g] := macs_i[g]$
                if ( $\exists x \neq\perp$: $|\{i :$ MACS$[i][g] = x\}| \geq f+1$ )
                    $signature[g] := x$
    }
    return $signature[1 \ldots N]$;
}

(bool, Signature) Client-Verify(Msg $Msg$, Signer $S$,
                                Signature $signature[1 \ldots N]$) {
    {
        $\forall s : 1 \leq s \leq n, received\_responses[s] := false$;
        $\forall g : 1 \leq g \leq N, yes\_count[g] := 0$;
        $\forall g : 1 \leq g \leq N, no\_count[g] := 0$;

        send $\langle$VERIFY$, Msg, signer, signature[1..N]\rangle$ to all.

        // gather responses until we have $f+1$ matching
        // responses for each of the $N$ macs.
        while ( $\exists g : yes\_count[g] < f+1$ )
        {
            on receive $\langle$RESULT$, res_i[]\rangle$ from server $i$.
                foreach $g := 1 \ldots N$
                    if ($i \notin G_g$)
                        continue;
                    else if ($res_i[g] =$ CORRECT)
                        $yes\_count[g] = yes\_count[g] + 1$;
                    else if ($res_i[g] =$ WRONG)
                        $no\_count[g] = no\_count[g] + 1$;

                    // If even one MAC-value is bad, we fail.
                    if ($no\_count[g] \geq f+1$)
                        return (false, signature);
        }

        return (true, signature);
    }
```

```
void Signing-Witness-Server(Id $i$) {
    $\forall g : 1 \leq g \leq N, macs_i[g] =\perp$

    while(true) {
        rcv $\langle$SIGN$, Msg, signer\rangle$ from signer

        for each g in $1 \ldots N$
            if( $i \notin G_g$ )
                $macs_i[g] :=\perp$;
            else
                $macs_i[g] :=$ HMAC$(signer : Msg, K_g)$;

        send $\langle$MACS$, macs_i[1 \ldots N]\rangle$ to signer
    }
}

void Verifying-Witness-Server(Id $j$) {
    $\forall g : 1 \leq g \leq N, result_i[g] :=\perp$

    while(true) {
        rcv $\langle$VERIFY$, Msg, signer, signature[1 \ldots N]\rangle$
                        from a verifier

        foreach $g$ in $1 \ldots N$
            if( $i \notin G_g$ )
                $result_i[g] :=\perp$
            else if ($signature[g] =$
                        HMAC$(signer : Msg, K_g)$)
                $result_i[g] :=$ CORRECT;
            else
                $result_i[g] :=$ WRONG;

        send $\langle$RESULT$, result_i[1 \ldots N]\rangle$ to the verifier
    }
}
```

Figure 3: Signing and Verifying procedures for MAC based Unique Signatures