# Low Latency and Congestion Broadcast Authentication in Fixed Topology Networks

Haowen Chan Adrian Perrig

December 22, 2008
CMU-CyLab-08-014

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

# Low Latency and Congestion Broadcast Authentication in Fixed-Topology Networks

Haowen Chan    Adrian Perrig

20 December 2008

**Abstract**

We present several latency optimizations on two classes of broadcast authentication schemes for networks where the topology and the identities of the receiving nodes are known a-priori. Let $n$ be the number of receivers in the network. The first class of protocols involves the construction and dissemination of a hash tree with message authentication codes (MACs) at the leaves. We show that this authentication method can be performed on linear network topologies with $n$ latency and at most $2\lceil\log_2 n\rceil$ communication overhead per node. On a fully-connected network, this method can be implemented with $3\lceil\log_2 n\rceil + 1$ latency and at most $5\lceil\log_2 n\rceil$ communication overhead per node.

The second class of protocols uses an idea related to TESLA where a single MAC for the broadcast message is released, and the key for the MAC is released (or reconstructed) later, once all receivers have received the MAC. Like TESLA, this class of protocols requires an additional minor bootstrapping assumption where an hash chain anchor value needs to be preloaded onto all receivers and all receivers are required to have a consistent view of the current hash chain status. However, unlike TESLA, in the new scheme we can remove the loose time synchronization requirement completely. We show that this allows messages to be authenticated on the any topology in two passes and low constant communications overhead per node. Hence, we can perform the protocol on the linear topology with with $2n$ latency and on the fully connected topology with $4.32\log_2 n + 2$ latency.

## 1 Introduction

One-to-many message authentication is a useful basic security primitive for computer networks. Intuitively, the problem can be described as follows: how can a sender $s$ craft a message such that any receiver will know that the message is from $s$? The general broadcast authentication problem focuses on protocols that allow the sender to authenticate the message to any other party. With the advent of recent resource-constrained applications such as sensor networks, vehicular networks and high volume Internet-based publish-subscribe services, the more specific problem of efficient authentication among known broadcast groups has become increasingly important. In this problem, the sender knows the exact set of receivers that will receive their messages, and is able to set up prior initialization steps with them.

We consider the broadcast authentication problem in fixed-topology networks. In this problem, the in-network identities of the receivers and their mutual communication relationships are common knowledge. We examine efficient protocols for highly resource constrained platforms: in particular, our protocols should not use relatively more expensive asymmetric key operations; the messages should be able to be quickly verified, and the communication overheads of the protocols should not be high.

We derive two families of authentication protocols. The first family of protocols uses the idea of a hash tree built over individual message authentication codes. The basic idea was first first proposed by Chan and Perrig and was for Tree Topologies [3]. While asymptotic performance bounds were computed in the original work, simply applying the original algorithm on a spanning tree of the network does not yield the best latency or communications bounds. We show optimizations for the linear and the fully-connected topologies that greatly reduce latency compared with a naive implementation of the original algorithm.

The second family of protocols is a new construction related conceptually to TESLA [8] but improving on it by the addition of efficient receiver-to-sender authenticated synchronization mechanisms. This allows the new protocol to remove the time synchronization requirement of TESLA. We investigate a further optimization that cuts down on the number of message rounds by encrypting the TESLA key in such a way that the receivers themselves can recover the key after all receivers have received the message, without requiring the sender to explicitly disseminate the key.

The different variants of the two protocols present several useful points in the trade-off space for broadcast and multicast authentication protocols. These results are useful both in direct application and as building blocks for protocol designers seeking low-cost primitives for one-to-many message authentication in their secure protocols.

We briefly describe some related work in Section 2, and some basic preliminary concepts in Section 3. The problem definition and the fixed topologies that are considered are described in Sections 4 and 5 respectively. We give a basic exhaustive-MAC approach in Section 6 as an example of how the metrics we consider can make the problem challenging. We describe our first family of protocols, the hash-tree-based protocols, in Section 7. We describe our second family of protocols, the hash-chain-based protocols, in Section 8. The complete set of analytical latency and congestion bounds for all variants of our algorithms are tabulated in Section 9.

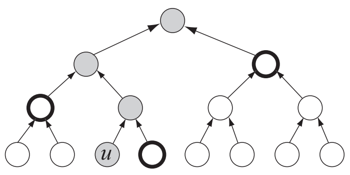


Figure 1: Authentication path of vertex $u$

## 2 Related Work

Ralph Merkle first described the use of a hash tree as a batch authenticator for multiple values [7]. In this original application, the hash tree is used for a series of one-time signatures by releasing authentication paths for each of its leaves in order. Subsequent work on hash trees have concentrated on optimizing this left-to-right traversal process for the signer. Examples of such optimizations include work by Buchmann et al. [2], Berman et al. [1], Michael Szydlo [10], and Jakobsson et al. [4]. However, the overhead of using one-time signatures in this application is extremely high.

The concept of constructing hash trees over message authentication codes for the purposes of multicast/broadcast authentication was first proposed by Chan et al. in 2008 [3]. The construction was only described for a tree topology and only asymptotic congestion bounds were described; precise latency and congestion optimizations for specific topologies were not investigated.

In the area of broadcast authentication for sensor networks, Perrig et al. propose $\mu$TESLA [9], which unfortunately requires lose time synchronization. Improvements to $\mu$TESLA have been proposed, but they all require lose time synchronization [5]. Luk et al. propose families of broadcast authentication mechanisms [6], but the communication overhead of their one-time signature schemes can be quite substantial. Our protocols are the first variants of TESLA that require no time synchronization.

## 3 Preliminaries

We briefly describe the cryptographic and algorithmic primitives we use in this work.

### 3.1 Hash Tree

The hash tree is a cryptographic data structure for integrity and authentication. It is constructed over a set of leaf values by repeatedly generating parent vertices to unify multiple subtrees. For subtrees with root vertices $c_1, c_2, \ldots, c_m$ respectively, a parent vertex is generating using the rule $p = H[c_1 \| c_2 \| \cdots \| c_m]$ where $H$ is a collision-resistant hash function. This process is repeated until all the vertices are in a single tree. Figure 1 shows a hash tree; each arrow indicates a hash dependency of a parent vertex on its (two) children.

Given the root vertex $r$ of a hash tree, we can verify the inclusion of a given leaf value $u$ by recomputing all hash tree vertices on the path from $u$ to $r$. To do this, we need the *off-path* vertices of $u$, i.e., all the siblings of every vertex on the path from $u$ to $r$. The off-path vertices of a vertex are highlighted in Figure 1. This suffices to perform all the hash computations to generate $r$.

In a hash tree, the root vertex acts as a concise commitment to the structure of the tree and the contents of all the tree vertices. It is computationally infeasible to find two distinct trees with the same root vertex values; the hardness of this problem is at least as great as finding a hash collision on the hash function (i.e., if an adversary has an efficient routine to find two hash trees with the same root vertex value, then they can use it as a subroutine in an efficient algorithm to find a hash collision on $H$). One implication of this is that if an adversary has no knowledge of a given leaf vertex value $x$, then it is hard for them to find a value which is the root vertex of a hash tree with $x$ as a leaf.

## 4 Problem Definition

We consider the broadcast authentication problem in fixed-topology networks. Consider a network consisting of a sender node $s$ and $n$ receiver/destination nodes $d_1, \ldots, d_n$. The network topology is represented by an undirected graph $G = (V, E)$ where $V = \{s, d_1, \ldots d_n\}$ and an edge connects any two nodes that can communicate (all communication links are bidirectional). All communication occurs along these edges, i.e., all messages are point-to-point between adjacent nodes in the graph. Each receiver knows the value $n$ and its particular index in the enumeration of the receivers (i.e., node $d_1$ knows that its index is 1, node $d_2$ knows that its index is 2, and so on). Likewise, the sender $s$ knows the topology $G$.

The sender $s$ shares a unique secret key $K_i$ with each receiver node $d_i$. The nodes have no capability for public key cryptography and are reliant on symmetric-key methods to provide authenticity and integrity.

In the broadcast authentication problem, the sender $s$ wishes to send a message $M$ to all receivers such that each receiver can check that the message is authentic, i.e., $M$ was truly sent by $s$. Some unknown fraction of the receivers may be malicious and may behave in arbitrary ways to subvert the protocol. The goal of the adversary is to cause some legitimate receiver to accept some forged $M'$ that was never sent by the sender $s$. We do not consider denial-of-service attacks (where a legitimate message $M$ that is sent by $s$ is rejected by a legitimate node due to the malicious actions of the adversary).

We consider two metrics in the design space for this problem. The first metric is communication congestion cost. Let $c(v)$ be the total amount of communications (transmission or reception) in the entire protocol performed by the node $v$. Then the congestion of the protocol is $\max_{v \in V} c(V)$, i.e., the greatest amount of communications performed by any single node.

The second metric is the latency of the protocol, defined as the the time between when the sender $s$ initiates the protocol (by sending the first message)

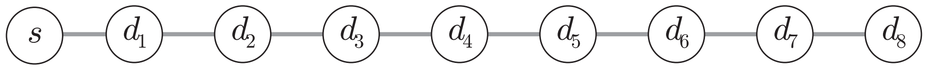


Figure 2: Linear Topology with 8 receivers

and the time when the last receiver node is able to authenticate the message. Since we are concerned only with correct authentication and not denial of service, we measure only *honest* latency, i.e., we assume that the adversary does not perform attacks specifically aimed at increasing the latency of the protocol by such actions as delaying messages or failing to respond as expected. To estimate latency, we divide time into *steps*. In each time step, every node is allowed to transmit OR receive (but not both) information from one other node. The amount of information exchanged in each time step is not limited by this definition; however to achieve a good communication congestion bound, a protocol typically has to send only a small amount of information in each time step.

## 5 Topologies Considered

We consider two topologies: linear and fully connected. In the linear topology, the sender lies at one end of a path of $n$ receivers. In other words, $G$ is a single path starting at $s$ and ending at $d_n$, in the sequence $(s, d_1, d_2, \ldots, d_n)$. In this topology, since the $n$th receiver is $n$ hops away from the sender, any protocol for disseminating $M$ must take at least $n$ time steps. Figure 2 shows an example of the linear topology for 8 receivers.

In the fully connected topology, $G$ is the complete graph of $n+1$ vertices (including the sender $s$ and the $n$ receivers). Figure 3 shows an example of the fully-connected topology for 5 receivers. In our definition of latency, we only allow each node to communicate with one other node in each time step. Consider the broadcast of a message $M$ from one node to all other nodes. An optimal broadcast schedule is as follows: in each time step, every node which has knowledge of $M$ communicates it to a different node that has not yet received $M$. Hence the number of nodes which have knowledge of $M$ doubles at each time step. Thus, any broadcast of a message $M$ to all nodes on the fully connected topology must take at least $\lceil \log_2(n+1) \rceil$ time steps. This algorithm is shown in Algorithm 1

## 6 Sending MACs for Each Receiver

The basic building block for message authentication is the *message authentication code* or MAC. The value $\mathrm{MAC}_K(M)$ is easily computed for any message $M$ by any node that holds the secret key $K$, but an adversary that does not know $K$ will find it computationally infeasible to guess $\mathrm{MAC}_K(M)$ even if given the values of $\mathrm{MAC}_K(M'_i)$ for a large number of different $M'_i \neq M$. Hence, if $K_i$ is a key shared only between the sender $s$ and a receiver $d_i$, then exhibit-

**Algorithm 1** Doubling Broadcast (for $n = 2^H$ nodes)

```
Require: Message $M$ for broadcast from $s$
1: Define $d_0 = s$
2: $s$ sends $M$ to $d_1$
3: for $h = 0$ to $H - 1$ do
4:     /* The next loop happens simultaneously for all i */
5:     for $i = 0$ to $2^h$ do
6:         $d_i$ sends $M$ to $d_{i+2^h}$
7:     end for
8: end for
```

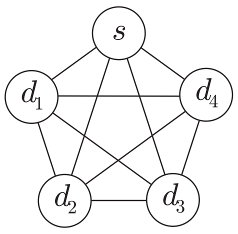


Figure 3: Fully-connected Topology with 5 receivers

ing $\text{MAC}_{K_i}(M)$ suffices to show to node $d_i$ that the sender $m$ originated $M$. Typically, an unrepeated nonce $N$ (such as a sequence number) is used to allow the same message $M$ to be sent more than once; the MAC construction in this case is $\text{MAC}_K(M\|N)$ (for simplicity, we assume that messages are all the same length - to ensure this, a hash function could be used).

The most straightforward approach to the broadcast authentication problem is for the sender to send the relevant $\text{MAC}_{K_i}(M\|N)$ individually to each receiver $d_i$. However, it has very poor communication congestion: the greatest communication occurs at $s$ since it must transmit $n$ cryptographic MACs.

## 7 Using a Hash Tree

Instead of sending a separate MAC for each receiver node, we can use a hash tree to batch these MACs. The basic idea was sketched in a prior publication for arbitrary and tree topologies [3]. We provide a more theoretically rigorous exposition of the general idea in this section.

The sender constructs a binary hash tree $T$ over $n$ leaf values which contain the message authentication codes for the message $M$. Each leaf in the hash tree is as follows, in order from left to right:

$$\text{MAC}_{K_1}(M\|N), \text{MAC}_{K_2}(M\|N), \ldots, \text{MAC}_{K_n}(M\|N)$$

We assume there is a canonical way to construct the tree $T$ and this method is known to all receivers; for example, $T$ could be constructed as the complete binary tree of $n$ nodes. Let $r$ be the root of $T$. Since each receiver $d_i$ knows $n$ and its own index $i$, it knows the position of its respective $\mathrm{MAC}_{K_i}(M\|N)$ in the hash tree, and thus knows the structure of the authentication path from $\mathrm{MAC}_{K_i}(M\|N)$ to $r$.

Define a *valid authentication path for $i$* as an authentication path in the complete binary hash tree of size $n$ from the $i$th leaf to the root, where the leaf value is $\mathrm{MAC}_{K_i}(M\|N)$ and the root vertex is $r$. Note that for an authentication path to be valid, it needs to have the correct shape as well as correct start and end values, e.g., the valid authentication path for node 1 is the path in $T$ consisting of $\lceil \log n \rceil$ left-children from $r$ down to $\mathrm{MAC}_{K_i}(M\|N)$.

We make two observations:

**The root vertex $r$ and the authentication path together act as an authenticator for $M, N$ to $d_i$.** Suppose a receiver $d_i$ accepts $M$ only if it receives $r$ and a valid authentication path for $i$ as defined above. We show that if the attacker does not know $K_i$, it is computationally hard for them to generate this data. Consider the task of the attacker attempting to guess any tuple $(M, N, r')$ and a valid authentication path $P'_i$. Let $r$ and $P_i$ be the canonically "correct" root vertex value and authentication path for $i$ that the sender would have computed using its full knowledge of all the keys. There are two cases: the attacker might guess correctly the value of $L_i = \mathrm{MAC}_{K_i}(M\|N)$. To do this, the attacker must break the forgery-resistance of the MAC, which is infeasible. In the second case, the attacker did not guess the correct value of the MAC, but instead guessed $T'_i \neq L_i$; however when the receiver plugged the correct $L_i \neq T'_i$ into the hash sequence represented by $P'_i$, it nonetheless resulted in the same root vertex $r'$. This implies that the adversary must engineer a hash collision on $H$, which is also infeasible.

**The receiver's MAC can be released to the (untrusted) network to facilitate the distributed derivation of authentication paths.** We can show that the following protocol between sender $s$ and receivers $d_i$ is secure:

1. $s$ constructs complete binary hash tree $T$ over the MACs; derives root vertex $r$.
2. $s \rightarrow d_i :\ (M, N, r)$
3. $d_i$ checks that $N$ has not been seen before, otherwise abort.
4. $d_i \rightarrow s :\ \mathrm{MAC}_{K_i}(M\|N)$
5. $s \rightarrow d_i :$ valid authentication path $P_i$ for $i$
6. $d_i$ checks that $P_i$ is a valid authentication path for $i$. If so, accept $M$.

The main idea is that the receiver $d_i$ can reveal its value of the MAC using its key for $M, N$ as soon as it receives $(M, N, r)$. We can show that even with the knowledge of $\mathrm{MAC}_{K_i}(M\|N)$ in the middle of the protocol, this does not help the adversary produce a valid authentication path since it occurs "after the fact", i.e., after the adversary has submitted its guess for $r$. Specifically, consider an adversary that submits a chosen tuple $(M', N', r')$ to a legitimate receiver $d_i$. If $N'$ is a valid nonce (i.e., $d_i$ has never before seen it), then $d_i$ releases to the adversary the value of $L_i = \mathrm{MAC}_{K_i}(M' \| N')$. The challenge now for the adversary is to derive some valid authentication path $P_i$ which works to take this newly discovered $L_i$ to the previously chosen value $r'$.

If the MAC is forgery-resistant, this will be the first time the attacker learns $L_i$ (i.e., it could not have chosen $r'$ with significant pre-knowledge of $L_i$). Furthermore, this property holds between attempts of the adversary against the protocol, i.e., the forgery-resistance of the MAC ensures that the use of a new nonce on each run of the protocol presents a (mostly) completely identical random game to the attacker in each run of the protocol; the amount of information gained about future outputs of the MAC in each round is negligible. In the following proof sketch, we only consider attackers who have a non-negligible chance of success in the very first round.

We can show that an attacker that succeeds in the first round is in fact infeasible if the hash function is collision-resistant. Let $A$ be the (deterministic) adversary function that guesses some $(M', N', r')$ and then efficiently and with non-negligible probability produces a valid authentication path $P_i$ when given $L_i = \mathrm{MAC}_{K_i}(M' \| N')$.

We can construct a reduction that produces hash collisions for $H$ using $A$. The structure of $P_i$ implies a fixed sequence of hashes which takes a sequence of inputs to the final value $r'$. Hence, if $A$ succeeds in finding valid authentication paths to the same $r'$ for two different values $T'_i$ and $L_i$ where $T'_i \neq L_i$, inspection of the two authentication paths will reveal a hash collision for $H$.

The reduction is as follows. First, run $A$ to obtain $M', N', r'$. Then select a random value for $K_i$ and feed $L_i = \mathrm{MAC}_{K_i}(M' \| N')$ to $A$. With non-negligible probability, $A$ will output a valid authentication path $P_i$ (if not, repeat this step). Now reset $A$ and run it again. Since it is deterministic, it will again output the same $M', N', r'$. We repeat the exact same process until $A$ succeeds in producing another authentication path $P'_i$ on another $K'_i \neq K_i$ s.t. $T'_i = \mathrm{MAC}_{K'_i}(M' \| N') \neq L_i$. The two distinct authentication paths $P_i$ and $P'_i$ terminate at the same value $r'$, implying a hash collision on $H$.

It should be noted that since the protocol involves letting a receiver release a MAC over an (potentially) arbitrary message, care should be taken that this functionality does not interact with other uses of the same key. For example, if messages between $s$ and $d_i$ are also authenticated using $K_i$, then an attacker could abuse the broadcast authentication protocol to forge valid messages between $s$ and $d_i$. Hence, we must ensure that either (1) the key used in the MAC should *only* be used for this protocol, or (2) all messages $M$ have a unique identifying header that is not shared by other types of messages.

Using our two observations it is clear that a family of broadcast authentication algorithms can take the following general form:

1. $s$ constructs complete binary hash tree $T$ over the MACs; derives root

vertex $r$.

2. $s$ disseminates $(M, N, r)$ to all receivers
3. Each receiver $d_i$ checks that $N$ has not been seen before, otherwise it aborts.
4. Each receiver $d_i$ releases $\mathrm{MAC}_{K_i}(M \| N)$ to the other receivers in the network.
5. The receivers collaborate to reconstruct valid authentication paths $P_i$ for each $i$
6. Each $d_i$ checks that $P_i$ is a valid authentication path for $i$. If so, accept $M$.

The only missing detail is how the authentication paths are reconstructed in step 5. We now address optimizations for this step for the different topologies.

### 7.1 Reconstructing Authentication Paths in the Linear Topology

Let $T$ be the complete binary hash tree. Let $F$ be a forest inside $T$, i.e. $F \subseteq T$. Let $C(F)$ be defined as the *completed merge* of $F$ in $T$ by the following process:

1. Find a vertex $v \in T - F$ that has two children that are both trees in $F$. If no such vertex exists, stop.
2. Add $v$ to $F$. Repeat step 1.

Essentially, we keep merging trees in $F$ as long as the resulting merged tree is also in $T$. Using this operation, we can describe a two-pass algorithm for reconstructing hash tree authentication paths in the linear topology. Intuitively, we pass a forest $F$ from $d_1$ to $d_2$, and so on to $d_n$, only communicating the root vertices of each tree in the forest. When each node receives the forest $F$, it adds its own MAC leaf value into the $F$ and computes the completed merge of the forest. It then forwards the root vertices of the trees in this larger forest onto the next node. Once the message front reaches $d_n$, an identical round of messages is performed in reverse, from $d_n$ to $d_{n-1}$ down to $d_1$. Algorithm 2 shows this algorithm; an example is shown in Figure 4.

Define the *left off path vertices* for a hash tree leaf $i$ as all sibling nodes that are to the left of the path from $i$ to the root of the hash tree $r$ (i.e., the set of all left-siblings of the nodes on the path); similarly the *right off path vertices* for a hash tree leaf $i$ are all sibling nodes that are to the right of the path from $i$ to the root of the hash tree $r$, (i.e., the set of all right-siblings of the nodes on the path).

**Lemma 1.** *For $i$ in $(2, \ldots, n)$, on the outgoing phase of Algorithm 2, each node $d_i$ receives all the left off-path vertices of the $i$th leaf vertex from node $d_{i-1}$.*

**Algorithm 2** Authentication Path Reconstruction in Linear Topology with $2n$ Latency

```
1:  $F \Leftarrow \phi$
2:  /* When $F$ is transmitted, only root vertices of each tree in $F$ are sent. */
3:  $s$ sends $(M, N, r), F$ to $d_1$
4:  /* Outgoing phase */
5:  for $i = 1$ to $n$ do
6:      Node $d_i$ receives $(M, N, r), F$ from Node $d_{i-1}$ (or $s$ in the case of $i = 1$)
7:      Node $d_i$ adds $L_i = \mathrm{MAC}_{K_i}(M \| N)$ to $F$
8:      $F \Leftarrow C(F)$
9:      if $i < n$ then
10:         Node $d_i$ sends $(M, N, r), F$ downstream to Node $d_{i+1}$
11:     end if
12: end for
13: /* Incoming phase */
14: $F \Leftarrow \phi$
15: for $i = n$ down to 1 do
16:     Node $d_i$ receives $F$ from Node $d_{i+1}$
17:     Node $d_i$ adds $L_i = \mathrm{MAC}_{K_i}(M \| N)$ to $F$
18:     $F \Leftarrow C(F)$
19:     if $i < n$ then
20:         Node $d_i$ sends $F$ upstream to Node $d_{i-1}$
21:     end if
22: end for
```

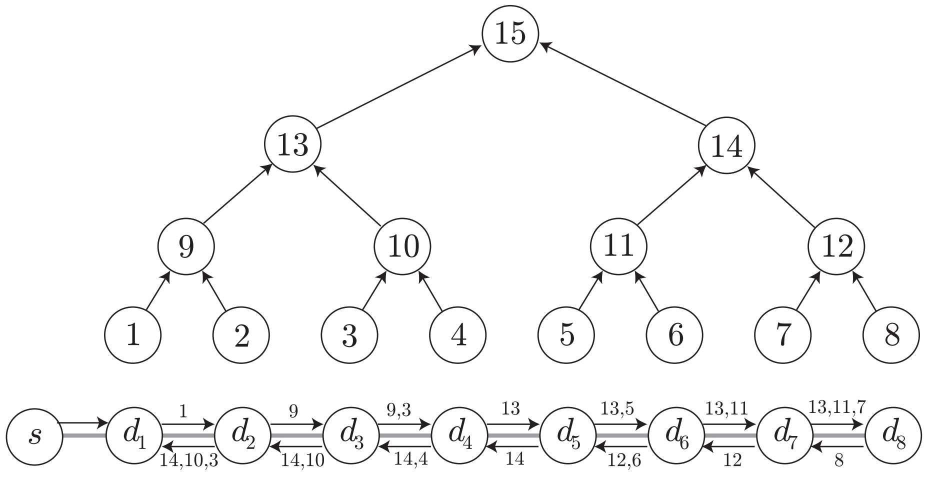


Figure 4: Example of Algorithm 1 with 8 nodes. Numbered circles represent hash tree vertices; vertices that are transmitted are shown on arrows between nodes.

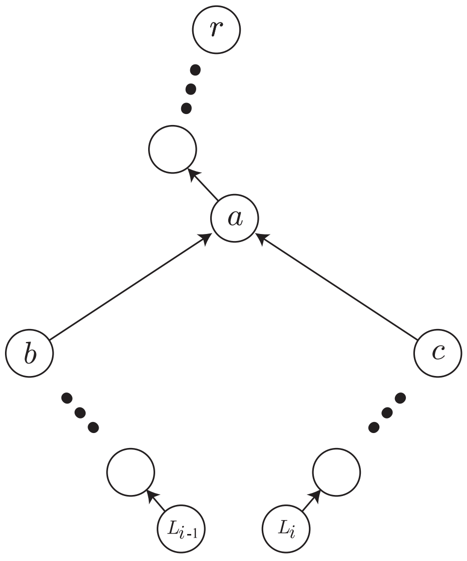
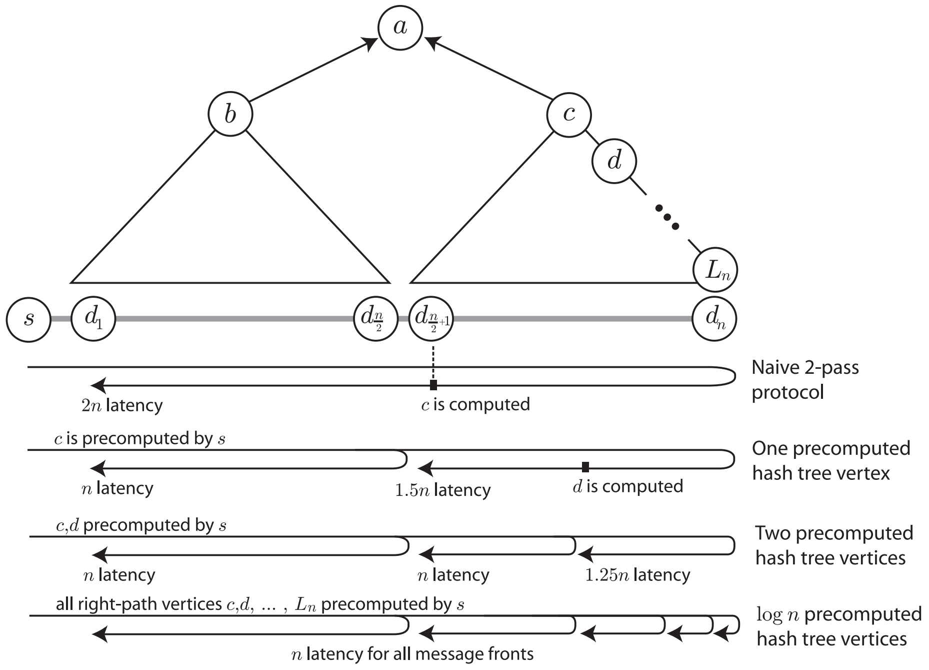


Figure 5: Proof idea for Lemma1

*Proof.* By induction on $i$. For $i = 1$, the lemma is true since $L_1$ is the leftmost vertex in the tree and has no left off-path vertices. Now suppose the lemma holds for $i - 1$. The vertex $L_i$ shares a lowest common ancestor $a$ with vertex $L_{i-1}$. Since $L_{i-1}$ and $L_i$ are adjacent in the tree, the path from $a$ to $L_{i-1}$ goes to a left child $b$, and then a series of 0 or more right-children down to $L_{i-1}$. Similarly, the path from $a$ to $L_i$ goes to a right child $c$, and then a series of 0 or more left-children down to $L_i$. This situation is illustrated in Figure 5. Since $d_{i-1}$ received all the left off-path vertices of $L_{i-1}$, it knows all the left-siblings (if any) of the nodes between $L_{i-1}$ and $b$. Hence when $d_{i-1}$ adds the vertex $L_{i-1}$ to the forest represented by the subtrees rooted at the left off-path vertices of $L_{i-1}$, and computes the completed merge of this forest, it will compute all the hash tree vertices from $L_{i-1}$ up to and including $b$. All the left off-path vertices above $a$ are untouched by the merge process because $d_{i-1}$ does not have enough information to compute $a$. The roots of the trees in the new merged forest $F$ are then sent to $d_i$. This is exactly the left off-path vertices of $L_i$, because $L_{i-1}$ and $L_i$ share the same set of left off-path vertices for the path from $a$ to the root, and there are no left siblings for any vertices between $L_i$ and $c$ inclusive. The only new left sibling for $L_i$ is the vertex $b$ which is the root of the new merged tree computed when $d_{i-1}$ added and merged $L_{i-1}$. Hence the lemma holds. □

**Lemma 2.** *For $i$ in $(n - 1, \dots, 1)$, on the incoming phase of Algorithm 2, each node $d_i$ receives all the right off-path vertices of the ith leaf vertex from node $d_{i+1}$.*



$a$
$b$
$c$
$d$
$L_n$
$s$
$d_1$
$d_{\frac{n}{2}}$
$d_{\frac{n}{2}+1}$
$d_n$
Naive 2-pass protocol
$2n$ latency
$c$ is computed
$c$ is precomputed by $s$
One precomputed hash tree vertex
$n$ latency
$1.5n$ latency
$d$ is computed
$c$,$d$ precomputed by $s$
Two precomputed hash tree vertices
$n$ latency
$n$ latency
$1.25n$ latency
all right-path vertices $c$,$d$, ... , $L_n$ precomputed by $s$
$\log n$ precomputed hash tree vertices
$n$ latency for all message fronts

Figure 6: Latency optimization by sending precomputed hash tree vertices at $s$. For simplicity, assume $n$ is a power of 2 (i.e., the hash tree is a perfect binary tree).

The proof of Lemma 2 is identical to the proof of Lemma 1. Together the two lemmas imply that each node receives its full set of off-path vertices by the end of the protocol and can each compute their respective authentication paths.

The communication congestion of this protocol is at most $\lceil logn \rceil + 1$ hash tree vertices. The worst-case number of transmissions occurs in the two inner vertices of every subtree of four leaves. The latency of the protocol is exactly $2n$ time steps since the protocol proceeds in two complete passes of the linear topology and the second pass can only start after the first pass is completed.

### 7.1.1 Latency Optimization

Figure 6 illustrates a method for speeding up the latency of the basic 2-pass protocol. For simplicity, assume $n$ is a power of 2 (i.e., the hash tree is a perfect binary tree). We describe how to deal with arbitrary $n$ later. Consider the latency of the original 2-pass protocol. The last node to verify $M$ is $d_1$, because it has to wait for the completion of the second pass. Specifically, $d_1$ is dependent on the hash tree vertex $c$ which is computed at $d_{n/2+1}$. In order to speed up the process, the sender $s$ could send $c$ directly to all the nodes in the subtree rooted at $b$, so that these nodes do not have to wait for the node $d_{n/2+1}$ to

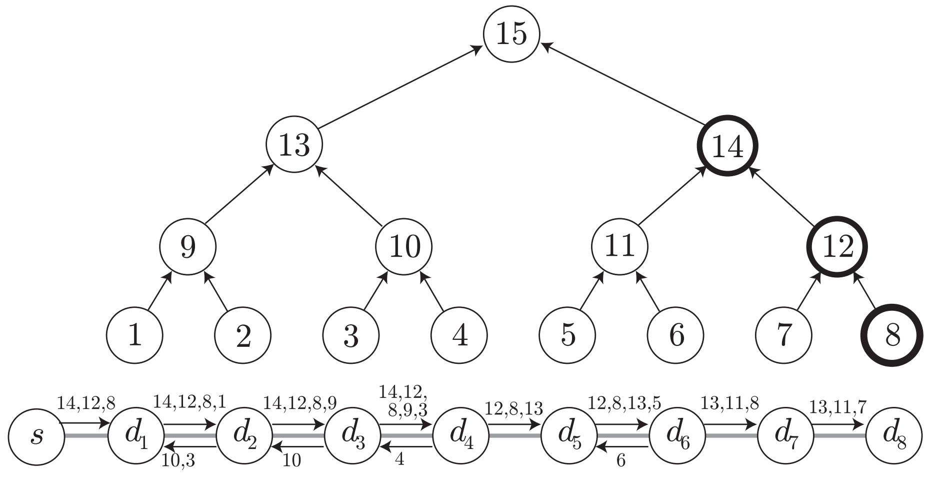


Figure 7: Example of latency-optimized hash tree authentication path reconstruction for 8 receivers in a linear topology. Not shown: the original tuple $(M, N, r)$ is carried on each outward-bound message from $d_i$ to $d_{i+1}$.

compute $c$. This allows the return pass for computing the subtree rooted at $b$ to "start early" when the outgoing pass reaches the node $d_{n/2}$. This results in two concurrent message fronts as indicated in the Figure 6. The latency for $d_1$ is now only exactly $n$ time steps: the message front needs to travel down to $d_{n/2}$ ($n/2$ steps); $d_{n/2}$ spends 1 extra step sending it on to $d_{n/2+1}$; and then on the next step $d_{n/2}$ immediately starts the returning message front back to $d_1$ ($n/2 - 1$ steps) for a total of $n$. The latency for $d_{n/2} + 1$ remains at $1.5n$ since the message front must travel down to $d_n$ and then back to $d_{n/2} + 1$. The worst case latency has thus improved to $1.5n$ from $2n$.

We can repeat this process to improve the latency at $d_{n/2} + 1$ by having the sender also include the hash tree vertex for the right child of $c$ as shown in Figure 6. This allows another message front to start early at $d_{3n/4}$ allowing $d_{n/2+1}$ to also perform verification after around $n$ time steps. The worst case is now node $d_{3n/4+1}$ which has a latency of $1.25n$. Taking this to the limit, we can have the sender $s$ send all $\lceil \log n \rceil$ vertices from $c$ down to the rightmost leaf vertex $L_n$. In other words, the sender sends the entire right-path from the root of the hash tree down to the right-most leaf vertex. This allows all nodes to perform verification in $n$ time steps. An example of the case for 8 nodes is shown in Figure 7.

To guarantee a latency bound of $n$ for values of $n$ that are not powers of 2, the structure of the hash tree must be selected such every vertex on the path between the root and the right-most leaf has two subtrees with equal numbers of nodes. A simple rule that guarantees the $n$ latency bound is to subdivide subtrees into equal halves, where if there is an odd number of nodes in the subtree, the right subtree gets one more node than the left subtree. For example, to subdivide 13 nodes into two subtrees, the left subtree will have 6 nodes and the right subtree will have 7. The individual subtrees are then subdivided in this way recursively. Since all nodes have knowledge of $n$ and their own index $i$, each node can identify its own position in this canonical tree construction. It is straightforward to see that this will result in a tree of height $\lceil \log n \rceil$ and will always ensure a worst-case latency of $n$ time steps to verification for any node, for any $n$.

This optimization adds an additional communication overhead of $\lceil \log n \rceil$ hash values. Of these extra hash values, the first one (i.e., node $c$ in Figure 6 does not add additional overhead since it is part of the authentication path for all nodes that forward it. Hence, with the base $\lceil \log n \rceil + 1$ overhead of the original scheme, this results in a communication congestion overhead of $2\lceil \log n \rceil$ hash values in total.

## 7.2 Authentication Paths in the Fully-connected Topology

We first describe an algorithm for the case where $n$ is a power of 2, i.e., the hash tree is a perfect binary hash tree.

The protocol proceeds in two phases. In the first phase, the message tuple $(M, N, r)$ is disseminated using the repeated doubling broadcast method of Algorithm 1. In the second phase, nodes collaborate to reconstruct authentication paths. The schedule of messages is illustrated in Figure 8. There are two time-steps of message exchanges for each level of the hash tree, starting from the leaves. At each level $h$, every node exchanges the hash tree vertex at level $h$ on its authentication path with that of its counterpart in the neighboring subtree at level $h$ (this takes two time steps). This allows it to compute the next higher hash tree vertex (at level $h - 1$) on its authentication path and thus repeat the process in the next pair of time steps. Within $2 \log n$ time steps all nodes will have computed their authentication paths to the root. This process is shown in Algorithm 3.

It remains to address the case where $n$ is not a power of 2. We can adapt the algorithm as follows: let the hash tree $T$ be a complete binary hash tree of height $H$. Let $T'$ be the perfect binary hash tree of height $H - 1$ resulting from deleting all the leaves of $T$ that are at maximum depth $H$. Clearly, we can run Algorithm 3 on $T'$. To do this, we let each internal vertex of $T$ at depth $H - 1$ be represented by one of the receiver nodes responsible for one of the children at depth $H$. Specifically, at the start of the protocol, for each even $i$ that has a leaf vertex at the maximum depth $H$, let $d_i$ communicate its leaf $\mathrm{MAC}_{K_i}(M \| N)$ to $d_{i-1}$, which then computes the internal vertex $v$ that is the parent of both leaf vertices of $d_i$ and $d_{i-1}$. We can then run Algorithm 3 on $T'$ using $d_{i-1}$ as the node responsible for the leaf $v$ in $T'$. This allows $d_{i-1}$ to compute the authentication path for $v$ (and thus the authentication path for its own leaf in $T$). If this is successful, $d_{i-1}$ can then send the correct authentication path to $d_i$ allowing $d_i$ to also perform authentication. the entire process adds 2 extra steps in addition to the $2\lfloor \log n \rfloor$ steps for executing Algorithm 3 on $T'$. Since these extra 2 steps are only needed when $n$ is not a power of 2, the latency for

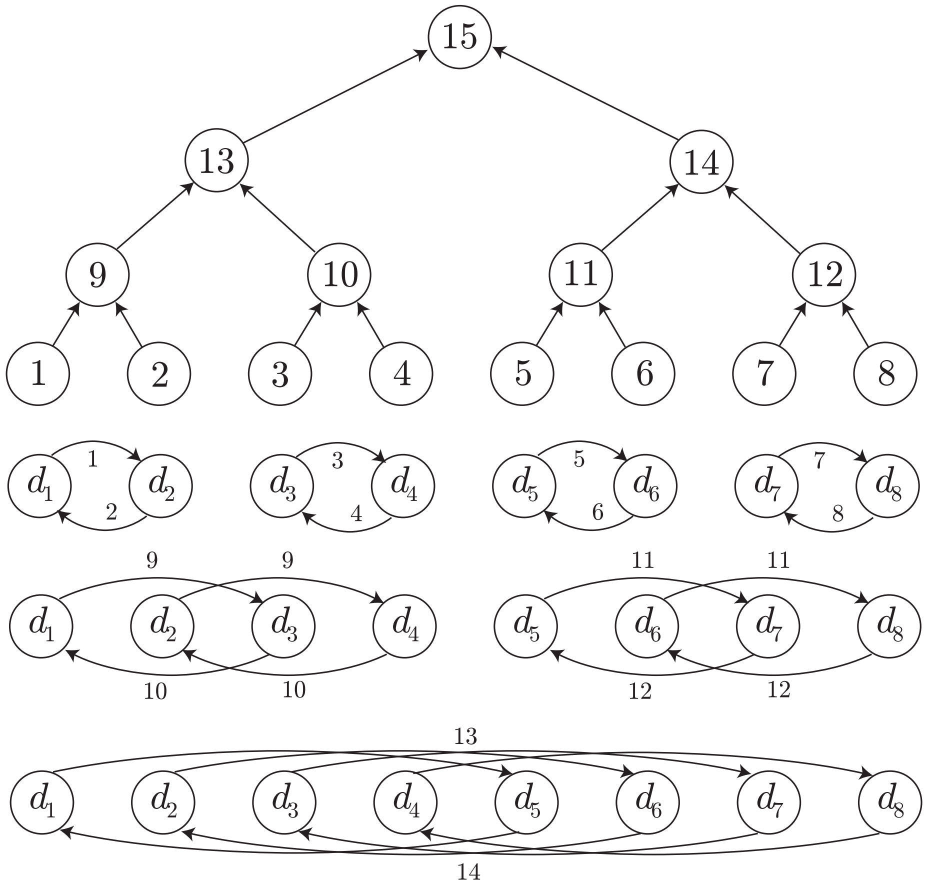


Figure 8: Hash Tree Authentication Path Reconstruction for Fully Connected Topology

**Algorithm 3** Authentication Path Reconstruction in Fully Connected Topology

```
1: $s$ sends $(M, N, r)$ to all receivers using Algorithm 1
2: If any receiver has seen $N$ used as a nonce before, abort.
3: for $i = 1$ to $n$ do
4:     Initialize $v_i[0] \leftarrow \mathrm{MAC}_{K_i}(M \| N)$
5: end for
6: for $h = 0$ to $(\log n) - 1$ do
7:     /* The loop below happens in two time steps for all i */
8:     for $i = 1$ to $n$ do
9:         /* $d_j$ is $d_i$'s counterpart in the other subtree; the -1 and +1 corrects for
           the fact that the receivers start their numbering at 1 instead of 0 */
10:        $j \leftarrow ((i-1) \oplus 2^h) + 1$
11:        $d_i \rightarrow d_j : \ v_i[h]$
12:        $d_j \rightarrow d_i : \ v_j[h]$
13:        if $i < j$ then
14:            $v_i[h+1] \leftarrow H[v_i \| v_j]$
15:        else
16:            $v_i[h+1] \leftarrow H[v_j \| v_i]$
17:        end if
18:    end for
19: end for
20: /* Authentication path for $d_i$ is now in $v_i[0..\log n - 1]$ */
21: Each $d_i$ checks that $\mathrm{MAC}_{K_i}(M \| N)$ is the $i$th leaf in a hash tree rooted at $r$
22: If so, $d_i$ accepts $M$ as authentic. Otherwise $M$ is rejected.
```

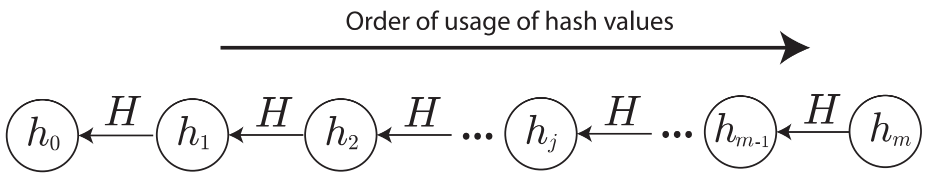


Figure 9: Hash chain

hash tree authentication path reconstruction is at most $2\lceil \log n \rceil + 1$ in general. Coupled with the $\lceil \log n \rceil$ overhead of the initial broadcast dissemination, the total latency is $3\lceil \log n \rceil + 1$.

The communication congestion overhead for this scheme is at most $5\lceil \log n \rceil$ values: $3\lceil \log n \rceil$ for the dissemination of the tuple $(M, N, r)$, and $\lceil \log n \rceil$ for the authentication path and an additional $\lceil \log n \rceil$ for the extra overhead of the cases where $n$ is not a power of 2.

## 8 Using Hash-Chains

Perrig et al. described the use of hash chains in broadcast authentication in the TESLA authentication scheme [8]. A sketch of TESLA follows. A hash chain of some predetermined length $m$ is generated by randomly selecting a secret seed value $h_m$ and then iterating a pre-image resistant hash function $H$ to generate a chain of $m$ values such that $h_{i-1} = H[h_i]$ for $i = n-1, n-2, \ldots, 1$. Figure 9 shows this hash chain. Receivers are then preloaded with $h_0$ and the hash chain values are then used in order of increasing index (i.e., first $h_1$ is used, then $h_2$, then $h_3$, and so on). This chain of $m$ values allows for the authentication of up to $m$ messages at fixed intervals. All nodes are loosely time-synchronized, and we can divide time into fixed-length epochs. Within each time epoch, we can use one hash chain value to authenticate one message as follows. For the message in the $j$th time epoch, the sender computes $\mathrm{MAC}_{h_j}(M_j)$ as the authenticator for message $M_j$. This MAC is sent to all receivers along with $M_j$. Receivers then record the MAC and message, and note the epoch in which it was received. The sender then waits for a time period sufficient to ensure that all receivers have received $M_j$ (at least one epoch), and then releases $h_j$ as the hash chain value for epoch $j$. The receivers can now verify that $H[h_j] = h_{j-1}$ and check the authentication code $\mathrm{MAC}_{h_j}(M_j)$ and discard any messages from epoch $j$ which did not have the valid MAC. The security of the scheme follows from the fact that the key $h_j$ to the MAC in epoch $j$ is released only after epoch $j$ has passed; this means that the adversary is unable to use this key $h_j$ to forge any acceptable MACs because the current epoch is at least $j+1$.

## 8.1 Hash-Chain-Keyed Authentication Codes

We describe a new construction that is similar to TESLA but removes the time synchronization requirement by borrowing some ideas from the hash tree authentication construction. We first construct a hash chain $h_0, \ldots, h_m$ in a similar way, such that $h_{i-1} = H[h_i]$. In this case, however, we do not require any kind of time synchronization and are free to use each hash chain value on an as-needed basis: define each use of a given hash chain value as a message *stage* (hence there can be up to $m$ stages, after which the protocol must be re-initialized). For now, assume that all nodes have received the previous value $h_{j-1}$ on the previous stage and are waiting for the $j$th stage message $M_j$ (we show how to relax this assumption later). The sender computes $T_j = \text{MAC}_{h_j}(M_j \| N_j)$ as the authenticator (where $N_j$ is a non-repeated nonce that includes an unambiguous encoding of the value of the stage $j$; for example, we can let $N_j = j$ in a fixed length encoding). The sender then disseminates $(M_j, N_j, T_j)$ to the network. Note that $M_j, N_j, T_j$ should each have a fixed length. Once each receiver $d_i$ has received the message tuple, it extracts the value of $j$ from $N_j$, and checks that this is the first time it has seen a tuple for the $j$th stage, and if so, $d_i$ releases $\text{MAC}_{K_i}(M_j \| N_j \| T_j)$ to the network. Note that each receiver only ever releases one MAC for each stage. The nodes in the network then collaborate to compute:

$$A_j = \bigoplus_{i=1}^{n} \text{MAC}_{K_i}(M_j \| N_j \| T_j)$$

The network then forwards $A_j$ to the sender. The sender can verify that the result was indeed composed of the XOR of all the respective MACs of each receiver (since it has all the keys). At this point the sender may release the hash chain value $h_j$. Receivers verify that $H[h_j] = h_{j-1}$ and that $T_j = \text{MAC}_{h_j}(M_j \| N_j)$. If so, then they accept $M_j$. The algorithm for the linear topology is shown in pseudo code in Algorithm 4.

The security of the algorithm follows from an identical argument to TESLA. The hash chain value $h_j$ is released only after all nodes have received a message containing $N_j$. Since nodes only respond to one message per value of $j$, the release of $h_j$ does not allow the adversary to create any MACs to messages that might be accepted by the receivers. In other words, once the adversary has obtained $h_j$, it is able to forge messages claiming to be the $j$th message, but any such message will always be ignored by all receivers because they have already seen a message claiming to be the $j$th one.

Stage consistency is important in this protocol. All stages prior to and including the last successful stage are insecure since their hash chain keys could potentially have been released by the sender. Fortunately, nodes have control over whether or not a stage is successful (if they did not release their MAC for a stage, then it is infeasible for an adversary to forge success for that stage). Let $j_i$ be the latest stage in which which $d_i$ released its MAC. We can be sure that no stages *after* $j_i$ could have been successful. Hence, node $d_i$ only responds to

**Algorithm 4** Hash-Chain-Keyed Authentication Code I for linear topology

```
Require: Hash chain $h_0, \dots, h_m$ with all receivers having $h_0$ preloaded.
 1: $M_j = j$th Message ; $N_j = j$th Nonce
 2: $s$ computes: $T_j = \mathrm{MAC}_{h_j}(M_j \| N_j)$
 3: /* Broadcast Dissemination */
 4: $s$ sends $(M_j, N_j, T_j)$ to the network:
 5: for $i = 1$ to $n - 1$ do
 6:     If $d_i$ inspects $N_j$ to extract $j$. If $d_i$ has seen any messages for stage $j$ prior
        to this, or if $j$ is less than or equal to the latest stage in which $d_i$ released
        a MAC, abort.
 7:     $d_i$ forwards $(M_j, N_j, T_j)$ to $d_{i+1}$
 8: end for
 9: /* Acknowledgment Aggregation */
10: $d_n$ computes: $A_j \leftarrow \mathrm{MAC}_{K_n}(M_j \| N_j \| T_j)$
11: $d_n$ sends $A_j$ to $d_{n-1}$
12: for $i = n - 1$ down to 1 do
13:     $d_i$ computes: $A_j \leftarrow A_j \oplus \mathrm{MAC}_{K_i}(M_j \| N_j \| T_j)$
14:     $d_i$ sends $A_j$ to $d_{i-1}$ (or to $s$ if $i = 1$)
15: end for
16: /* Dissemination of hash chain value $h_j$ */
17: $s$ checks: $A_j = \bigoplus_{i=1}^{n} \mathrm{MAC}_{K_i}(M_j \| N_j \| T_j)$
18: If check succeeds, disseminate $h_j$ to the network by sending it to $d_1$
19: for $i = 1$ to $n$ do
20:     $d_i$ checks that $h_j$ is exactly the $j$th value on the hash chain by checking
        that $H[h_j] = h_{j-1}$
21:     If $d_i$ does not have $h_{j-1}$, it can iterate the hash function to see if $h_j$
        generates the latest known hash chain value in the appropriate number
        of applications of $H$.
22:     $d_i$ checks $T_j = \mathrm{MAC}_{h_j}(M_j \| N_j)$; if success, accept $M_j$.
23:     $d_i$ forwards $h_j$ to $d_{i+1}$
24: end for
```

messages that appear to be from a stage strictly after $j_i$. This ensures that the protocol is secure; in fact it functions correctly even if receivers all have different $j_i$ and have different latest-known-values on the hash chain.

Ensuring stage consistency might allow an adversary to launch a persistent denial of service attack by disseminating a message tuple claiming to be from the latest possible stage (e.g. stage $m$). Even though this message tuple will be correctly rejected by all receivers, the receivers will now stop responding to all legitimate messages (since they appear to be from stages prior to $m$). A method to prevent this attack is to let $N_j$ also be derived from a hash chain such that $H[N_j] = N_{j-1}$. All receivers are additionally required to check the validity of this hash chain relationship when receiving a message tuple. Since the attacker is computationally unable to invert the hash function, it cannot derive message tuples claiming to be from a stage later than the most current stage at the sender.

If the protocol is performed on a receiver group with changing membership, it is important that new members be correctly initialized with the correct $j$ such that they do not erroneously accept broadcast messages secured by MACs for which the hash chain keys have already been exposed.

Since this protocol also involves the voluntary release of MACs computed over arbitrary messages, the usual caveats regard key-use apply, e.g. the keys used to compute the MACs in this protocol should not be re-used for other functions.

This description of the hash-chain-keyed authentication code protocol requires 3 passes between the sender and the network and thus has a latency bound of $3n$ for the linear topology. In a general, for any topology, the unoptimized protocol requires $3d$ time steps where $d$ is the greatest hop distance of any node to the sender. In the next section we show how to improve this.

## 8.2 Optimization for latency

In the original 3-pass protocol, the third pass where $h_j$ is disseminated depended on the reception of the aggregated MAC $A_j$ by the sender. This ensures that all receivers have received the message tuple for stage $j$, and will not accept any more (potentially forged) messages in that stage. This makes it "safe" for $h_j$ to be released. However, passing the aggregated MAC $A_j$ all the way back to the sender, and then passing $h_j$ out to all receivers, involves a round-trip to the sender and is thus time-consuming. We can optimize this two-step process by having the sender include an encrypted version of $h_j$ in the outgoing message such that only a successful computation of $A_j$ can reveal it. Specifically, the sender computes $A_j = \bigoplus_{i=1}^{n} \mathrm{MAC}_{K_i}(M_j \| N_j \| T_j)$ and includes $E_j = E_{A_j}(h_j)$ in the broadcast, where $E_K(PT)$ indicates the encryption of string $PT$ with key $K$. In this way, any node that successfully computes $A_j$ can immediately decrypt $E_j$ to receive $h_j$. This avoids the need for a third pass where $h_j$ is disseminated in the network.

In general $E$ can be any encryption function (e.g., AES), but since $A_j$ is used as a key to encrypt only one (short) plaintext $h_j$, if we set the MAC length and the hash length to be equal, we can simply use $A_j$ as a pseudorandom masking factor for $h_j$ as follows:

$$E_j = \left[\bigoplus_{i=1}^{n} \mathrm{MAC}_{K_i}(M_j \| N_j \| T_j)\right] \oplus h_j$$

This simplifies the security analysis since it removes the adversary's advantage in attacking the encryption function. From this construction, it is clear that if even one MAC is unknown, the adversary finds it computationally difficult to determine $h_j$ from $E_j$. Hence, the adversary can only feasibly recover $h_j$ after all legitimate nodes have released their MACs for stage $j$. At this point $h_j$ is useless for forging messages for stage $j$ since all legitimate nodes will ignore any messages claiming to be from stage $j$. Hence the security of the scheme holds. The modified protocol is described in pseudocode in Algorithm 5.

**Algorithm 5** Hash-Chain-Keyed Authentication Code II for linear topology

```
Require: Hash chain $h_0, \ldots, h_m$ with all receivers having $h_0$ preloaded.
 1: $M_j = j$th Message ; $N_j = j$th Nonce
 2: $s$ computes: $T_j = \mathrm{MAC}_{h_j}(M_j \| N_j)$
 3: $s$ computes: $E_j = \bigoplus_{i=1}^{n} \mathrm{MAC}_{K_i}(M_j \| N_j \| T_j) \oplus h_j$
 4: $A_j \leftarrow 0$
 5: /* Broadcast Dissemination */
 6: $s$ sends $(M_j, N_j, T_j, E_j, A_j)$ to $d_1$:
 7: for $i = 1$ to $n - 1$ do
 8:     If $d_i$ inspects $N_j$ to extract $j$. If $d_i$ has seen any messages for stage $j$ prior
        to this, or if $j$ is less than or equal to the latest stage in which $d_i$ released
        a MAC, abort.
 9:     $d_i$ computes: $A_j \leftarrow A_j \oplus \mathrm{MAC}_{K_i}(M_j \| N_j \| T_j)$
10:     $d_i$ forwards $(M_j, N_j, T_j, E_j, A_j)$ to $d_{i+1}$
11: end for
12: /* Decryption of $h_j$ */
13: $d_n$ computes: $A_j \leftarrow A_j \oplus \mathrm{MAC}_{K_n}(M_j \| N_j \| T_j)$
14: $d_n$ computes: $h_j = A_j \oplus E_j$
15: $d_n$ checks that $h_j$ is exactly the $j$th value on the hash chain by checking
    that $H[h_j] = h_{j-1}$
16: If $d_n$ does not have $h_{j-1}$, it can iterate the hash function to see if $h_j$ generates
    the latest known hash chain value in the appropriate number of applications
    of $H$.
17: $d_n$ checks $T_j = \mathrm{MAC}_{h_j}(M_j \| N_j)$; if success, accept $M_j$.
18: $d_n$ forwards $h_j$ to $d_{n-1}$
19: for $i = n - 1$ downto 1 do
20:     $d_i$ checks $T_j = \mathrm{MAC}_{h_j}(M_j \| N_j)$; if success, accept $M_j$.
21:     $d_i$ forwards $h_j$ to $d_{i-1}$
22: end for
```

The resultant protocol terminates in 2 passes and thus results in a latency of $2n$ for a linear topology. The communication overhead is unchanged, i.e., 6 values are passed around by each receiver in the network.

## 8.3 Hash-chain-keyed Broadcast Authentication for Fully Connected Topology

We show two versions of hash-chain-keyed broadcast authentication for the fully connected topology. In the first version, we optimize for latency; in the second version, we optimize for communication congestion.

### 8.3.1 Latency-optimized Hash-Chain-Key

The Hash-tree reconstruction authentication algorithms of Section 7 are conceptually related to the (optimized) hash-chain-keyed broadcast authentication algorithms because both classes of algorithms involve a first phase where data from the sender is disseminated to the network, followed by a second phase involving an all-to-all dissemination of $MAC$ information between receivers. In other words, we can transform any algorithm of the first type into an algorithm of the second type with equivalent latency by replacing all hash function evaluations of the form $g(v_1, v_2) = H[v_1\ Vert v_2]$ with $f(v_1, v_2) = v_1 \oplus v_2$.

Applying this intuition to the algorithm described for the fully-connected topology in Section 7.2, we can implement the hash-chain-keyed method by using Algorithm 1 to first disseminate $(M_j, N_j, T_j, E_j)$. Subsequently, we use the algorithm described in Figure 8 to exchange MAC information between receivers. Wherever we compute a hash over two values to get an internal node in the hash tree in the original algorithm for Section 7.2, we now instead just compute the XOR over the two values. At the conclusion of the algorithm, all receivers will have enough information to compute $A_j = \bigoplus_{i=1}^{n} \mathrm{MAC}_{K_i}(M_j \| N_j \| T_j)$. Each node then individually uses this value to decrypt $E_j$ and retrieve $h_j$, and use $h_j$ to verify the MAC $T_j$. The algorithm is described in pseudocode in Algorithm 6. The case where $n$ is not a power of 2 is handled in an identical manner to Section 7.2.

Since the protocol takes steps that are very similar to the algorithm of Section 7.2, it has an identical latency, i.e., $3\lceil \log n \rceil + 1$. The congestion overhead is also similar:

The communication congestion overhead for this scheme is at most $5\lceil \log n \rceil + 2$. We need $4\lceil \log n \rceil$ overhead for the dissemination of the broadcast tuple $(M_j, N_j, T_j, E_j)$; one message overhead for each of the $\lceil \log n \rceil$ rounds needed for the reconstruction of $A_j$ and an additional two messages for the extra overhead of the cases where $n$ is not a power of 2.

This protocol has almost identical latency and congestion compared with the hash-tree based scheme of Section 7.2. We describe another protocol which achieves another trade off point in the next section.

**Algorithm 6** Hash-Chain-Keyed Authentication Code II for Fully Connected Topology

```
Require: Hash chain $h_0, \ldots, h_m$ with all receivers having $h_0$ preloaded.
Require: $n$ is a power of 2
 1: $M_j = j$th Message ; $N_j = j$th Nonce
 2: $s$ computes: $T_j = \mathrm{MAC}_{h_j}(M_j \| N_j)$
 3: $s$ computes: $E_j = \bigoplus_{i=1}^{n} \mathrm{MAC}_{K_i}(M_j \| N_j \| T_j) \oplus h_j$
 4: /* Broadcast Dissemination */
 5: $s$ sends $(M_j, N_j, T_j, E_j)$ to all receivers using Algorithm 1
 6: Each receiver $d_i$ inspects $N_j$ to extract $j$. If $d_i$ has seen any messages for
    stage $j$ prior to this, or if $j$ is less than or equal to the latest stage in which
    $d_i$ released a MAC, abort.
 7: for $i = 1$ to $n$ do
 8:     Initialize $A_{j,i} \leftarrow \mathrm{MAC}_{K_i}(M_j \| N_j \| T_j)$
 9: end for
10: for $h = 0$ to $(\log n) - 1$ do
11:     /* The loop below happens in two time steps for all i */
12:     for $i = 1$ to $n$ do
13:         /* $d_k$ is $d_i$'s counterpart in the other subtree; the -1 and +1 corrects
            for the fact that the receivers start their numbering at 1 instead of 0 */
14:         $k \leftarrow ((i-1) \oplus 2^h) + 1$
15:         $d_i \rightarrow d_k : \; A_{j,i}$
16:         $d_k \rightarrow d_i : \; A_{j,k}$
17:         $A_{j,i} \leftarrow A_{j,i} \oplus A_{j_k}$
18:         $A_{j,k} \leftarrow A_{j,i} \oplus A_{j_k}$
19:     end for
20: end for
21: /* Now $A_{j,i} = \bigoplus_{i=1}^{n} MAC_{K_i}(M_j \| N_j \| T_j)$ for all i */
22: /* Decryption of $h_j$: happens for all i simultaneously */
23: $d_i$ computes: $h_j = A_j \oplus E_j$
24: $d_i$ checks that $h_j$ is exactly the $j$th value on the hash chain by checking that
    $H[h_j] = h_{j-1}$
25: If $d_i$ does not have $h_{j-1}$, it can iterate the hash function to see if $h_j$ generates
    the latest known hash chain value in the appropriate number of applications
    of $H$.
26: $d_i$ checks $T_j = \mathrm{MAC}_{h_j}(M_j \| N_j)$; if success, accept $M_j$.
```

### 8.3.2 Congestion-optimized Hash-Chain-Key

The algorithm of Section 8.3.1 has $\theta \log n$ communication congestion. However, one of the main advantages of the hash-chain-key method over the hash-tree method is that authentication is based on a single value $h_j$ rather than a $\log n$ size path in a hash tree. We show how to use this property to achieve constant communication congestion with only a mild dilation in latency.

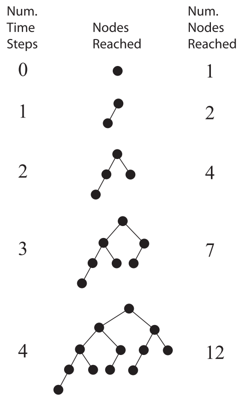
First, we consider a schedule for broadcast on a subgraph of $K_n$ that is a binary tree. Consider the following broadcast algorithm: when each node first receives the message to be disseminated, it only forwards the message to two other nodes (who each have not yet heard the message at the time of the forwarding). The message dissemination process can be visualized as occurring in a binary tree, where in each time step, each node that just received the message in the previous time step now forwards it to its left child; and every node that forwarded a message to its left child in the previous time step now forwards the message to its right child in the binary tree. Hence, each node is only transmitting in two time steps in the protocol, first to the left then to the right. The algorithm is summarized in Algorithm 7.

**Algorithm 7** Fibonacci-tree Broadcast

```
Require: Message $M$ for broadcast from $s$
1: Let $d_0 = s$
2: $sent_i = 0$ for all $i$
3: while some nodes have not yet received $M$ do
4:    for all $d_i$ that have received $M$, s.t. $sent_i < 2$ do
5:       $d_i$ sends $M$ to some $d_j$ that has not received $M$
6:       $sent_i \leftarrow sent_i + 1$
7:    end for
8: end while
```

The message dissemination process of Algorithm 7 is illustrated in Figure 8.3.2. The message dissemination pattern is a well-studied structure called a *Fibonacci Tree*, recursively defined as follows: a Fibonacci Tree of height $n$ has a Fibonacci Tree of height $n-1$ as a left child and a Fibonacci Tree of height $n-2$ as the right child; the Fibonacci trees of height 0 and height 1 are the trees with 1 and 2 vertices respectively. It is clear that sequence of sending first to the left and then to the right at every node exactly induces a Fibonacci Tree. A Fibonacci Tree of height $h$ has $F(h+2)-1$ vertices where $F(x)$ is the $x$th Fibonacci number. From Binet's formula we can derive a bound on the height $h$ of a Fibonacci tree that contains at least $n$ vertices as $h \leq \lceil \log_\phi \sqrt{5}n \rceil - 2$. From this we can estimate the height bound of a Fibonacci tree with at least $n$ nodes as no more than $\lceil 1.44 \log n - 0.328 \rceil$. Thus a broadcast using Algorithm 7 takes no more than $\lceil 1.44 \log n - 0.328 \rceil$ time steps and involves at most 2 transmissions per node.

The Fibonacci Tree can also be used for convergecast by running the broadcast algorithm in reverse. Each node performs one transmission to its parent



when it has received a transmission from all its children (or if it is a leaf). This guarantees a conflict-free schedule in a Fibonacci tree. Define the *incomplete* Fibonacci Tree of $n+1$ nodes as the Fibonacci Tree of at least $n$ nodes, with some of its leaves truncated such that only $n+1$ vertices remain. For an incomplete Fibonacci tree, conflicts may only occur in the first time step (where pairs of leaves might share a parent). In this case, one of the two leaves simply defers its transmission until the second time step. The latency remains bounded by $\lceil 1.44 \log n - 0.328 \rceil$.

We are now ready to describe the algorithm for the hash-chain-keyed algorithm in fully-connected networks. Let $T$ be an (incomplete) Fibonacci tree with $n+1$ nodes and $s$ at the root, and assume that there is a canonical method for constructing such a tree and numbering its vertices such that all receivers can determine their positions in $T$ from their index $i$ and knowledge of $n$. We perform all communications in the protocol on edges of $T$.

First, let the sender $s$ disseminate $(M_j, N_j, T_j)$ using Algorithm 7. Once all nodes have received this tuple, a convergecast begins at the leaves to collate the XOR of all MAC values backwards towards the sender. In the convergecast, each leaf $d_i$ sends $\mathrm{MAC}_{K_i}(M_j \| N_j \| T_j)$ to its parent; each internal node $d_i$ on receiving values $v_l, v_r$ from its children, computes $v_l \oplus v_r \oplus \mathrm{MAC}_{K_i}(M_j \| N_j \| T_j)$ and forwards that to its parent. At the end of the convergecast $s$ can recompute $A_j$ from the messages received from the network. If this value is equal to $\bigoplus_{i=1}^{n} \mathrm{MAC}_{K_i}(M_j \| N_j \| T_j)$ then all nodes have correctly received $(M_j, N_j, T_j)$ and so $s$ can release $h_j$ to the network (using another round of broadcast using Algorithm 7. Receivers then check $h_j$ and use it to verify the MAC $T_j$ on $M_j$ and $N_j$. The algorithm is described in Algorithm 8.

The algorithm performs three passes of $\lceil 1.44 \log n - 0.328 \rceil$ time steps each. Hence, the total latency is $3\lceil 1.44 \log n - 0.328 \rceil < 3(1.44 \log n + 0.672) = 4.32 \log n + 2.016$ time steps. The total transmission congestion is 9 cryptographic values (including $M$) per node.

**Algorithm 8** Hash-Chain-Keyed Message Authentication for Fully Connected Topology

```
Require: Hash chain $h_0, \dots, h_m$ with all receivers having $h_0$ preloaded.
Require: All nodes know their position in Fib. Tree $T$ and the IDs of their
    children and parent.
 1: $M_j = j$th Message ; $N_j = j$th Nonce
 2: $s$ computes: $T_j = \mathrm{MAC}_{h_j}(M_j \| N_j)$
 3: /* Broadcast Dissemination */
 4: $s$ sends $(M_j, N_j, T_j)$ to all receivers using Algorithm 7
 5: Each receiver $d_i$ inspects $N_j$ to extract $j$. If $d_i$ has seen any messages for
    stage $j$ prior to this, or if $j$ is less than or equal to the latest stage in which
    $d_i$ released a MAC, abort.
 6: /* MAC Aggregation */
 7: repeat
 8:   for all $d_i$ that have received messages from all their children (if any) do
 9:     if this is the first round and $d_i$'s parent has two leaf children, and $d_i$ is
        a left-child then
10:       break
11:     end if
12:     Let $v_l$ (resp. $v_r$) be the message from the left (resp. right) child. If
        there is no such child then let $v_l$ (resp. $v_r$) $= 0$.
13:     $d_i$ sends to its parent: $v_l \oplus v_r \oplus \mathrm{MAC}_{K_i}(M_j \| N_j \| T_j)$
14:   end for
15: until $s$ has received messages from all its children
16: /* $h_j$ dissemination */
17: $s$ checks that XOR of values received are consistent with:
      $A_j = \bigoplus_{i=1}^{n} \mathrm{MAC}_{K_i}(M_j \| N_j \| T_j)$.
18: If so, $s$ sends $h_j$ to all receivers using Algorithm 7
19: Each receiver $d_i$ checks that $h_j$ is the $j$th hash value on the hash chain, and
    that $T_j = \mathrm{MAC}_{h_j}(M_j \| N_j)$. If so, accept $M_j$.
```

| | Latency | Congestion |
|---|---|---|
| **Linear Topology** | | |
| *Unsecured Broadcast* | $n$ | 1 |
| Hash Tree (Sec.7.1) | $2n$ | $\lceil logn \rceil + 1$ |
| Hash Tree w/ r.path (Sec.7.1.1) | $n$ | $2\lceil logn \rceil$ |
| Hash Chain, unoptimized (Sec.8.1) | $3n$ | 5 |
| Hash Chain, w/ encrypted $h_j$ (Sec.8.2) | $2n$ | 6 |
| **Fully Connected Topology** | | |
| *Doubling Broadcast* (Alg.1) | $\lceil \log n \rceil$ | $\lceil \log n \rceil$ |
| *Fib-tree Broadcast* (Alg.7) | $\lceil 1.44 \log n + 1.7 \rceil$ | 1 |
| Hash Tree (Sec.7.2) | $3\lceil \log n \rceil + 1$ | $5\lceil \log n \rceil$ |
| Hash Chain, unoptimized (Sec.8.3.1) | $3\lceil \log n \rceil + 1$ | $5\lceil \log n \rceil + 2$ |
| Hash Chain, optimized (Sec.8.3.2) | $4.32 \log n + 2.016$ | 9 |

Table 1: Latency and congestion bounds of our protocols. Optimal unsecured broadcast protocols are shown in italics for reference.

## 9 Summary of Results

A summary of the results presented in this paper is presented in Table 1. In general, hash-tree based algorithms achieve lower latency and can operate without the need to preload hash chain anchors and synchronized hash chain positions for newly added nodes. Hash chain algorithms can achieve much lower congestion but tend to have slightly higher latency and require more sophisticated administration. All of the results in Table 1 have only a small factor increase in overhead compared to optimally-scheduled unsecured broadcast in the metrics chosen.

## 10 Conclusion and Future Work

We have shown several variants of protocols optimized for latency and congestion in linear and fully-connected topologies. None of these protocols require support for public key cryptography and can thus be used in a wide variety of resource-constrained applications. Each of these protocols plots a different potentially desirable trade-off point in the latency / congestion metrics. An interesting open problem we have not addressed is finding the fundamental limits of symmetric-key methods in achieving low latency and communication overhead in broadcast authentication. Intuitively, it seems that achieving exactly the same latency as unsecured broadcast with only a constant communication overhead is impossible for symmetric-key methods while being easily achievable with public-key methods. In the future we hope to address this hypothesis and its implications for cryptographic protocols in this problem area.

## References

[1] P. Berman, M. Karpinski, and Y. Nekrich. Optimal trade-off for merkle tree traversal. *Theor. Comput. Sci.*, 372:26–36, 2007.

[2] J. Buchmann, E. Dahmen, and M. Schneider. Merkle tree traversal revisited. In *Post-Quantum Cryptography: Second International Workshop, Pqcrypto 2008 Cincinnati, Oh, USA October 17-19, 2008 Proceedings*, page 63. Springer, 2008.

[3] H. Chan and A. Perrig. Efficient security primitives derived from a secure aggregation algorithm. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 521–534, New York, NY, USA, 2008. ACM.

[4] M. Jakobsson, T. Leighton, S. Micali, and M. Szydlo. Fractal merkle tree representation and traversal. In *Topics in Cryptology ? CT-RSA 2003*, pages 314–326, 2003.

[5] D. Liu and P. Ning. Multi-level uTESLA: Broadcast authentication for distributed sensor networks. *ACM Transactions in Embedded Computing Systems (TECS)*, 3(4):800–836, Nov. 2004.

[6] M. Luk, A. Perrig, and B. Whillock. Seven cardinal properties of sensor network broadcast authentication. In *Proceedings of ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, Oct. 2006.

[7] R. Merkle. *Secrecy, authentication, and public key systems.* PhD thesis, Stanford University, 1979.

[8] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. The tesla broadcast authentication protocol. *RSA CryptoBytes*, 5(Summer), 2002.

[9] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler. SPINS: Security protocols for sensor networks. *Wirel. Netw.*, 8(5):521–534, 2002.

[10] M. Szydlo. Merkle tree traversal in log space and time. In *Advances in Cryptology - EUROCRYPT 2004*, pages 541–554, 2004.