Authenticated Communication and Computation in Known-Topology Networks with a Trusted Authority

Haowen Chan

CMU-CS-09-165

September 30, 2009

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Adrian Perrig, chair
Virgil D. Gligor
Anupam Gupta
Panganamala Ramana (P.R.) Kumar
Hui Zhang

Submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

This research was sponsored by the National Science Foundation under grant numbers CNS-0347807 and CCR-0122581, the Council for Logistics Research, Inc. under grant number CLR071345024, the University of California (National Science Foundation) under subcontract SA48961080PG, and International Business Machines under grant number 5002518338. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: secure aggregation, sensor networks, data aggregation, authenticated communication, known topology networks

PhD. Dissertation: Haowen Chan

Abstract

We show that two distinguishing properties of sensor networks, i.e., the presence of a trusted base station, and the pre-knowledge of the fixed network topology, can yield security protocols that are both communication-efficient and highly general. We show new protocols for broadcast authentication, credential dissemination and node-to-node signatures. For securing in-network distributed computations, we show an algorithm for securely computing the sum of sensor readings in the network, which we can generalize to tree computations for any combination of continuous real-valued functions. Each of these primitives involves per-node communication costs that scale logarithmically with the number of nodes in the network, do not require public key cryptography, and are secure against arbitrary coalitions of malicious nodes. The broadcast authentication scheme achieves better properties with fewer assumptions than existing work, and the other new protocols have no known previous approach that do not require either expensive network-wide unicast or public key cryptography.

Acknowledgments

My advisor, Adrian Perrig, has been my teacher, guide, counsellor, and friend during the entire PhD process. He gave unstintingly of his guidance, encouragement and advice to show me the most fruitful avenues of development for myself and my work. I thank him for not only working with me and helping me during this process but also for providing a wonderful role model as a researcher, faculty member and mentor.

I would also like to thank the other members of my thesis committee: Virgil Gligor, Anupam Gupta, P.R. Kumar and Hui Zhang for their guidance and assistance during the process of working on this dissertation. Their insightful feedback and guidance has been invaluable in refining my approach to the thesis and their understanding and encouragement have made the process of researching and writing this dissertation both instructive and fulfilling. I am deeply thankful for their support.

I am grateful to my co-authors as well as the students and faculty at CMU for their gracious collaboration and feedback in the various discussions I have had with them: Ross Anderson, Lujo Bauer, Debabrata Dash, Anupam Datta, James Hendricks, Abhishek Jain, Mark Luk, Bryan Parno, Bartosz Przydatek, Mike Reiter, Dawn Song, Avi Yaar, and Xin Zhang. I would also like to thank my collaborators William Aiello and Patrick McDaniel (at AT&T Research) and Aca Gacic and Soundar Srinivasan (at Robert Bosch Corp.) for their guidance and assistance during my internships at those respective locations.

Finally, I would like to thank my wife, Jody Elliott, for her continuous love, patience and support.

Contents

Chapter 1

Introduction

Sensor networks are an increasingly important application area for distributed algorithms and security protocols. Sensor networks consist of a number of *sensor nodes* deployed over a geographical area. The nodes form a wireless network which forwards sensed data to one or more *base stations* which act as gateways, forwarding the data to off-site servers which can then be queried for the sensed data. The sensor nodes can be used to perform distributed computations to process aggregate sensed data in-network, or to perform control functions (for example, controlling the temperature and light levels of rooms based on occupancy).

Sensor networks represent a new adversary model for which new efficient security protocols must be devised. Traditional adversary models have either assumed that the entire network consists of legitimate processors and security is directed against external threats, or that the network is a self-organized system with no coordinating authority, consisting of autonomous, independent entities each of which may be arbitrarily malicious [34, 45, 48]. The first network model (the Dolev-Yao model) constrains the attacker to control the medium only; the second network model (Byzantine networks) introduces a much stronger attacker model, allowing the attacker to potentially control both the medium, the network topology, and any subset of the active principals.

A realistic attacker model for sensor networks lies in the continuum between these two extremes. In sensor networks, it is infeasible to guarantee the integrity of every processor in the network; since sensor nodes are often deployed in a physically insecure locations and are not designed to be tamper-resistant, a malicious attacker may physically compromise a subset of the nodes and subvert the computation results or alter the messages exchanged in the system. The presence of malicious nodes implies the existence of malicious insiders in the network, making the adversary model necessarily stronger than simple control of the medium.

Although some unknown fraction of sensor nodes may be malicious, it is reasonable from the point of view the sensor nodes to assume that some *master authority* remains uncompromised. This makes the sensor network adversary model somewhat weaker than the Byzantine model, since we can always rely on the fact that the most important principal is trusted by all others and is free from fault.

There are several reasons supporting the argument for assuming the presence of a secure master authority in the design of a distributed sensor network algorithm. First, since the network is owned and operated by a single entity, it is natural to perform administrative functions from a centralized point: this avoids the problems of coordination and consistency in fully distributed implementations. For example, the Zigbee standard specifies a Zigbee Coordinator which performs centralized security services, and topology and membership control [87]. It is much easier to secure a high-capability centralized administrative device than a distributed functionality over resource-constrained sensor nodes; for example, base stations can be made tamper-resistant and remote servers can be stored in locations that are secure against physical compromise, and independent countermeasures can be implemented to resist online intrusion. Finally, given the existence of a centralized point of administration, it is clear that very few functions of the network can be preserved if the central administrative functions are successfully attacked by an adversary. As such, it is reasonable to assume that, from the point of view of the distributed protocols running on the wireless nodes, we must necessarily depend on the availability and security of the central administrative functionality. Actually securing the device that provides the central administrative function is a security problem orthogonal to the design of resilient secure distributed algorithms on the wireless nodes.

The other distinguishing property in our assumption set is that the central point of administration is aware of the static network topology; specifically, we have knowledge of the fixed communication relationships between nodes. This assumption is reasonable for sensor networks since in the majority of applications sensor nodes tend to be stationary and inter-node relationships tend to be stable over time.

In summary, the distinguishing characteristic of the standard sensor network attacker model is that, while insider sensor nodes may be malicious, there exists a secure *trusted administrative functionality* that has a global view of relevant details of the network (for example, including topological relationships of the nodes in the network) and also possesses all security information relevant to any sensor node. This model of node compromise with a trusted authority is also the security model used by Zigbee, the de-facto communication standard for sensor networks [87].

The central thesis of this dissertation is that the two properties of *topology knowledge* and having a *trusted authority* allow the development of powerful and efficient algorithms for distributed computation and communication that achieve performance bounds that are not currently feasible in the more general autonomous network model. A brief discussion of the limitations of existing work follows.

Secure distributed computation for sensor networks, ad-hoc networks and other distributed systems has been approached in distributed systems research with specific algorithms designed for specific application areas [39, 58, 63, 24]. Such protocols typically deal with a specific function computation such as summation and cannot be generalized further; in addition, they often do not work when removed from their specific topological assumptions; furthermore, as is commonly the case in protocols optimised for specific systems, very often their efficiency is only empirically studied through simulation and not asymptotically proven; and their security analyses are informal, typically proving that specific classes of attacks are foiled rather than rigorously reducing the attack problem into a computationally difficult (or impossible) task for the attacker. Our work provides provably secure, efficient and general algorithms for all the basic directions of communications between nodes and the trusted authority in the network, and also for several general modes of in-network computation. It is the first comprehensive general framework for secure and efficient computation and communication in structured networks.

In contrast to the specificity and empirical nature of the protocols proposed by the distributed systems community, research in cryptography and distributed algorithms have examined the reliable computation problem with a more theoretical approach: for the computation of general functions with proven security and asymptotic bounds on time and communication complexity. However these approaches have their own assumptions and inherent weaknesses. Cryptographers study the problem of reliable distributed computation in the form of secure multiparty computation. In this setting, besides requiring that the result of the computation be correct, there is the additional requirement that each party gains no information about what the other parties are contributing as inputs to the computation. To fulfil this stronger requirement, algorithms for secure multiparty computation typically require communication between all parties and are not scalable to large networks. Distributed algorithms typically approach the reliable computation problem in the form of Byzantine agreement. Such approaches generally have similar drawbacks in terms of high communication overhead and requiring high connectivity. Certain algorithms work for bounded-degree graphs but typically these graphs are selected by the algorithm rather than being given as an input; thus limiting their usefulness to parallel computation networks and peer-to-peer overlay networks where any-to-any addressing is possible. In particular, such algorithms are ill-suited to networks with a pre-determined *structure* which is typical of wireless sensor networks.

In contrast to these traditional approaches, in this dissertation we show that, by leveraging the power of the single trusted authority and the predetermined network structure, we can design efficient, secure and general communication and computation protocols. We address a number of general network communication and computation problems with a single construction for authentication and integrity. None of the constructions in this dissertation require the use of computationally expensive public key cryptography, making them highly feasible for resource constrained applications such as sensor networks. Specifically, we design the following mechanisms:

1. Broadcast Authentication: Disseminate a single message from the base station to all nodes.

2. Information Binding: Allow nodes to receive centrally-authorized attestations regarding certain properties associated with them (e.g., a public key), these attestations must be verifiable by any other node.

3. Node-to-Node Signatures: Allow any two nodes to send authenticated, non-repudiable messages to each other without prior key establishment.

4. Secure Aggregation (tree computation): In-network computation of general numerical functions by collating data from the leaf nodes of the topology inwards towards the root. We show constructions with different performance tradeoffs for the addition "sum" computation, for general real-valued functions, and for completely general functions. This allows us to build applications for securely performing data queries on the network as well as to perform secure auditing and monitoring functions.

In our proposed approach, we note that each of these problems can be solved the using the base station as a direct trusted intermediary. We assume that each sensor node shares a unique pairwise key with the trusted base station. This allows us to use the base station as a central point of trust and coordination. The straightforward method for each of these protocols is therefore simply for each node to only communicate with the base station directly. This leads to an unicast-based solution for all the above problems (e.g., broadcast simply becomes multiple unicasts from the base station to every node in the network). The issue with such a simplistic solution is that it is highly inefficient and unscalable. To maximize sensor node lifetimes, we use network congestion as our metric, which is defined as the total communication overhead on the most heavily loaded link in the network. All unicast-based approaches incur $O(n)$ congestion overhead (where n is the number of nodes in the network). In all our proposed protocols, the congestion is either logarithmic in n or proportional to the distance of the farthest node from the base station (which is also roughly $O(\log n)$ in a uniformly distributed sensor network).

We introduce related work in Chapter 2 and define the problem formally in Chapter 3.

In Chapter 4 we describe a primitive based on the distributed construction of cryptographic hash trees; in subsequent Chapters 5 through 8 we show how to use the distributed hash tree as a framework to efficiently "batch" large numbers of unicast communications to and from the base station. This approach allows us to achieve the security functionality of the "base-station to all nodes" unicast method, while achieving $O(\log n)$ congestion overhead in most cases. This yields efficient protocols for highly general applications, providing authenticity for all combinations of node-to-base station, base-station-to-node and node-to-node communication as well as for general tree function computations, which are in most cases secure against *arbitrary* numbers of adversaries.

In Chapter 9, we show a case study of how a higher-level application can be constructed from the general primitives designed in Chapters 4 through 8. We examine the problem of *Rate Limiting*, or controlling the sending rate of nodes in the network. Using the same primitives as those in the Hash Tree framework we show that it is possible to efficiently (with $O(\log n)$ congestion) detect and locate nodes that are exceeding their traffic rates in a tree network.

In Chapter 10, we show how topology knowledge can be used to significantly speed up the broadcast authentication protocol that was proposed in Chapter 5 as well as a well-known subclass of efficient broadcast authentication protocols known as hash-chained authentication protocols. We consider the linear topology (i.e., a path) and the fully connected topology (i.e., the complete graph, representing any-to-any communication such as in the Internet) and show latency optimizations that improve verification time significantly for both class of protocols, as well as a lower bound showing that these verification times are close to optimal.

Our results show that by leveraging known topology and the trusted authority, secure protocols for general communication and computation can be derived with efficiency far surpassing known algorithms for secure computation and communication in the general autonomous network model. These results indicate that secure computation and communication in structured networks with a trusted authority admits algorithms that are fundamentally more practical than those for arbitrary autonomous networks.

Chapter 2

Related Work

In this dissertation we examine how to provide authenticity and integrity for many general forms of computation and communication in structured networks. This dissertation is unique in developing a comprehensive framework for general integrity in fixed-topology networks; however the specific subproblems and practical applications addressed arise in a number of well-studied subfields from various branches of computer science. We describe prior work in each of these in turn, in roughly the order that they are presented in this dissertation.

2.1 Secure Communications in Structured Networks

We review relevant literature related to broadcast authentication, credential dissemination and node-to-node signatures in fixed-topology networks.

In the area of public-key-free broadcast authentication, the general Guy Fawkes protocol framework was proposed by Anderson et al. [1]; Bergadano et al. propose CSA, a variant using explicit synchronization with acknowledgements [4]. CSA uses per-receiver acknowledgements; Yao et al. improve this for tree-based networks by proposing the construction of an authenticated aggregate acknowledgement using hashes [80]. The main drawback of Guy Fawkes schemes is the requirement for *comprehensive acknowledgements*: if even a single receiver did not receive the initial broadcast (or did not acknowledge reception) then the protocol must abort. In comparison, in Chapter 5 we present a scheme for authenticated broadcast that can recover from a few receivers not receiving the initial broadcast. The detailed differences are discussed in Section 5.5.

Perrig et al. propose TESLA [66, 67], which replaces explicit synchronization with loose time synchronization. Improvements to TESLA have been proposed, all requiring some form of time synchronization [51, 19]. Lu and Pooch suggest a parameterization where messages carry their expected time of key disclosure [52]. Zhu et al. propose a lightweight variant for one-hop communications where messages are considered authentic if immediately followed by a valid hash chain value [86]; this approach does not work for multi-hop broadcasts. Luk et al. propose families of broadcast authentication mechanisms [53], but the communication overhead of their one-time signature schemes can be quite substantial.

Multicast MACs (MMACs) are another major class of broadcast authentication mechanisms. An inexhaustive list of work on MMACs follows. Desmedt et al. propose a polynomial based multicast authenticator [21], which is generalized by Safavi-Naini and Wang [71]. Canetti et al. propose an MMAC made by concatenating bits of multiple MACs [11]. All MMACs involve communication overhead proportional to the number of node compromises that the scheme tolerates. Boneh et al. show that this overhead is a lower bound for all MMAC schemes which do not involve digital signature properties [7]. Stream signatures involve amortizing the cost of a signature over a large number of packets [30, 77]. These approaches either require the sender to know the data stream beforehand or use expensive one-time signatures. Zhu et al. implement a variant of the multiple-MAC approach, essentially requiring t individual MACs to protect against up to t malicious nodes [85].

Stream signatures involve amortizing the cost of a signature over a large number of packets [30, 65, 62, 77]. These approaches either require the sender to know the data stream beforehand or use one-time signatures that are expensive to communicate, or require the receiver to buffer a large number of packets before it can reconstruct the authenticator, and are hence more limited in their applicability than general broadcast authentication techniques.

In the field of efficiently building and using hash trees, Ralph Merkle first described the use of a hash tree as a batch authenticator for multiple values [60]. 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.

[8], Berman et al. [6], Michael Szydlo [72], and Jakobsson et al. [42]. However, as always, the overhead of using one-time signatures in this application is extremely high.

To provide resistance against computational DoS attacks for signature-based broadcast authentication in sensor networks, Ning et al. propose several mechanisms [22, 64]. Fortunately, our approaches for authentication and signature are inherently robust against computational DoS attacks.

Relatively little work has been dedicated to the important problem of performing public key management in sensor networks. Ning et al. propose to use hash trees for authenticating node certificates [25]. However, in this work the hash trees are assumed to be preloaded as part of the initial bootstrapping; unlike our Credential Dissemination functionality of Chapter 6, this method cannot be used to efficiently generate and disseminate new credentials on-the-fly.

In the area of efficient sensor network signatures, most work focuses on various optimizations of Elliptic Curve Cryptography to make public key cryptography feasible on resource-constrained nodes. Examples of such work include work by Liu and Ning [50], Malan et al. [57], Gupta et al. [37], and Gaubatz et al. [29]. All these efforts require increased computation latency and code overhead in their implementations of public-key cryptography. In contrast, we propose a novel approach that enables a signature operation based on purely symmetric functions without using one-time signatures, by only trusting the base station.

2.2 Secure Aggregation

Researchers have investigated resilient aggregation algorithms to provide increased likelihood of accurate results in environments prone to message loss or node failures. This class of algorithms includes work by Gupta et al. [36], Nath et al. [63] and Manjhi et al. [58]. However, this family of work only addresses omission of data and does not address the possibility of malicious data manipulation or injection.

A number of aggregation algorithms have been proposed to ensure *secrecy* of the data against intermediate aggregators. Such algorithms have been proposed by Girao et al. [32], Castelluccia et al. [13], and Cam et al. [9]. Hu and Evans [39] propose securing in-network aggregation against a single Byzantine adversary by requiring aggregator nodes to forward their inputs to their parent nodes in the aggregation tree. Jadia and Mathuria [41] extend the Hu and Evans approach by incorporating privacy, but also considered only a single malicious node. While secrecy is an important security property, in this dissertation we focus on the orthogonal problem of providing integrity for the data. None of the schemes described are able to provide both secrecy and integrity simultaneously.

Several secure aggregation algorithms have been proposed for the single-aggregator model. Przydatek et al. [69] proposed Secure Information Aggregation (SIA) for this topology. Also for the single-aggregator case, Du et al. [24] propose using multiple *witness* nodes as additional aggregators to verify the integrity of the aggregator's result. Mahimkar and Rappaport [56] also propose an aggregation-verification scheme for the single-aggregator model using a threshold signature scheme to ensure that at least t of the nodes agree with the aggregation result. Yang et al. [79] describe a probabilistic aggregation algorithm which subdivides an aggregation tree into subtrees, each of which reports their aggregates directly to the base station. Outliers among the subtrees are then probed for inconsistencies. In contrast to this family of work, our work presents an approach for the more general multi-aggregator case.

Wagner [74] addressed the issue of measuring and bounding malicious nodes' contribution to the final aggregation result. The paper measures how much damage an attacker can inflict by taking control of a number of nodes and using them solely to inject erroneous data values, without manipulating the aggregate computation. Our work focuses on the complementary problem of ensuring that the attacker truly is restricted only to this form of attack; hence in essence Wagner's work informs our work by quanitifying the remaining amount of damage that can be inflicted in the presence of an algorithm ensuring that intermediate computations are performed correctly.

Yu proposes sampling techniques to ensure that the computed aggregate tolerates a number of malicious nodes [83]. Like all sampling techniques (such as those proposed by Przydatek et al. [69]), there is a chance of error for the computed result, and the communication costs of the protocol increase for lower error parameters. In comparison, the protocols that we propose are not probabilistic and have no chance of error, effectively restricting the adversary to the tightest bounds possible without assuming any prior distribution on the data.

Manulis and Schwenk provide a rigorous security analysis of our hash tree based framework, for general aggregation computations but for the single aggregator case [59].

2.3 Secure Distributed Computation

The security community has visited the idea of secure distributed computation mainly from the problem definition of secure multiparty computation [34, 18, 70, 10, 12, 26]. This problem is distinct from the reliable distributed computation problem we propose in this dissertation mainly because we drop the secrecy requirement and further assume the existence of a trusted querier; also none of the SMC protocols are able to function in arbitrary network topologies under the sublinear congestion constraint that we propose.

2.4 Distributed Algorithms

The majority of work in byzantine-resilient computation in distributed algorithms has focused on byzantine agreement in fully-connected networks [27, 2]. The majority of these algorithms require $\Omega(n)$ message complexity from each node.

A substantial body of work exists for resilient computation on networks of bounded degree [73, 5, 3, 49, 84]; more recently King et al. describe protocols that require polylog message complexity at each node, where each node has polylog degree [46, 47]. However the majority of this work refers to network *designs* where the bounded degree topology can be selected by the algorithm; typically this topology is constructed in a very specific manner to create a graph with good expansion properties. These algorithms are thus appropriate for parallel computer systems but unsuitable for most wireless networks where the topology is not under the arbitrary control of the algorithm.

A brief sampling of the very wide field of literature in byzantine-resilient systems follows. Wang et al. describe a method for byzantine agreement in a general network that is not fully connected [75]. Hildrum and Kubiatowicz describe a method for making peer-to-peer data structures resilient against limited (non-byzantine) failures [38]. Wang et al. [76] and Drabkin et al [23] and Cheng et al. [20] describe methods for byzantine agreement in adhoc networks. Georgiou et al describe a method for reliable gossip with crash failures [31]. All these methods, as with all literature in distributed algorithms, focuses on the much more difficult problem scenario where there is no reliable trusted authority. As such, not only are their overheads significantly higher for actual computations, but the properties achieved are much more basic: for example, a large body of work exists on the single problem of *agreement*, which in the presence of a trusted authority becomes trivially reducible to broadcast authentication (all nodes simply agree on what the trusted authority says).

Giridhar and Kumar considered the information communication complexity of general function computation in sensor networks without failures [33]. However, this work only addresses communications between two nodes rather than network-wide computations.

2.5 Rate Limiting in Wireless Networks

There are two aspects to packet injection attacks in a sensor network environment: in a data-collection network such as a sensor network, a compromised node may send out *false* data to prevent correct functioning of the network, or it may inject a large number of packets to crowd out legitimate traffic or cause legitimate nodes to exhaust their battery. Previous solutions detect specific spurious packets containing spurious information, for example work by Ye et al. [82] and Zhu et al. [85]. This family of work focuses on early detection of false injected packets such that energy is not expended in forwarding them. However, the need to authenticate the data in each packet require that each event be sensed by multiple nodes that can corroborate each other before the packets are forwarded. Hence, the techniques only apply to *event reports* which are sensed by multiple nodes, which other traffic can still be injected into the network.

In contrast, we are interested in arbitrary data injection attacks where the network traffic is not restricted to specifically formatted event reports. Wood et al. [78] and Karlof et al. [43] summarize such attacks in sensor network environments in their survey paper, however, besides our approach to this problem there have been no publications on general rate-limiting or injection detection that we know of.

Chapter 3

Thesis, Problem Statements and Assumptions

3.1 General Thesis

We hypothesise that the availability of a trusted authority with full pre-knowledge of the network topology allows for the design of highly efficient and general security primitives for communication and computation in structured networks such as sensor networks, ad-hoc networks and peer-to-peer networks.

For brevity, in the remainder of the dissertation we will refer to the trusted authority as the "base station", which is the device in the wireless network most suited for this role. In actual implementation, the trusted authority may in fact be one or more remote machines on a separate network or on the Internet.

Given the presence of the base station, there are three general classes of communication patterns (and associated patterns of computation) in the network: from the base station to the nodes, from the nodes to the base station, and from node to node. To support our thesis hypothesis, we will show secure and efficient algorithms for all the above forms of communication, encompassing several general and common forms of computation and communication authentication with common applications in actual sensor networks.

For base station to node communication, we describe a new construction for broadcast authentication. The new algorithm does not require public key cryptography and has several important advantages over existing protocols: unlike TESLA [66], it does not require time synchronization; unlike the Guy Fawkes schemes [1], it can recover from a small number of receivers not receiving or not acknowledging the original broadcast. We secure two forms of node-to-node authentication: first, we show a protocol for *credential dissemination*, or for authenticating properties of a node that are approved by the central authority (such as public keys, priority levels, or access privileges), and a protocol for *node-to-node signatures* allowing nodes to authenticate messages to each other with the non-repudiation property. Each of these primitives induces $O(\log n)$ communication congestion where n is the number of nodes in the network, and are secure against arbitrary coalitions of malicious nodes.

In terms of securing mass node-to-authority communication, we examine the problem of secure tree computation, of which data aggregation is the primary application. We show secure formulation for computing the SUM function with $O(\log n)$ congestion, which we can generalize to tree computations over arbitrary continuous real-valued functions. The generalization requires $O(hd)$ congestion where h is the depth of the topology (distance of the farthest node from the base station) and d is the maximum degree of a node in the topology. In the case where the subfunctions are identical, associative, and commutative, we can apply an optimization from the SUM secure aggregation protocol to improve congestion to $O(\log n)$. Finally, we show a construction for securing fully arbitrary tree computations against up to k adversaries with $O(3\alpha k(d+1)+\beta hd)$ congestion where α and β are bounds on the communication complexity of the inputs and intermediate results of the subfunction computations, respectively.

To demonstrate the application of these primitives in a higher-level application, we show how these primitives can be applied to *Rate Limiting*, where we bound the ability of an adversary to send large numbers of packets and overwhelming the system.

Finally, we show how topology knowlege in common special-case topologies such as a linear topology or a fully-connected topology yields significant latency speedups in the broadcast authentication schemes we have designed.

3.2 Problem Model

3.2.1 Network Assumptions

We assume a static multihop network with a set $S = \{s_1, \ldots, s_n\}$ of n sensor nodes including a single trusted base station R. We assume that communication relationships can be modeled as a graph with a (simple) edge connecting any two nodes that can communicate. All wireless communications are node-to-node and we do not consider local wireless broadcast communications in this work. This is for two reasons: first, the availability local broadcast does not significantly change the congestion of our algorithms; second, in all our applications we implicitly require receiving nodes to perform acknowledgements and sending nodes to perform retransmissions if acknowledgements are not received. This eliminates any advantage in using local broadcast communications.

We assume that the base station knows the total number of sensor nodes n. The network is organised in a tree topology, with the base station at the root. The default assumption is that the base station also knows the network topology (e.g., which nodes are neighbors of each other); in some cases we do not need this assumption and we will note this in the text.

3.2.2 Security Infrastructure

We assume that each sensor node has a unique identifier s and shares a unique secret symmetric key K_s with the querier. We assume the sensor nodes have the ability to perform symmetric-key encryption and decryption as well as computations of a collision-resistant cryptographic hash function H.

3.2.3 Attacker Model

We assume that the attacker is in complete control of an *arbitrary number* of sensor nodes, including knowledge of all their secret keys. The attacker has a network-wide presence and can record and inject messages at will.

3.2.4 Problem Metrics

As a metric for communication overhead, we consider node *congestion*, which is the worst case communication load on any single sensor node during the algorithm. Congestion is a commonly used metric in ad-hoc networks since it measures how quickly the heaviest-loaded nodes will exhaust their batteries [55, 35]. Since the heaviest-loaded nodes are typically the nodes which are most essential to the connectivity of the network (e.g., the nodes closest to the base station), their failure may cause the network to partition even though other sensor nodes in the network may still have high battery levels. A lower communication load on the heaviest-loaded nodes is thus desirable even if the trade-off is a larger amount of communication in the network as a whole.

Chapter 4

A Protocol for Hash Tree Construction

The basic building block we will use in most of the algorithms in this dissertation is to construct a distributed cryptographic hash tree over the network. The hash tree will be used as a commitment and authentication structure in the subsequent protocols. In this chapter, we describe how to efficiently generate and use these hash trees.

4.1 Background: Hash Trees

A hash tree 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 generated using the rule $p = H[c_1 \| c_2 \| \cdots \| c_m]$ where H is a collision-resistant hash function.

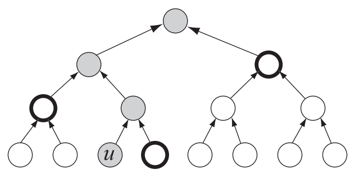


Figure 4.1: Shaded nodes: Authentication path of u; Bolded nodes: off-path vertices of u. Together they form an authentication subtree for u.

This process is repeated until all the vertices form a single tree. Figure 4.1 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 the path from u to r, or its *authentication path.* The authentication path can be computed from the siblings of each vertex on the path from u to r, or the *off-path* vertices of u. This is highlighted in Figure 4.1. The graph induced by the authentication path vertices and the off-path vertices is an *authenthentication subtree* consisting of all nodes from u to r and their siblings, as well as an encoding of the left-to-right order of each set of siblings.

In subsequent sections, we will use the following useful properties of a hash tree:

1. *The root vertex acts as a concise commitment to the structure of the tree and the contents of all 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., finding a hash collision is reducible to finding two distinct hash trees with the same root vertex value).

2. *The root vertex can act as an aggregate authenticator (e.g., signature or MAC) if the leaf vertices consist of unforgeable authenticators.* Since the hash function is collision resistant, the adversary cannot predict the root vertex of a tree with unknown values at the leaves; also, if the adversary is forced to commit to a root vertex value r first, the leaf vertices can subsequently be revealed; due to the one-wayness of the hash function, it remains infeasible for the adversary to determine valid authentication paths for any legitimate leaf values released after the commitment to r.

4.2 The *HT* Protocol

In this section we describe the *HT* protocol for computing a hash tree in a distributed manner over a tree network. We first describe the construction of a hash tree that follows the topology of a tree network; we discuss optimizations in subsequent sections. Let each sensor node i be associated with an arbitrary bit-string L_i. We desire a protocol to construct a hash tree containing all the L_i

HT Functionality
Inputs: (from each sensor node i)
String or value L_i
Computes:
Tree via the following:
Leaf vertex u_i (one per node i): $\langle 0 \| L_i \rangle$
Internal vertex u_0 with child vertices $u_1, \ldots, u_k$:
$\langle H[1 \| u_1 \| \cdots \| u_k] \rangle$
Outputs:
To Base Station: Hash tree root vertex r
To each sensor node i: Off path vertices for u_i

Table 4.1: Definition of the *HT* Functionality

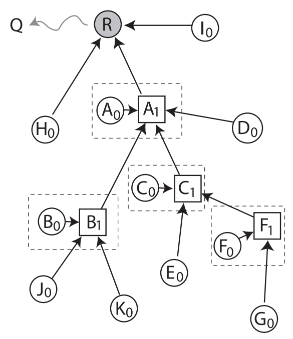
as hash inputs to its leaf vertices (defined as $u_i = H[0 \| L_i]$), such that the base station receives the root r of the hash tree while each node i receives all the off-path vertices between u_i and r (thus allowing i to recompute r). Note that the *HT* functionality is only a building block and by itself is not necessarily secure; for example, the adversary could cause the computation of an incorrect hash tree, or disseminate false verification paths to the nodes. However, the idea is that such inconsistencies can be detected subsequently by other protocols that build on the *HT* primitive. Table 4.1 summarizes the functionality that we wish to implement. The protocol completes in two phases: hash tree construction and authentication-path dissemination.

4.2.1 Hash Tree Construction

The hash tree is constructed from the bottom up in a distributed fashion, with new internal vertices added by each internal node in the network topology. The leaf vertex of the hash tree for a node with ID i is composed of the hash of a bit 0 identifying this as a leaf vertex, concatenated with the input bit-string L_i:

$$v_i = \langle H[0 \| L_i] \rangle$$

For example, in Figure 4.2, sensor node G constructs a leaf vertex consisting of its input value L_G (e.g., a sensor reading) and its node ID G. Each leaf node in the network topology transmits its leaf vertex to its parent (e.g., G sends its leaf vertex to F).



$$R = \langle H[1\|H_0\|A_1\|I_0]\rangle$$

$$A_1 = \langle H[1\|A_0\|B_1\|C_1\|D_0]\rangle$$

$$C_1 = \langle H[1\|C_0\|E_0\|F_1]\rangle$$

$$F_1 = \langle H[1\|F_0\|G_0]\rangle$$

$$G_0 = \langle H[0\|L_G]\rangle$$

Figure 4.2: Basic hash tree, showing derivations of some of the vertices. For each sensor node X, X_0 is its leaf vertex, while X_1 is the internal vertex On the right we list the labels of the vertices on the path of node G to the root.

Each internal (non-leaf) sensor node i in the topology receives from each of its children a hash tree vertex. The parent node i then generates its own leaf vertex, and creates a new parent over its leaf vertex and the vertices supplied by its children as follows:

$$u_i = \langle H[1\|y_i\|u_1\|u_2\|\cdots\|u_k]\rangle$$

Where u_i is an internal vertex created by node i, and hash tree vertices $u_1, \cdots, u_k$ are received from the children of i (or generated by i itself). The leading 1 bit distinguishes the hash inputs that generate an internal vertex from the hash computation that generates leaf vertices (which contain a leading 0 bit).

For example, in Figure 4.2, sensor node A receives internal vertices B_1 and C_1 and the leaf vertex D_0 as well as its own leaf vertex A_0. Node A then creates a new parent vertex by computing a hash over $1\|A_0\|B_1\|C_1\|D_0$. The result is a new internal vertex in the hash tree which is the parent of all the vertices received by A. Once each node i has computed all its internal hash tree vertices it transmits them to its parent, which will then construct its own internal vertex as the parent of all the vertices it receives, and so on. This also occurs at the base station, which computes the root vertex r of the hash tree.

4.2.2 Authentication Path Dissemination

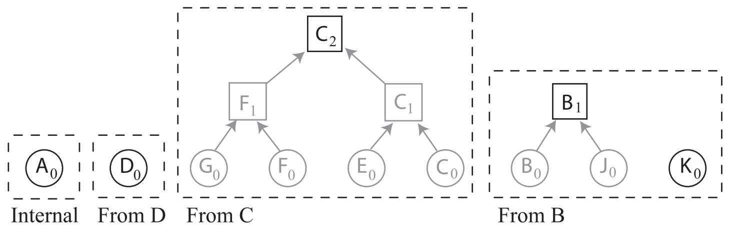
Once the hash tree is constructed, and the root vertex r is disseminated to all nodes, each node must check that its leaf vertex is included in the hash tree rooted at r. Specifically, each node must recompute the sequence of hash tree vertices between its leaf vertex and the root. For example, node G in Figure 4.2 must recompute the vertices F_1, C_1, A_1 and R_1. To perform this series of computations, each node must receive all the off-path vertices of its leaf vertex. In Figure 4.2 this means that node G must receive all the child vertices of F_1, C_1, A_1 and R_1 respectively; this corresponds to the set $\{F_0, E_0, C_0, B_1, A_0, D_0, H_0, I_0\}$. To facilitate this verification, each internal node (including the base station) broadcasts the set of hash-tree vertices it received to all the nodes in its subtree. For example, node A would broadcast the vertices A_0, B_1, C_1, D_0 to all the vertices in the subtree rooted at A on Figure 4.2. This allows each node i in the network to receive the set of off-path vertices of u_i.

Note that Definition 4.1 only requires that the HT algorithm provide the correct authentication paths for each node in the absence of malicious interference. For example, the adversary could cause the construction of an incorrect hash tree, or the dissemination of false verification paths to legitimate nodes. In subsequent chapters we will see how such attacks can be detected in a straightforward way by the protocols that build on the HT primitive.

4.3 Optimizations to the HT Protocol

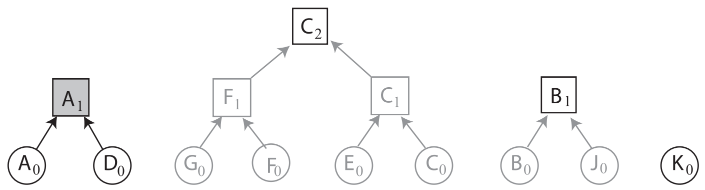
Consider the congestion on the naive scheme. Let h be the height of the hash tree and d be the maximum degree of any node inside the tree. Each leaf vertex has $O(hd)$ off-path vertices, and it needs to receive all their labels to verify its contribution to the aggregate, thus leading to $O(hd)$ congestion at the leaves of the commitment tree. The height h of the aggregation tree depends on the diameter (in number of hops) of the network.

We present a simple optimization to improve the congestion cost. The main observation is that, since the aggregation trees are a subgraph of the network topology, they may be arbitrarily unbalanced. Hence, if we decouple the structure of the commitment tree from the structure of the aggregation tree, then the commitment tree could be perfectly balanced.



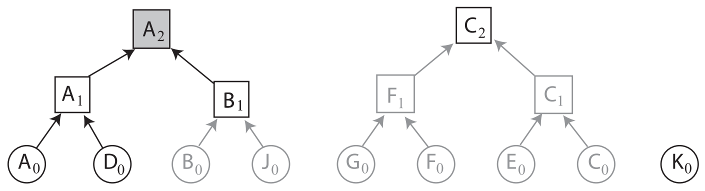
$$A_0 = \langle H[0 \| L_A] \rangle$$
$$D_0 = \langle H[0 \| L_D] \rangle$$
$$C_2 = \langle H[1 \| F_1 \| C_1] \rangle$$
$$B_1 = \langle H[1 \| B_0 \| J_0] \rangle$$
$$K_0 = \langle H[0 \| L_K] \rangle$$

(a) Inputs: A generates A_0, and receives D_0 from D, C_2 from C, and (B_1, K_0) from B. We assume that A is aware of the size of the subtree at each received hash forest vertex. Each dashed-line box shows the commitment forest received from a given sensor node. The solid-line box shows the vertex labels, each solid-line box below shows the labels of the new vertices.



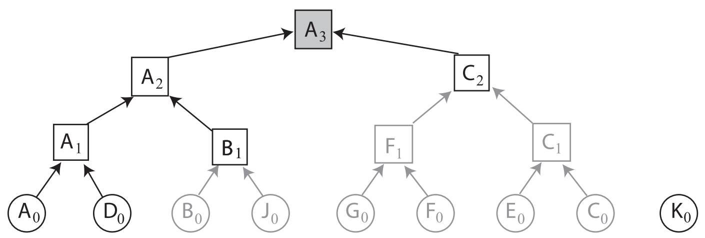
$$A_1 = \langle H[1 \| A_0 \| D_0] \rangle$$

(b) First merge: Vertex A_1 created



$$A_2 = \langle H[1 \| A_1 \| B_1] \rangle$$

(c) Second merge: Vertex A_2 created



$$A_3 = \langle H[1 \| A_2 \| C_2] \rangle$$

(d) Final merge: Vertex A_3 created. A_3 and K_0 are sent to the parent of A in the aggregation tree (not shown: in the transmission, these vertices are labelled with their respective sizes, i.e., A_3 is the root of a subtree of height 3, and K_0 is a leaf)

Figure 4.3: Process of node A (from Figure 4.2) deriving its commitment forest from the commitment forests received from its children.

In the basic hash tree construction, each sensor node passes to its parent a single message containing the label of the root vertex of its subtree T_s. Instead, now let each sensor node now pass on the labels of the root vertices of a *set* of hash subtrees $F = \{T_1, \ldots, T_q\}$. We call this set a *hash forest*, and we enforce the condition that the trees in the forest must be complete binary trees, and no two trees have the same height. These constraints are enforced by continually combining equal-height trees into complete binary trees of greater height. For a set of n leaves, there are always less than $\log n$ trees in the forest, and the tallest tree in the forest has height at most $\log n$.

When a sensor node s receives a hash forest from each of its q children, it must combine the q hash forests $F_1, \ldots, F_q$. We let the intermediate result be $F = F_1 \cup \cdots \cup F_q$, and repeat the following until no two trees are the same height in F: Let h be the smallest height such that more than one tree in F has height h. Find two trees T_1 and T_2 of height h in F, and merge them into a tree of height $h+1$ by creating a new vertex that is the parent of both the roots of T_1 and T_2. This process terminates in $O(\log n)$ steps since each step reduces the number of trees in the forest by one, and there are at most $q \log n + 1$ trees in the forest. Hence, each sensor node creates at most $O(\log n)$ vertices in the hash forest. An example of ths process is shown in Figure 4.3.

Once the base station has received all the hash forests from its immediate children, it then merges them all into a single hash tree by creating parent vertices (possibly over subtrees of different height). The final balanced tree now has a height of $h = \lceil \log n \rceil$ and a degree of $d = 2$. Since the congestion cost of disseminating verification paths for each vertex (internal and external) is $O(hd) = O(\log n)$, and each sensor node is responsible for creating $O(\log n)$ vertices in the hash tree, the final congestion via this method is $O(\log^2 n)$.

Frikken and Dougherty [28] propose an improved optimisation. They observe that, while each node is responsible for creating $O(\log n)$ hash tree vertices, if these vertices are all arranged along a single common path to the root, then they will share a common verification path, thus avoiding the extra $O(\log n)$ multiplicative factor. This rearrangement was induced by introducing a small number of "dummy"vertices. It was shown that the number of such dummy vertices needed was small (relative to n) and the resultant trees still have a height bound of $O(\log n)$. This results in the asymptotically optimal congestion bound of $O(\log n)$.

4.4 Network-wide Aggregated Acknowledgement

The distributed hash tree construction mechanism provides an efficient way to construct hash trees and send authentication paths to the leaves; however this presents only a single-direction mode of authenticated communication in the topology (from the base station to the nodes). In order for all nodes to efficiently communicate back to the base station without causing $\Omega(n)$ congestion near the base station, we describe a novel acknowledgement primitive that is used by many of the protocols that use the *HT* protocol in subsequent chapters.

We desire an efficient protocol to allow every node in the network to respond with some kind of authentic confirmation message to the base station, typically to acknowledge the reception of a valid authentication path. The straightforward method is to have each sensor node send a message to the base station containing some previously agreed-upon confirmation code, e.g., $A_i = \mathrm{PRF}_{K'_i}(N \| OK)$ where K'_i is a key shared with the base station used only for this purpose (it can be generated by hashing a master key K_i shared with the base station), N is a nonce or sequence number to prevent replay, and OK represents the acknowledgement message. Since there is one such message per node, sending all the messages to the base station directly would cause high congestion near the base station. In the case where we are only interested in whether or not the entire network replied with an affirmative confirmation, and not the exact set of nodes which responded, we can aggregate these confirmation messages using XOR: each internal node in the topology waits to receive the confirmation messages from each of its children, then computes the XOR of all received messages with its own confirmation message and forwards the final result to its own parent. If all nodes successfully verified the aggregation process, then the final confirmation value transmitted to the base station is the XOR of all confirmation messages, i.e., $\mathrm{PRF}_{K'_1}(N \| OK) \oplus \cdots \oplus \mathrm{PRF}_{K'_n}(N \| OK)$. Since the base station has knowledge of all the secret keys used in the construction of this aggregate confirmation message, it can reconstruct what it should expect to receive if every node did indeed reply with a confirmation. If the results do not match, then we can infer that some node did not correctly respond with its confirmation message. We first presented this kind of aggregate confirmation code as part of the secure hierarchical aggregation algorithm [17]; it was subsequently analyzed in a more general and formal context by Katz and Lindell [44].

A straightforward generalization of this primitive allows confirmations of specific values rather than just binary acknowledgements. For example, if we desire to confirm that each node i received the same broadcast message M, then the confirmation code for node i could be $M_i = \mathrm{PRF}_{K_i'}(N \| M)$.

4.4.1 Network-wide Confirmations on a Computed Value

This process of network-wide confirmation can be generalized to ensure that all nodes agree upon a specific computed value. For example, if we desire to confirm that each node i performs some computation with the expected result r_i (where r_i is known to the base station), then the confirmation code for node i could be $M_i = \mathrm{PRF}_{K_i}(N \| r_i)$. This allows the base station to implicitly confirm that each node has computed their respective correct results r_i.

Chapter 5

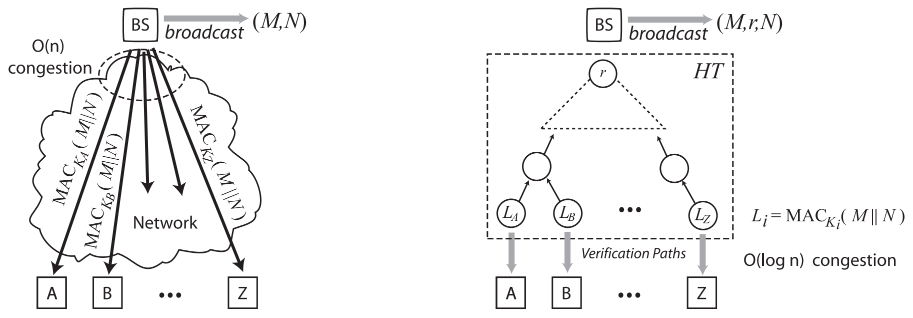
Authenticated Broadcast

In this chapter we show how the HT functionality presented in Chapter4 can be used to generate an authenticated broadcast primitive. Specifically, this primitive enables the base station to send an authenticated broadcast message M such that all nodes can verify that M truly originated from the base station.

5.1 Problem Statement and Assumptions

The Broadcast Authentication functionality can be informally described as follows: the base station constructs a message M, which is disseminated to all nodes in the network. Each node accepts M *only if* M is a legitimate message from the base station. Our goal is to achieve existential unforgeability under adaptive chosen-message attack: in other words, the adversary is allowed to cause the protocol to execute for a polynomial number of messages $M_1, \ldots, M_k$ chosen adaptively by the adversary; if the adversary is subsequently able to cause some legitimate node v to accept any M' which was not in $M_1, \ldots, M_k$ and was never sent by the base station, then the adversary is said to have successfully attacked the protocol. In a secure broadcast authentication scheme, the chances of success should be negligible in the security parameter.

In this chapter, in addition to the assumptions of Section 3.2, we require the additional prior assumption that the network topology is static and known to the base station.



(a) Using unicast (b) Using the HT functionality

Figure 5.1: Disseminating per node MACs: Naive method and HT method

5.2 Using HT for Broadcast Authentication

Before describing the details of the algorithm, we first describe the intuition behind the approach. Consider a base station which shares a unique secret key K_i with each sensor node i. To authenticate message M to node i, the base station can attach a message authentication code (MAC) using the key they share, e.g., $\mathrm{MAC}_{K_i}(N\|M)$ (the nonce / sequence number N is used to prevent replay of M in the future: we assume the nodes keep track of which nonces have been used, and nonces have a fixed length). However, since each MAC for each sensor node uses a different key, unicasting a different MAC to each node in the network is very inefficient, incurring $O(n)$ congestion in the worst case (see Figure 5.1(a)).

Instead of using a separate unicast of a MAC to each node, we can use a hash tree to "batch" the entire set of authenticators into a single structure. Specifically, construct a hash tree whose set of leaves is the set of MACs of M, one for each receiver node, where each MAC is computed using the key shared with the base station and the receiver node. In this application we use a keyed PRF as a MAC, hence the set of batched PRF values would be $\{\mathrm{PRF}_{K_1}(N\|M), \ldots, \mathrm{PRF}_{K_n}(N\|M)\}$. Let the root vertex of the hash tree be r. Then, for each node i, assuming that node i itself has never divulged the value of $\mathrm{PRF}_{K_i}(N\|M)$ in the past, exhibiting the value r suffices as proof of the ability to independently compute $\mathrm{PRF}_{K_i}(N\|M)$ since r was computed via a sequence of collision-resistant hash functions evaluated with $\mathrm{PRF}_{K_i}(N\|M)$ as an input. In other words, we can show that it is computationally infeasible for an adversary that does not know K_i or $\mathrm{PRF}_{K_i}(N\|M)$ to produce a new pair M', r', N' in such a way that $H[0\|\mathrm{PRF}_{K_i}(N'\|M')]$ is a leaf vertex in some hash tree with root r'. Note that in such a construction, each node only needs to verify the inclusion of its own PRF value as an input to a leaf in the hash tree; the PRF values of other nodes are irrelevant.

The *HT* functionality is well-suited for efficiently generating and disseminating exactly such a hash tree. The details are shown in Figure 5.1(b). We assume that the base station is a priori aware of the specific method used by *HT* to construct the hash tree, i.e., given the list of leaf data values, the base station is able to replicate the hash tree constructed by *HT*.

To construct an authentication tag for a message M, the base station internally replicates the hash-tree construction of *HT* using the leaf values $\{\mathrm{PRF}_{K_1}(N\|M), \ldots, \mathrm{PRF}_{K_n}(N\|M)\}$ where K_i is the key shared with node i and N is a nonce that is never re-used. Once the root r of the hash tree is computed, the base station can then broadcast the triplet (M, r, N). To authenticate M, each node first checks that it has never seen N before, then performs the *HT* protocol using $\mathrm{PRF}_{K_i}(N\|M)$ as the input L_i to its hash tree leaf value $u_i = H[0\|L_i]$. The network recomputes the hash tree via the *HT* protocol and releases the relevant verification information to each node i allowing it to verify that u_i is indeed a leaf of the hash tree with root r. If the node successfully verifies this then it can accept M as authentic.

Algorithm 1 Authenticated Broadcast using *HT* Functionality

```
Require: Nonce $N$ not previously used; Message $M$
1: Base station simulates the operation of $HT$ on the leaf vertex definitions $L_i = \mathrm{PRF}_{K_i}(N\|M)$,
   computes root vertex $r$.
2: Base station broadcasts $(M, r, N)$.
3: Each node $i$ checks that $N$ was not previously seen; if so, stop.
4: Otherwise, release $L_i = \mathrm{PRF}_{K_i}(N\|M)$.
5: Nodes collaborate to implement $HT$ functionality, recomputing the hash tree with root $r$.
6: As per $HT$ functionality, verification paths are disseminated back to the nodes after hash tree
   is computed.
7: Each node $i$ verifies that $u_i = H[0\|L_i]$ is a leaf vertex in hash tree with root $r$, with correct
   authentication path shape.
8: If verification successful, node $i$ accepts $M$.
9: (Optional) Base station can request a network-wide ACK by implementing verification confir-
   mation functionality over confirmation messages $C_i = \mathrm{PRF}_{K'_i}(M\|r\|N\|ACK)$ where ACK is a
   unique identifier indicate broadcast authentication success.
```

5.3 Security Analysis

It is computationally infeasible for an adversary to produce a triplet (M', r', N') that correctly verifies for any legitimate node i in the network. This is because, assuming the adversary does not know $\mathrm{PRF}_{K_i}(N' \| M')$ (since the evaluation results of the PRF are unpredictable if K_i is unknown to the adversary), the adversary is computationally unlikely to be able to (a priori) deduce an r' that it knows can be the root of a hash tree containing $\mathrm{PRF}_{K_i}(N' \| M')$. Even though the value $\mathrm{PRF}_{K_i}(N' \| M')$ is subsequently released, since the adversary must commit to r' before learning $\mathrm{PRF}_{K_i}(N' \| M')$, the adversary cannot (with non-negligible probability) find an authentication path linking $\mathrm{PRF}_{K_i}(N' \| M')$ to r' unless it can also break the collision resistance of the hash function. Under this basic idea, we can produce two formal proofs of security for the broadcast authentication protocol. The two proofs use different assumptions: the first proof assumes that the hash function is a good simulation of a random oracle; the second proof only requires that the hash function be collision-resistant, but requires a further assumption that the shape of authentication paths for each node are known to the respective nodes.

5.3.1 A Proof under the Random Oracle Model

We consider the following adversary-vs-challenger security game (Game 0) for the protocol. Following standard cryptographic analysis, we condense the nonce and message into the same bitstring M; the adversary's task is to find some bitstring M not previously authenticated by the base station and convince a legitimate node to accept M as authentic.

- The protocol is initialized as normal for n nodes and all receiver nodes are given their respective keys.
- The adversary selects a target receiver i. The challenger provides the adversary with the keys and states of all receivers that are not i.
- The adversary is allowed to send adaptive queries M_i to the challenger which responds with $T_j = \mathrm{PRF}_{K_i}(M_j)$ for $j = 1, \ldots, q$.
- The adversary now sends M, r to the challenger where $M \neq M_j$ for $j = 1, \ldots, q$.

- The challenger responds with $\mathrm{PRF}_{K_i}(M)$
- The adversary now sends P_i which is an authentication path taking $\mathrm{PRF}_{K_i}(M)$ to r
- The adversary wins if P_i takes $\mathrm{PRF}_{K_i}(M)$ to its preselected vertex value r.

We first analyze the security of a related game (Game 1) where each PRF_{K_i} is replaced with a random function f_i. If we assume that authentication paths must of of a fixed shape for each node i, the security of Game 1 can be proven using a reduction to attacking the collision resistance of the hash function h: this proof is shown in the subsequent Section 5.3.2. We present a shorter proof of a weaker security property that does not assume knowledge of authentication path shape; we show that Game 1 is secure if the hash function $h : R \times R \to R$ is replaced by a random oracle. We can show that the protocol is as secure as other protocols in the random oracle model, i.e., any feasible attack on the protocol must involve finding some feasible exploit of the internal structure of the hash computation.

Let q_1 be the number of random oracle queries made by the adversary before learning $f_i(M)$ (which is the Game 1 counterpart of $\mathrm{PRF}_{K_i}(M)$ in Game 0). With probability at least $1 - 2q_1/|R|$ the adversary has never queried the random oracle on $f_i(M)$ as a left or right input. In this case, adversary has no relevant information that can help to find an oracle queries with $f_i(M)$ as an input and outputting a specific target value. Let q_2 be the number of random oracle queries made by the adversary after learning $f_i(M)$. Now the adversary must find a sequence of oracle queries with $f_i(M)$ as an input and ending in r. The only way to search for such a sequence is to iterate the oracle on $f_i(M)$ to find some path starting with $f_i(M)$ that overlaps with a known path ending in r. Since the hash function takes two arguments, there are at most $2q_1$ values that are known partial pre-images of r (this upper bound occurs in the unlikely worst-case scenario where every one of the first q_1 queries to the random oracle happened to result in r). If, by iterating the oracle on values derived from $f_i(M)$, the adversary lands on any of these $2q_1$ values, then it has derived an authentication path from $f_i(M)$ to r. Each time the adversary queries the oracle, there is a $2q_1/|R|$ chance of successfully landing on a preimage of r. By the union bound the chances of success after q_2 queries is at most $2q_1q_2/|R|$. Summing over these events, the probability of success is at most $2q_1(1 + q_2)/|R|$ which is negligible for q_1, q_2 polynomial in the security parameter $\log|R|$.

It remains to show that converting from Game 1 to Game 0 confers an additional advantage no more than the distinguishability advantage ϵ_F of the PRF. We prove this in the usual way by constructing a distinguisher D for PRF_{K_i} and random function f_i from an adversary A for Game 0. Given oracle access to a function F, we implement Game 0 substituting F for PRF_{K_i}; hence if F is pseudorandom then the game is equivalent to Game 0, if F is a random function then it is equivalent to Game 1. The distinguisher D runs adversary A on the game; and outputs 1 if A succeeds and 0 otherwise. If adversary A has a significantly different chance of success in game 0 vs game 1 then D has a significant chance of distinguishing the PRF from a random function; hence the advantage of A in Game 0 must be no more than ϵ_F higher than the advantage bound on adversaries in Game 1.

One implication of this security proof is that $|R|$ must be sufficiently large that $2q_1q_2/|R|$ is neglible; typical values used to ensure collision-resistance against brute force search suffices for this.

5.3.2 A Proof assuming Hash Function Collision Resistance

We present another security proof that makes a weaker assumption on the properties of the hash function but a stronger assumption on the preloaded information on each node. We show that if the hash function is collision resistant, and each node knows the logical *shape* of its authentication path, then the broadcast authentication scheme is secure. The *shape* of an authentication path of a node u is defined as an unambiguous encoding of the topology of the authentication path from the leaf vertex L_u of u up to the root of the hash tree. It can be encoded as a sequence of numbers indicating which edge is taken on the path from the root to the leaf. For example, in the authentication path shown in Figure 4.1, the shape of the path to u can be encoded as "left,right,left" in a binary tree, or "1 of 2, 2 of 2, 1 of 2" in a more general tree. The notion is that each path shape specifies unambiguously a vertex in the hash tree.

We use a different proof for the security of Game 1 in the proof of Section 5.3.1. We show any adversary that can win Game 1 with non-negligble probability can be used as subroutine to break the collision resistance of H with non-negligible probability.

Let A be a adversary attacking Game 1. We assume that A is deterministic, i.e., for any random choices made by A, it takes as its source of randomness an input of some sequence of random bits

R. We further assume that A can be *reset*, i.e., returned to the initial state in which it started the Game, including all memory and state variables. Let the probability of A winning Game 1 be ϵ_A. We construct an adversary B for attacking the collision resistance of the hash function H which produces a hash collision with probability at least $\epsilon_A^2 - 2^{-k}$ where k is the security parameter (the number of bits of each value output by the random function f).

Adversary B takes as input the hash function H and proceeds as follows. It generates a (correctly randomized) random string of bits R for use as input to A. Adversary B then simulates the challenger in Game 1 to adversary A, using the hash function H as normal. It makes note of the random value y_1 produced in step 5 of Game 0. If A does not win the game, B aborts with failure. Otherwise, B resets A and re-runs the experiment of Game 1 with exactly the same parameters (including the same random string R) except that in step 5 the random value is *reselected* at random, resulting in a value y_2 (which is distinct from y_1 with high probability $1 - 2^{-k}$). Since A is deterministic and the inputs to A are identical before step 5, A will produce the same guess for the root vertex of the hash tree r. If A again wins the game in the second round, B inspects the sequence of hashes that take y_1 to r in the first iteration of the game and the sequence of hashes that take y_2 to r in the second iteration of the game. If $y_2 \neq y_1$, then we know two identically-structured hash sequences take distinct inputs y_1, y_2 to the identical result r; this must imply a hash collision at the first point where the two authentication paths result in identical values. If such a collision is found the B reports success in breaking the collision resistance of H.

It remains to show that the probability of both experiments succeeding is substantial. While each experiment is a legitimate run of Game 1 and must result in success for A with probability at least ϵ_A, it is unclear if the correlation between the two experiments can negatively affect the chances of *both* games being successful. Let X be the space of random coin flips of B during the simulation of Game 1, prior to Step 5 and let Y be the space from which y_1 and y_2 are uniformly selected at random; in this case $Y = 0, 1^k$ since the random function produces a random value of k bits. Let $I(x, y), x \in X, y \in Y$ be 1 if adversary A succeeds given that B makes the coin flips represented by x and produces the challenge y in step 5; let it be 0 if A fails. Since A wins with probability at least ϵ_A over a random choice of x and y, we have that:

$$\sum_x^X \sum_y^Y \mathbf{Pr}[x]\mathbf{Pr}[y]I(x,y) \geq \epsilon_A$$

Equivalently, we can say that $\mathbf{E}[J(x)] \geq \epsilon_A$ where $J(x) = \sum_y^Y \mathbf{Pr}[y]I(x,y)$ and the expectation is taken over all $x \in X$. Let W be the event that both experiments succeed.

$$\begin{aligned}
\mathbf{Pr}[W] &= \sum_x^X \sum_{y_1}^Y \sum_{y_2}^Y \mathbf{Pr}[x]\mathbf{Pr}[y_1]\mathbf{Pr}[y_2]I(x,y_1)I(x,y_2) \\
&= \sum_x^X \mathbf{Pr}[x] \left[\sum_{y_1}^Y \mathbf{Pr}[y_1]I(x,y_1) \sum_{y_2}^Y \mathbf{Pr}[y_2]I(x,y_2) \right] \\
&= \sum_x^X \mathbf{Pr}[x] \left[\sum_y^Y \mathbf{Pr}[y]I(x,y) \right]^2 \\
&= \mathbf{E}\left[J(x)^2\right]
\end{aligned}$$

Now by definition $\mathbf{Var}[J(x)] = \mathbf{E}\left[J(x)^2\right] - \mathbf{E}[J(x)]^2$, and $\mathbf{Var}[J(x)] \geq 0$. Hence:

$$\begin{aligned}
\mathbf{Pr}[W] &= \mathbf{E}\left[J(x)^2\right] \\
&= \mathbf{Var}[J(x)] + \mathbf{E}[J(x)]^2 \\
&\geq \mathbf{E}[J(x)]^2 \\
&\geq \epsilon_A^2
\end{aligned}$$

Hence the chances of both experiments succeeding is at least ϵ_A^2. We only have a collision if the values chosen at step 5 were not the same, hence:

$$\begin{aligned}
\mathbf{Pr}[W] &= \mathbf{Pr}[W \cap y_1 \neq y_2] + \mathbf{Pr}[W \cap y_1 = y_2] \\
\mathbf{Pr}[W \cap y_1 \neq y_2] &= \mathbf{Pr}[W] - \mathbf{Pr}[W \cap y_1 = y_2] \\
&\geq \mathbf{Pr}[W] - \mathbf{Pr}[y_1 \neq y_2] \\
&\geq \epsilon_A^2 - \frac{1}{2^k}
\end{aligned}$$

Hence the probability of adversary B successfully deriving a hash collision is at least $\epsilon_A^2 - 2^{-k}$. If the hash function advantage bound for collision resistance is at least ϵ_H then we have $\epsilon_H \geq \epsilon_A^2 - 2^{-k}$ or $\epsilon_A \leq \sqrt{\epsilon_H + 2^{-k}}$ which bounds the advantage of adversary A to be negligible in the security parameter. Similar to the proof in Section 5.3.1 we can then convert Game 1 to the original Game 0 with an additive factor of ϵ_F which is the distinguishability advantage bound of the PRF, yielding a final advantage bound for the protocol of $\sqrt{\epsilon_H + 2^{-k}} + \epsilon_F$.

5.4 Nonce Implementation Details

A potential implementation pitfall exists. Once a triple (M', r', N') with a valid nonce N' is received by node i, the node will release $\mathrm{PRF}_{K_i}(N' \| M')$ to allow the rest of the network to perform distributed verification. The release of this value potentially allows an adversary to now compute some new r'' such that (M', r'', N') will verify correctly and be accepted by node i. Hence it is important that a nonce must never be re-used for the same key, e.g., they could be increasing sequence numbers.

Keeping track of which nonces have been used introduces a new problem. Since we have to remember which nonces have been used, random nonces are impractical. Suppose we use increasing sequence numbers, and nodes keep track of the largest sequence number they have seen. Hence, a node will only release its PRF value for triples with sequence numbers larger than the largest one yet seen. This introduces a long-term denial of service attack where the adversary can shut down a node for an extended period of time with a single spurious triple containing a large sequence number. To address this issue, we propose replacing the sequence number with a hash chain. A hash chain is constructed by repeatedly evaluating a pre-image resistant hash function h on some (random) initial value. The final result (or "anchor value") is preloaded on the nodes and the base station uses the pre-image of the last-used value as the nonce for the next broadcast. For example, if the last known value of the hash chain was $h^m(IV)$, then the next broadcast would use $h^{m-1}(IV)$ as the nonce. When a node receives a new nonce N', it verifies that N' is a precursor to the most recently received (and authenticated) nonce N on the hash chain, i.e., $h^i(N') = N$ for some i bounded by a fixed k of number of hash applications. This prevents an adversary from performing sequence number exhaustion denial of service attacks since it would have to reverse the hash chain computation to get an acceptable pre-image. The hash computations do present an additional minor opportunity for a computational DoS (by flooding a node with multiple messages containing invalid nonces, each of which must be checked to show it does not belong on the hash chain); however since hash computations are efficient and the maximum number of hash computations per message is bounded by some parameter k, such attacks cannot cause persistent outages. New nodes entering the network can be sent their hash chain anchors via unicast: this is a one-time operation and does not increase the congestion complexity of the protocol.

Since the hash chain is generated at the resource-rich base station (not on the nodes), it can potentially be extremely long. When the base station wishes to generate a new hash chain, the new anchor value can be efficiently broadcast to the nodes in the network using the remaining values on the old hash chain. This renewal process is subject to disruption by adversary nodes in the usual way (e.g., by releasing spurious leaf values to cause the hash tree authentication step to fail). Hence, if hash chain renewal is performed too late (e.g., when the old hash chain only has a few values remaining), then, if the renewal broadcasts are disrupted by the adversary, legitimate nodes may not receive the new anchors. In this case the entire protocol must be reinitialized via an expensive unicast from the base station to each node. To remedy this, hash chain renewals can be performed early, when there is a sufficient number of remaining hash values such that a short period of disruption attacks by an adversary does not result in exhaustion of the values in the old hash chain. A *continuous* DoS by a stubborn adversary in every round of broadcasts can still incapacitate the broadcast mechanism; as discussed in Section 3.2 we assume that such persistent adversarial behavior can be addressed out-of-band.

5.5 Discussion and Analysis

Like all protocols in this dissertation, the overhead of this protocol is a bound of $O(\log n)$ congestion on all links in the network.

The algorithm is highly efficient in both communication and storage and has a clear advantage over Multi-MACs and One-time signatures, both of which can be prohibitively large in size. Furthermore, it does not require public key cryptography which avoids the issues of having to implement public key cryptography on resource-limited nodes or opening a vulnerability to resource-draining public-key verification attacks.

In comparison with TESLA [66], our scheme does not require time synchronization. This avoids the necessity for accuracy in the internal clocks on sensor nodes or for the need to perform periodic secure time synchronization across the entire network, which would be yet another source of communication overhead.

In comparison with Guy-Fawkes schemes, our scheme has the advantage of being able to recover from partial reception. In the Guy Fawkes scheme, the broadcast message M is first disseminated with $\text{MAC}_K(M)$; the sender then receives acknowledgements from all receivers, and once the acknowledgements are all received, then the key to the MAC K is disseminated to all receivers. The key K is part of a hash chain and is hence self-authenticating; after receivers have obtained K, they can then use it to verify the MAC. One disadvantage of delayed key dissemination is that as long as a single receiver has not acknowledged reception of the broadcast message, the key cannot be safely disclosed, since it may be used to forge a MAC that will be accepted by any receivers which did not receive the original message. For example, if node i did not receive the original broadcast message but the sender prematurely disseminates K, the adversary can then use K to construct a forged message M' with a correct $\text{MAC}_K(M')$ that will be accepted by node i. Hence, if any receiver does not respond with a positive acknowledgement, the Guy Fawkes scheme terminates without any form of recovery; a second round of network-wide broadcasts must then be made. In contrast, with the HT-based scheme, suppose some node i could not receive the initial broadcast.

The parent node j of i could note that i did not acknowledge the reception of the broadcast and inform the base station. The base station can then selectively repair the protocol by just sending the relevant value of $\mathrm{PRF}_{K_i}(N\|M)$ to j. This allows j to effectively complete the protocol on behalf of node i, rebuilding the hash tree such that the rest of the network can authenticate M. This repair operation involves just a single unicast, and does not affect the security of the protocol, because unlike in the Guy Fawkes case, the single value $\mathrm{PRF}_{K_i}(N\|M)$ is not useful for computing the PRF value on another forged message M'. In particular if i eventually manages to restore its connectivity, it can do so simply by verifying the known value of $\mathrm{PRF}_{K_i}(N\|M)$. In this example we discussed only repairing the protocol for a single unreachable node i; clearly this is directly generalizable to any subset S of unreachable nodes with an increase in communication congestion of $|S|$ PRF values.

Chapter 6

Credential Dissemination

With the authenticated broadcast primitive described in Section 5, we now have the ability to create and disseminate hash trees with root vertices that are authenticated by the base station. One application of this is to use a single authenticated value (i.e., the root vertex of a hash tree) to attest to the integrity of many different leaf values. In this section we consider the problem of authentically binding information to specific nodes using this structure.

6.1 Problem Statement and Assumptions

Similar to the assumptions in Section 5, we assume the base station has full pre-knowledge of the topology of the network.

The general formulation of the credential dissemination problem is as follows: suppose that each node i has a label L_i which is known both to itself and to the base station. Our goal is to enable i to prove to an arbitrary node j that the label L_i is legitimate, e.g., it was approved by the base station and not fabricated by an adversary (which could be controlling node i itself). Intuitively, we can think of the problem as issuing credentials L_i to each node i such that the credentials are existentially unforgeable under adaptive chosen-credential attack by an adversary. That is, the adversary is given k rounds where it runs the protocol to attach arbitrary credentials to each node (k is polynomial in the security parameter); subsequently, the adversary attempts to produce a forged credential L_i' for node i (which may be malicious) to a legitimate node j, where L_i' was never issued to i in the query phase. If node j accepts the credential, then the adversary has successfully broken the credential scheme; for the scheme to be secure the probability of this successful attack should be negligible in the security parameter.

The general information-binding problem described above is analogous to the task of efficiently issuing credentials to large set of principals. Public key management is an important application of this functionality.

6.1.1 The Public Key Management Problem

Public key management is one of the most important applications of the information-binding functionality. With continued improvements in the performance of elliptic curve algorithms, public key cryptography hardware and software are becoming increasingly feasible for low cost sensor nodes. However, at the moment it is still unclear how to manage public keys in a sensor network. There are two major problems with deploying asymmetric key cryptography related to public key management.

Public key authentication. To prevent node-in-the-middle attacks, sensor nodes should not accept public keys from any other sensor node except those with which it knows it should associate. The standard method of implementing this is with a PKI, i.e., all public keys are certified (signed) by a central authority (e.g., the trusted base station). This is subject to a battery-exhaustion denial of service attack from an outsider who can bombard a legitimate node with thousands of false public key certificates. This problem is particularly serious in sensor networks due to the resource constraints of sensor nodes.

Public key revocation. As nodes die or are revoked from the network, their old public keys must be invalidated. The simplest approach is a centralized approach: for each public key that a node receives, the node must communicate with the base station to verify the current status of the newly received public key. This is expensive in communication overhead and does not scale to large networks. The standard distributed method for public key revocation is either for a node to keep extensive certificate revocation lists (CRLs), or for authority signatures on public key certificates to periodically time out. Neither approach is practical for sensor networks. CRLs are impractical because sensor nodes cannot spare the RAM to exhaustively remember the identities of every dead node. Certificate timeouts are also impractical since periodically unicasting a newly signed public key certificate to every node in the network is prohibitively expensive.

6.2 Using HT for Credential Dissemination

We show that any form of credential binding and dissemination can be performed similarly to the process described in Section 5. We focus on public key dissemination as a concrete case of this functionality; in general this functionality can be used to bind and distribute any label to each sensor node.

We use the HT functionality with leaf vertex values $L_i = (PK_i, i, T)$ where PK_i is the public key of node i (which is known to both node i and the base station), and T is a sequence number or timestamp which guarantees the freshness of the certificate. Similarly to Section 5, we assume that the topology is static and the hash tree formation/dissemination algorithm HT is known to the base station. Hence, the base station can, internally (i.e., without communicating with any nodes), construct a hash tree with each of the L_i as the leaves in an identical manner to the HT functionality. The root r of this hash tree is then disseminated to the network using authenticated broadcast. The method of Section 5 may be used for the authenticated broadcast. Depending on the freshness method used, T may also be included in the authenticated broadcast (to invalidate any old certificates). To prevent desynchronization, a network-wide acknowledgement of reception of this broadcast is performed using the process described in Section 4.4. This ensures that all nodes receive the most up-to-date root vertex r. Subsequently, each node releases its L_i. Using these as inputs to the leaf vertices, the HT functionality generates a hash tree with the same root vertex r as was broadcast by the base station, and disseminates sufficient information to each node to allow it to reconstruct the sequence of hashes leading from L_i to r. Once each node has successfully performed verification, then it can retain the information it used in verification for use as a proof of validity to any other node in the network which also remembers or can verify that r is a valid root vertex from the base station. To ensure that all nodes receive their respective proofs of validity for their labels, one final round of the network-wide confirmation functionality is performed. The algorithm is summarized in Algorithm 2. Note that L_i can in fact be constructed

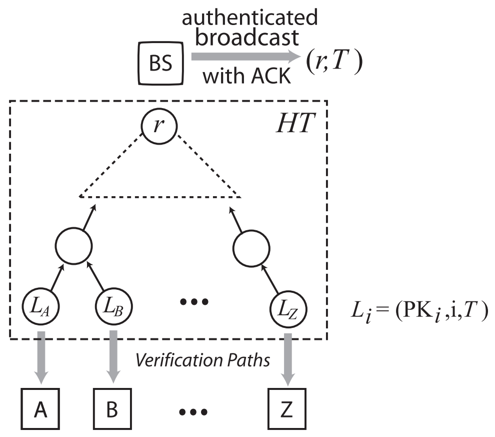

authenticated broadcast with ACK
BS
(r, T)
r
HT
L_A
L_B
L_Z
$L_i = (\mathrm{PK}_i, \mathrm{i}, T)$
Verification Paths
A
B
Z

Figure 6.1: Use of HT as a Public Key distribution primitive

arbitrarily and hence the protocol supports the dissemination of arbitrary credentials, not just public key certificates.

6.2.1 Composition with Authenticated Broadcast of Chapter 5

If we desire to use the method of Section 5 for the authenticated broadcast portion of this protocol, the two protocols can be merged in a straightforward way to reduce the number of rounds of communication.

A direct composition of the method for authenticated broadcast of Chapter 5 with the credential dissemination protocol of Section 6 is as follows: First compute the credential hash tree T_c over leaf vertex values L_i to determine the root vertex r_c of the credential tree, then invoke the broadcast authentication method of Chapter 5 with $M = r_c$. This involves computing a second hash tree T_b over $\mathrm{PRF}_{K_i}(r_c \| N)$ and disseminating and checking authentication paths in T_b.

The direct composition method involves two trees T_c and T_b, one for credentials and one to authenticate the root of the credential tree, and involves multiple runs of the HT protocol (once for each tree). We make the observation that in fact the functionality of the two trees can be combined into a single tree T. The crucial observation is the following generalization of the security argument for the broadcast authentication protocol of Chapter 5:

Theorem 1. *If the HT protocol is performed on leaf values consisting of the evaluation of a keyed pseudorandom function on an input that contains a fixed length non-repeated nonce field as a prefix,*

Algorithm 2 Public Key Dissemination using HT Functionality

```
Require: Certificate expiry time $T$; Public keys $PK_i$ for each node $i$ (known to BS and node $i$).
1: Base station simulates the operation of $HT$ on the leaf vertex definitions $L_i = (PK_i, i, T)$, computes root vertex $r$.
2: Base station authentically broadcasts $(r, T)$.
3: Each node, on reception, discards old root vertex and updates its current root vertex to $r$.
4: Verification confirmation functionality is used to ensure all nodes received the broadcast. If not, stop.
5: Each node releases $L_i = (PK_i, i, T)$.
6: Nodes collaborate to implement $HT$ functionality, recomputing the hash tree with root $r$.
7: As per $HT$ functionality, verification paths $P_i$ are disseminated back to the nodes after hash tree is computed.
8: Each node $i$ verifies that $u_i = H[0\|L_i]$ is a leaf vertex in the hash tree with root $r$.
9: Verification confirmation functionality used to check that all nodes received their correct verification paths successfully. If not, stop.
Public Key Authentication Procedure:
1: When node $i$ declares its public key certificate $L_i = (PK_i, i, T)$ to another node $j$, node $i$ authenticates $L_i$ by including the information to recompute verification path $P_i$.
2: Node $j$ can authenticate that $PK_i$ is a valid key by recomputing the path $P_i$ and checking that it terminates in the most recent root vertex $r$.
```

then the root r of the hash tree constructed by the HT protocol is self-authenticating to receiver i if it is sent to receiver i before i releases its leaf value. Self-authentication of r refers to the property that it is hard for an adversary to find some r' such that it can provide an authentication path for a legitimate node taking its leaf value to r'.

Proof. We can in fact prove this directly using a minor modification of any of the proofs for the broadcast authentication protocol in Chapter 5, i.e., by just slightly modifying the games in Section 5.3.1 or Section 5.3.2 to prove the authenticity of the hash tree root r instead of the broadcast message M. For brevity, here we present an argument that the security of the broadcast protocol implies the theorem. We use the standard technique of first replacing the PRF with a random function. Let A be an adversary that can break the self-authentication property of the theorem, i.e., it can first choose some r' so that upon being given a random value v_i, it can then find an authentication path taking v_i to r' within polynomial time with probability ϵ. Then we construct an adversary B for the broadcast authentication scheme in the natural way: first B picks an arbitrary message M' for forgery. Then B calls A to receive get its guess for the hash tree root

r'; then it queries its target node on M' to get the random value $v_i = \mathrm{PRF}_{K_i}(N\|M')$. This value v_i is then given to A which provides an authentication path taking v_i to r' with probability ϵ. Then B gives the authentication path is then given to the target node i which will accept, thus the attack succeeds with probability ϵ. □

Note that the theorem is just a more general (i.e., weaker) form of the security theorem for the broadcast authentication scheme, since if the theorem is not true then we can achieve arbitrary message forgery for the broadcast authentication scheme and not just existential forgery.

We can now present the modification of the credential dissemination algorithm of Section 6.2 to eliminate the need for an explicit round of broadcast authentication. We simply alter the definition of the leaves of the hash tree. In the original definition of Section 4.2.1, the hash tree leaves were defined as:

$$v_i = \langle H[0\|L_i]\rangle$$

Where L_i were arbitrary bit strings. For the broadcast authentication application of Chapter 5, each L_i was a keyed PRF evaluation:

$$v_i = \langle H[0\|\mathrm{PRF}_{K_i}(N\|M)]\rangle$$

We thus extend the definition to arbitrary bit strings L_i for each node:

$$v_i = \langle H[0\|\mathrm{PRF}_{K_i}(N\|L_i)]\rangle$$

Where N is a non-reusable nonce. To prevent the adversary from flooding the network with invalid nonces, the hash chain method of Section 5.4 may be used. This change removes the need to distribute r using authenticated broadcast; unsecured broadcast will suffice. The rest of the algorithm remains the same.

6.3 Security Analysis

The protocol involves the dissemination of a hash tree throughout the network with an authenticated root hash value. Correctness thus follows from the observation that, for each hash tree T with a root hash value r, it is computationally infeasible for an adversary to find another hash tree $T' \neq T$ that also has root hash value r. If the protocol is completed successfully, then every unrevoked node will have received its certificate and can compute the sequence of hashes to the publicly-known root r; revoked nodes are not part of the tree and cannot produce any sequence of hashes to r.

A more rigorous proof follows. We show that a successfully forged credential must involve either a forged root vertex r' (thus breaking the broadcast authentication of r), or else the adversary can derive a collision on the hash function H. We define the existential credential forgery game as follows: For query rounds $j = 1, \ldots, k$, the adversary can select arbitrary labels $L_{i,j}$ for each node $i = 1, \ldots, n$. The challenger responds by disseminating these $L_{i,j}$ as credentials, i.e., it constructs a tree T_j with root vertex r_j, sends r_j to all nodes using authenticated broadcast, and sends out $L_{i,j}$ to node i along with its authentication subtree $T_{i,j}$. We assume the adversary also receives all these communications. After the query rounds are complete, the adversary can attempt to forge a credential L' (for an arbitrary node) where L' was not in any of the previous queries, i.e., $L' \neq L_{i,j}$. If the adversary can induce any of the nodes to accept this credential then it wins the forgery game.

Suppose the adversary did not forge a root vertex r'. Then the forged credential L' has an authentication subtree T' to a legitimate root vertex r. Since r is legitimate, there must be some legitimate hash tree T with r as the root which does not have L' as a leaf that was constructed in one of the query rounds; the adversary thus knows this entire tree T. We show an adversary B attacking the collision resistance of H. Adversary B performs a traversal of the vertices in the authentication path in T', starting at the root r and proceeding with increasing depth. Let the vertex v'_j on the authentication path at depth j, and v_j be the corresponding vertex at the same position in T. At the beginning of the iteration for depth j (starting at depth 0), we have that $v'_j = v_j$ (at depth 0, they are both identically r). We have three cases. If both vertices are leaf vertices, $v'_j = H[0\|L']$ and $v_j = H[0\|L]$ where $L \neq L'$ since T does not contain L' as a leaf. Hence we have found a hash collision. If one of v_j, v'_j is an internal vertex and the other is a leaf vertex, then the internal vertex is computed from $H[1\|\cdots]$ but the leaf vertex is computed from a hash computed over a different value: $H[0\|\cdots]$, since $v_j = v'_j$ this implies a hash collision. If both vertices are internal vertices in T and T', we can compare the child vertices of v'_j in T' to the child vertices of v'_j in T. If they are different in any way, then we have found a hash collision. If they are the same, then we continue on the to next depth down. If B cannot find any collisions in the entire traversal then $T' \subseteq T$, implying that L' was a leaf in T which is a contradiction.

Hence, if the adversary's advantage in attacking the broadcast authentication is at most ϵ_{BA}, and the adversary's advantage in attacking the collision resistance of H is at most ϵ_{CR}, then the advantage of the adversary in generating a forged credential is at most $\epsilon_{BA} + \epsilon_{CR}$.

6.4 Performance Analysis

The public key dissemination method described in this section incurs $O(\log n)$ congestion overhead, and can be used to either refresh keys periodically or as needed to revoke old keys. The HT method of public key management has two advantages over conventional mechanisms:

1. The certificate-verification attack is negated because authenticating a public key only requires $O(\log n)$ hash function evaluations, which is significantly faster than a public key signature verification for any network of reasonable size.

2. Public key revocation is greatly simplified: each time a node is revoked, the base station reforms the topology around the revoked node (via a series of authenticated unicasts to the nodes affected by the change) and then repeats the public key binding algorithm for a total of $O(\log n)$ congestion overhead. Given that node revocations are infrequent occurrences, this is a significantly lower overhead than periodically unicasting newly signed certificates to each node, and also does not require the use of node revocation lists.

3. Instead of explicit revocation, if we assume loose time synchronization, we can efficiently deploy time-limited certificates in the network. Suppose we wish each public key certificate to be valid for at most T time periods; then every T time periods the network can repeat the public key binding algorithm for just $O(\log n)$ congestion overhead instead of having to

re-distribute new certificates to every node in the network.

One potential drawback of using the HT functionality in this manner is its vulnerability to denial of service attacks. Specifically, a malicious node in the algorithm can sabotage the hash tree dissemination process causing the verification of legitimate nodes to fail. However, such an attack is much less severe than the certificate-revocation attack because: (a) it is easily detectable (via the verification confirmation functionality in the algorithm) and, once detected, countermeasures can then be taken to locate and revoke the malicious node; (b) the attacker can only disrupt one round of the algorithm per attack, with no lasting impact on the network, instead of being able to completely drain the physical battery reserves of a given node; (c) the attacker can only perform the attack from inside the network using a compromised node instead of being able to freely perform the attack from an external device outside of the network.

Hence, using the information-binding functionality in this manner to perform public key management addresses a difficult problem in a highly efficient manner.

6.5 Further Applications

Public key management is only one example of the usefulness of the HT structure in authoritatively binding information to a specific node. In general, the same algorithm can be applied to create a publicly verifiable attestation to the veracity of any deterministic node property. We call the general property the "information-binding functionality". We include a short list of some briefly described examples to highlight its generality and usefulness.

Network access prioritization. In certain applications, nodes with tighter requirements on latency or bandwidth may need prioritized access to the network. For example, more aggressive MAC layer access, or prioritized traffic queues. The information binding functionality can be used to efficiently bind priority levels to nodes such that any neighbor node can readily verify the authorized priority level of a node.

Local topology control. The only topology that is required to be static for the purposes of applying the HT functionality is the aggregation tree structure; nodes may be free to associate with other nodes within their immediate neighborhood to exchange sensed information or for coordination functions such as sleep scheduling. Topology control may be necessary in such situations to prevent a given node from associating with nodes outside of its designated neighbor set. This can be implemented with an authorized neighbor list bound to each node.

Node type credentials. Nodes should not be able to masquerade as entities that they are not. For example, a light switch should not be allowed to claim that it is a fire alarm. Credentials binding nodes to their roles can be used to prevent this kind of unauthorized claims by malicious nodes.

Coordination schedules. Deterministic schedules can be bound to nodes. For example, to ensure a fair rotation as cluster head node, to ensure sensor coverage in sleep scheduling, and to ensure uniform power consumption. Publicly verifiable bindings of specific schedules to nodes can allow local groups to work out fair schedules without fear of cheating. For example, if a deterministic random sequence is bound to each node, this can be used to arbitrate which node gets to be cluster head at any given time.

Chapter 7

Node-to-Node Signatures

The HT functionality can be further applied to create a node-to-node signature scheme requiring only each node to share a secret key with the (universally trusted) base station. A node-to-node signature scheme allows any node u to create arbitrary messages which will be recognised as authentically from u by any other node v in the network, even if v had no prior contact with u.

Consider the following simple solution: for each node i, the node sends its message L_i (authentically, using the secret key shared with the base station) to the base station. When node i wants to prove the authenticity of L_i to another node j, it just instructs j to check with the base station. Since the base station is completely trusted to tell the truth, we achieve all the properties we require. Unfortunately having every authentication go directly through the base station is prohibitively expensive in terms communication congestion.

Our observation is that, through the use of a hash tree, the base station can efficiently "batch authenticate" the origin of an entire set of messages $L_1, \ldots, L_n$ by just authenticating the root vertex r of the hash tree constructed over these messages. The verification path of each L_i to the root vertex r then acts as a proof of authenticity which can be verified by any node that also knows the veracity of the root vertex r.

We will follow a similar protocol to that of Chapter 6, but with the per-node labels L_i now independently chosen by the nodes instead of selected by the base station. Since the base station now cannot precompute a hash tree over the relevant labels L_i, we let the network perform this computation; to verify that the computation was correct, we must allow for each node to independently check its own label in the computed hash tree. Once this check is completed, we can then use the network-wide acknowledgement primitive (Section 4.4) to confirm this to the base station, which will the release an authentication of the computed hash tree root r. This allows the respective L_i to be authenticated just by producing their respective verification paths in the hash tree.

7.1 Problem Statement and Assumptions

Similar to the assumptions in Section 5, we assume the base station has full pre-knowledge of the topology of the network.

The Node-to-Node signature problem is defined as follows: suppose each node i in the network has a message L_i (where L_i could be arbitrarily chosen by i itself, with the base station unaware of this choice). We wish to provide the capability for each node i to create a single tag (signature) indicating that i was responsible for L_i. This signature should be verifiable by any other node in the network, i.e., it should have the non-repudiation property (given the signature as evidence, i cannot deny that it was responsible for L_i).

Furthermore, the signature scheme should be existentially unforgeable under an adaptive chosen-message attack; i.e., the adversary is first given oracle access to the protocol for generating signatures for a set of node-message pairs, and then is challenged to find some node-message pair i, m that was not previously queried, where i is legitimate, such that some legitimate node j will accept that m was from i.

7.2 Using HT for Node-to-Node Signatures

Before the algorithm can be executed, we must first bind each node identity to a specific verification path of the hash tree. We assume that the node topology is static and known to the base station. For a given HT algorithm operating on a fixed topology, assuming that each node contributes exactly one vertex to the hash tree, each node must have a fixed path from its vertex to the root of the hash tree constructed by HT. The base station can compute this verification path from its knowledge of the topology and the HT algorithm, and can thus bind the path to the node identity using the protocol of Section 6. Note that this path is constant regardless of the data value contributed by the node to the HT functionality; in a network with (mostly) fixed topology this binding only needs to be performed each time the topology changes. We assume that each topological binding b has an identifier s_b (e.g., a sequence number that increases by one each time the topology changes and a new binding is issued to the nodes). This identifier is embedded into the binding of nodes to paths; specifically, in binding b for each node with identifier ID_i, we bind the tuple $\langle s_b, P_i, ID_i \rangle$ indicating that in the topological binding s_b, node ID_i has path P_i.

As mentioned, the intuition behind the algorithm is that the base station performs a "batch authentication." The algorithm proceeds in the following steps: (1) HT constructs a hash tree over the set of values to be authenticated; (2) each node i checks that L_i is in the correct position in the hash tree and (3) the base station confirms this to all nodes using an additional self-authenticating broadcast.

We make the standard assumption that the messages L_i have some property that makes them useless for replay (e.g., timestamp, or sequence number, or application-level message idempotency).

The details of the algorithm are as follows: each sensor node i inputs its L_i to the HT protocol, which then constructs the hash tree in the usual way. The root r of this hash tree is reported to the base station. The base station then authentically broadcasts to all nodes the message $\langle r, s_b, H[N'] \rangle$ where s_b is the identifier for the current topological binding, N' is a randomly chosen nonce and $H[\cdot]$ is a pre-image resistant hash function. Note that the method of Section 5 can be used for the authenticated broadcast. The HT functionality provides the requisite information such that each node can perform distributed verification (i.e., each node i recomputes the hash tree vertices from its leaf vertex L_i to the authenticated root vertex r). An important *additional* step is performed during this verification: each node i must check that the verification path computed in this process is exactly the authenticated verification path P_i that is bound to its ID prior to the algorithm. Verification confirmations are collected from all nodes with the verification confirmation message from node i being $\mathrm{PRF}_{K_i'}(r \| s_b \| H[N'] \| OK)$. If the base station determines that all distributed verification has succeeded, then it broadcasts the value N' (this message is self-authenticating since $H[N']$ was part of an earlier authenticated broadcast). This means that r

should be considered valid and can be used to verify the authenticity of messages from other nodes. Once the value N' is received by a node, a final round of verification confirmations is performed using $\text{PRF}_{K'_i}(r\|s_b\|N'\|OK)$ as the confirmation message.

After this process, a node i can authenticate its message L_i to node j as follows. As a signature over L_i, Node i transmits to node j all the verifying information it used in the distributed verification step it performed (i.e., j gets enough information to reconstruct the path from L_i to r). Node i also transmits the binding of its verification path to its identity (for the topological binding with identifier s_b). Upon reception, Node j checks that the value r is a valid root vertex, i.e., in some earlier phase, the hash pre-image N' was released by the base station indicating that all nodes (including i) must have successfully completed distributed verification over r prior to this. Node j also checks that when r was broadcast by the base station, the topological binding identifier associated with r is s_b. Node j then verifies L_i using the information provided by Node i, confirming that the verification path is indeed the one bound to node i's identity (in the topological binding s_b) and that the final root vertex computed is r. If the checks complete successfully then Node j knows that L_i must have been originated from node i. Since the verification process is identical for all nodes, node j can retain all the verification information it used and use it to prove the origin of L_i to any third-party node j'. This shows the non-repudiable quality of the signature, i.e., once i proves to j that it originated L_i, it cannot retract that claim since the proof that j now holds is publicly verifiable. The algorithm is summarized in Algorithm 3.

Note that we are unable to apply the arguments of Section 6.2.1 for making the hash tree root self-authenticating in this case, because the root value r is only computed after nodes have revealed their leaf values.

7.3 Security Analysis

A proof sketch of unforgeability follows. Suppose that an adversary was able to present to some node j a successfully forged message $L'_i \neq L_i$ purportedly from some legitimate node i. Let r be the hash tree root computed by j. Since j accepted the message authentication, it must have received $N', \langle r, s_b, H[N'] \rangle$ in a prior authenticated broadcast; if the adversary did not successfully forge

Algorithm 3 Signature Scheme using HT Functionality

```
Require: Replay-resistant messages $L_i$ for each node $i$
Require: Each node $i$ bound to a fixed verification path topology with a proof $P_i'$ and a binding identifier $s_b$ (See Section 6)
1: Each node releases $L_i$.
2: Nodes collaborate to implement $HT$ functionality, recomputing the hash tree with root vertex $r$.
3: Root vertex $r$ is reported to the base station. Base station picks a random $N'$ and disseminates $(r, s_b, H[N'])$ using authenticated broadcast
4: As per $HT$ functionality, verification paths $P_i$ are disseminated back to the nodes after hash tree is computed.
5: Each node $i$ verifies that $u_i = H[0\|L_i]$ is a leaf vertex in the hash tree with root $r$ via its fixed, known verification path.
6: Verification confirmation functionality (with $C_i = \mathrm{PRF}_{K_i'}(r\|s_b\|H[N']\|OK)$) used to check that all nodes succeeded in verification. If not, stop.
7: Base station broadcasts $N'$
8: Upon reception of $N'$, nodes store $r$ as being usable for authentication.
9: Verification confirmation functionality (with $C_i = \mathrm{PRF}_{K_i'}(r\|s_b\|N'\|OK)$) is used to ensure all nodes received the broadcast. If not, stop.
Message Authentication Procedure:
Require: Sender $i$, Receiver $j$, Message $L_i$
1: Node $i$ transmits $L_i, r$ along with information to recompute its verification path $P_i$ and proof of correct path topology $P_i'$.
2: Node $j$ checks that $r$ is a valid root vertex, recalls the topological binding identifier $s_b$ that was associated with $r$, then verifies that $L_i$ is a descendant of $r$ in the position expected of $i$ as established by $P_i'$ for binding identifier $s_b$.
```

this broadcast message then it is from the base station. The base station would only release this broadcast if the authenticated acknowledgement phase completed successfully; unless the adversary successfully attacked the authenticated acknowledgement primitive, this implies that node i has successfully verified the inclusion of L_i in the hash tree with root vertex r, in the location bound to its ID in the topological binding s_b. Assuming that the topological binding is authentic, $L_i' \neq L_i$ implies a hash collision somewhere in the verification path that j computed and the verification path that i computed (the reasoning is similar to the argument presented in Section 6.3). In summary, the adversary has to break the broadcast authentication protocol, the aggregated authentic acknowledgement protocol, the credential binding protocol, or the collision resistance of the hash function. Let the advantage bounds for the adversary in attacking the broadcast authentication protocol, aggregate authenticated acknowledgement, and hash collision resistance be $\epsilon_{BA}, \epsilon_{AAA}$ and ϵ_{CR} respectively. Then, substituting the result from Section 6.3, the advantage of the adversary in forging a node-to-node signature is no more than $2\epsilon_{BA} + \epsilon_{AAA} + 2\epsilon_{CR}$.

7.4 Performance Analysis

The authentication primitive described in this section allows the protocol designer to build protocols where nodes can construct messages that can be origin-authenticated by any other node in the network. Previously, the only known method for such a capability involves the use of public-key cryptography; our scheme uses only symmetric key cryptography. More significantly, the protocol does not involve any kind of prior key establishment algorithm to provide this authenticity: each node only needs a single unique key shared with the base station.

We note that not only does the authentication structure give unforgeability (i.e., integrity and source verification) properties, it also has the property of non-repudiation in the sense that once a message with an authentic tag of this form is released, the originating node cannot plausibly deny that it was responsible for creating the message (assuming the base station is not compromised). This makes this authentication structure somewhat more useful than, for example, a MAC using a shared secret key between two nodes (where the originating node can always claim that the verifying node was the one which actually originated the message). Based on this property, it is clear that such tags can be used to create publicly-verifiable commitments; such commitments can be used, for example, to expose nodes which attempt to cheat in a protocol.

The overhead of the scheme is a signature of length $O(\log n)$, the generation of which causes $O(\log n)$ congestion in the network.

7.5 Applications of Node-to-Node Signatures

The use of node-to-node authenticated broadcast in constructing general resilient applications are too numerous to discuss in detail. Some examples include: allowing node cluster heads to authentically broadcast schedules to their children; authenticated broadcasts of node power levels (e.g., for traffic shaping); authenticated routing distance metrics (for secure routing). In this section we briefly focus on some examples of uses of the authentication primitive in constructing other basic security protocols.

Multi-message Signatures

A basic limitation of this HT-based signature scheme is that nodes must generate signatures in coordinated network-wide phases; each phase allows each node to generate an authenticator for an arbitrary message L_i. In the case where a node has several messages $M_1, \ldots, M_k$ that it may wish to authenticate in a given phase, it could generate a hash tree over these k messages and set L_i to be the root of the tree. Then each message M_i could be individually authenticated by showing that M_i is a leaf in the hash tree rooted at L_i. Clearly, the messages $M_1, \ldots, M_k$ must be fixed prior to executing the network-wide phase and cannot be changed once L_i has been signed. For a more flexible signature method, we can assign each M_i as the "public key" of a one-time signature (such as Merkle-Winternitz signatures [61]). The nodes may then use these authenticated one-time public keys to sign up to k fixed-length messages at any time without needing network-wide coordination. If loose time synchronization is available, then broadcast authentication techniques like μTesla are more efficient and can effectively authenticate a much larger number of messages without byte-length constraints. We describe the details below.

Initializing Hash Chains for μTesla.

The HT-based signature primitive described in this section complements nicely with the μTesla broadcast authentication scheme described by Perrig et al. [68]. When the HT-based signature scheme is used to bootstrap μTesla, the two schemes cover each other's weaknesses. The HT-based signature has the following weaknesses: (1) inflexibility: it requires the entire network to participate in signing one message from each node; (2) long signatures: each signature carries $O(\log n)$ hash values of authenticating information. The μTesla scheme does not suffer either of these weaknesses but instead has the drawback of being troublesome to bootstrap: it requires a per-source hash chain "anchor" value to be somehow loaded onto every verifying node. Due to these issues, μTesla is typically only used for authentication from base station to node. The HT-based signature scheme enables node-to-node use of μTesla, because it provides an easy way to reload hash chain anchors onto the receiving nodes. When used in this fashion the drawbacks of HT-based signatures are minimized since, due to the time-synchronized nature of μTesla, all nodes need to refresh their hash chain anchors at approximately the same time. Once the hash chain anchors are initialized on the receiver nodes, the more efficient and flexible μTesla can be used for broadcast authentication.

Distributed Node Revocation.

One application of this authentication primitive is in the distributed revocation protocol of Chan et al. [14]. In that protocol, when one node u detects another node v to be acting maliciously, a local broadcast (i.e., a broadcast transmission to all nodes in the neighborhood of v) is used to issue "revocation votes" against the detected node, such that if enough neighbors vote indicating they believe v to be malicious then v is ejected from the network. The original protocol required the use of deterministic key establishment schemes (e.g.,the random pairwise scheme [16]). In such key distribution schemes, for each node v, there exists a fixed set of nodes S_v each of which shares a preloaded key with v. In the revocation scheme, each of the nodes in S_v is thus given a revocation vote against node v and these votes are authenticated using a hash-tree mechanism with the votes as the leaves. Since each of these votes (and their authenticating information) needs to be stored on the node, this yields a massive memory overhead: each node must store around $|S_v|$ revocation votes each of which requires $O(\log |S_v|)$ authentication information; since $|S_v| = O(n)$, this means

$O(n \log n)$ 128-bit hash values must be a-priori loaded onto each sensor node. Furthermore, since the nodes that share a pairwise keys with a given node are a subset of the neighbors of a given node, only a small subset of a given node's neighbors can issue a revocation vote against it. With the use of the HT-based signature scheme, distributed revocation can now work independently of the key distribution scheme, because any node u can issue a signed message voting for the revocation of any other node v. This signature can be verified by any other node without needing any additional preloaded information except for the single key that node shares with the base station. Specifically, each node can create a message voting for the revocation of each of its neighbors in the network, and authenticate each of them in a single pass of the HT-based signature protocol using the multi-message signature of Section 7.5. This reduces the storage overhead of the scheme to $O(\log n + \delta)$ for a node with δ neighbors. This greatly increases the practicality of distributed node revocation.

Chapter 8

Secure Data Aggregation

The problem of secure data aggregation is motivated by the particular problem of securely and efficiently performing aggregate queries (such as MEDIAN, SUM and AVERAGE) on sensor networks. In-network data aggregation is an efficient primitive for reducing the total message complexity of aggregate sensor queries. For example, in-network aggregation of the SUM function is performed by having each intermediate node forward a single message containing the sum of the sensor readings of all the nodes downstream from it, rather than forwarding each downstream message one-by-one to the base station. The energy savings of performing in-network aggregation have been shown to be significant and are crucial for energy-constrained sensor networks [81, 40, 54].

In this section we explore the general problem of securing in-network computation. Informally, we consider a computation which takes as inputs various unknown sensor readings in the network and outputs a result to the base station. This distributed computation is performed *in-network*, by having sensor nodes compute and exchange intermediate results with each other before the final result is passed to the base-station. Given that a number of nodes in the network may be maliciously modifying the intermediate results of certain subcomputations, we would like to bound the amount of deviation that can be caused from the correct result which would have been returned if all nodes had acted honestly.

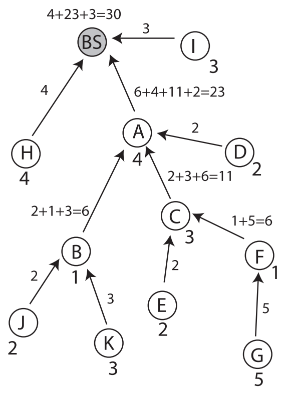


Figure 8.1: Standard tree-based SUM aggregation as described in TAG. Numbers next to nodes represent data/readings; Numbers and sums on edges represent intermediate sums sent from nodes to parents.

8.1 Problem Statement and Assumptions

For this chapter, we will only consider real-valued converging tree computations, i.e., computations over a tree topology, rooted at the base station, where each node produces only one intermediate result, which is sent to its parent. An example of such a tree computation is computing a sum over all the sensor readings in the network.

We first introduce the problem statement for real-valued aggregates such as SUM, and later modify this to address more general computations. Each node that is a leaf in the tree topology reports its input value (e.g., a sensor reading) to its parent. Once an internal node has received data from each of its children, it evaluates the *intermediate aggregation operator* over this data and its own reading. In the case of the summation aggregation, the intermediate aggregation operator is addition, i.e., an intermediate sum of the data received from each of the node's children and the node's own reading is performed. The result of the intermediate aggregation operation is reported to the node's parent, which then repeats the process until the final sum is computed at the base station. An example of this process is shown on Figure 8.1.

Each sensor node s_i reports a data value a_i. We assume that the data value is a *non-negative* bounded real value $a_i \in [0, r]$ for some maximum allowed data value r. We assume that the value has a constant size representation. The objective of the aggregation process is to compute some function f over all the data values, i.e., $f(a_1, \ldots, a_n)$. Note that for the SUM aggregate, the case where data values are in a range $[r_1, r_2]$ (where r_1, r_2 can be negative) is reducible to this case by setting $r = r_2 - r_1$ and add nr_1 to the aggregation result.

Definition 2. *A**input value injection attack** occurs when an attacker modifies only the data readings reported by the nodes under its direct control, under the constraint that only legal readings in $[0, r]$ are reported. For all other parts of the protocol, nodes under attacker control behave honestly, i.e., they adhere strictly to the algorithm definition.*

Without domain knowledge about what constitutes an anomalous sensor reading, it is impossible to detect a direct input value injection attack, since they are indistinguishable from legitimate sensor readings [69, 79]. Hence, if a secure aggregation scheme does not make assumptions on the distribution of data values, it cannot limit the adversary's capability to perform direct input value injection. Under such conditions we can thus define an maximum level of aggregation security as follows.

Definition 3. *An aggregation algorithm provides **input-value-injection-equivalent integrity** if, by tampering with the aggregation process, an adversary is unable to induce the querier to accept any aggregation result which is not already achievable by a direct input value injection attack.*

This definition provides a *soundness* property for accepted aggregation results; a corresponding *completeness* property simply states that if the sensors each report legitimate readings, and the algorithm proceeds correctly (i.e., as defined, without random or byzantine failures in processing or communication) in all nodes, then the querier should accept the final result.

8.2 Secure SUM Aggregation

We now address the secure computation of the sum of all inputs of the nodes in the network. Note that, unlike the preceding chapters, we do not require pre-knowledge of the network topology for the SUM algorithm to function.

The basic idea behind the secure SUM aggregation protocol is to perform three steps: first, we use the HT structure of Chapter 4.2 (with some modifications) to commit to all the aggregation operations that occured in the network; second, we use these commitments to prove to each legitimate node that every computation that is dependent on the input supplied by the node is computed correctly; finally, we ensure that every node successfully validates the computations that are dependent on their inputs and confirm this using the network-wide confirmation protocol of Section 4.4.

8.2.1 Lower-bounding the SUM Protocol

We first show a construction for lower-bounding the reported aggregate result, i.e., we detect aggregate results that are less than the total sum of all the values input by the legitimate nodes. Subsequently, we will show that, given some reasonable assumptions, upper-bounding the aggregate is reducible to the lower-bounding problem, resulting in full input-value-injection-equivalent integrity. First, we focus on the lower-bounding problem.

We use a natural generalization of the *HT* functionality that uses the internal vertices of the hash tree to commit to the intermediate aggregation operations. We define a *sum hash tree* in an equivalent way to the hash tree construction described in Section 4.2. We now incorporate the computation results into the hash tree vertices. In the basic hash tree, we defined parent vertices as the hash computed over all the child vertices, i.e., $p = H[c_1 \| c_2 \| \cdots \| c_m]$. In a sum hash tree, each vertex v additionally contains the sum of all the leaf values in the subtree rooted at v.

Specifically, leaf vertices remain mostly unchanged, containing the ID i of the sensor node and its input to the aggregation, v_i:

$$u_i = \langle v_i, H[i, v_i] \rangle$$

Each internal vertex in the sum hash tree contains a value v_u in addition to the standard cryptographic hash information:

$$\langle v_u, H[c_1 \| c_2 \| \cdots \| c_m] \rangle$$

In the above, $c_1, c_2, \ldots, c_m$ are the child vertices of the node, and v_u is the sum of the values in these child vertices, i.e., $v_u = v_{c_1} + v_{c_2} + \cdots + v_{c_m}$. It is clear from this construction that the value of each internal node is the sum of the values at the leaves of the subtree rooted at the node; in particular when the base station receives or computes the root vertex of the summation hash tree, then it will obtain the sum of all inputs of the sensor nodes in the network. Furthermore, the sum hash tree retains all the integrity properties of a hash tree. In particular it is hard to find two trees with the same root vertex r.

We can use the sum hash tree to provide a lower bound for the sum aggregate as follows. At the start of the algorithm, nodes execute the HT protocol with the above modifications to create a sum hash tree, the root of which is reported to the base station. As a result of the HT protocol, each sensor node also receives a verification path from its input value to the root. The base station distributes the root vertex r that it received to the rest of the network (this broadcast need *not* be authenticated). Now, each sensor node recomputes all the sum hash tree vertices between its leaf vertex and the received root vertex r. The basic idea is that if any (deliberate or malicious) errors were committed in the process of building the sum hash tree, causing some internal vertex u to *not* contain the sum of the values of its children, this will result in an inconsistent root vertex to be computed by any legitimate sensor node in the subtree rooted at u. The nodes use the network-wide confirmation functionality of Section 4.4 to inform the server of the correctness of the verification computation. If all sensor nodes successfully perform verification, then each node will release the confirmation message $M_i = \mathrm{PRF}_{K_i}(N \| r)$ (where N is a freshness nonce associated with this run of the protocol). If the base station receives the XOR of all these confirmation messages as expected, then it accepts the aggregation result. If any sensor node fails its verification then it will compute a different root r' of the hash tree resulting in a different confirmation message $M_i = \mathrm{PRF}_{K_i}(N \| r')$. This will cause the aggregated confirmation message to be different from what the base station expects and so the base station will reject the result.

8.2.2 Analysis of the Lower Bound Property

We present an informal argument for the security of the protocol. Elaboration of these arguments into formal reductions of cryptographic hardness is straightforward. All theorems below are stated using standard crpytographic hardness assumptions and assuming an adversary with polynomially bounded computing power; for brevity, these assumptions are omitted from the actual theorem statements.

We assume that the adversary is able to freely choose **any** arbitrary topology and set of labels for the final sum hash tree. We then show that any such tree which passes all the verification tests must be have input-value-injection-equivalent integrity. First, we define the notion of an *inconsistency*, or evidence of tampering, at a given node in the commitment forest.

Definition 4. *Let $t = \langle v_t, H_t \rangle$ be an internal vertex in a commitment forest. Let its two children be $u_1 = \langle v_1, H_1 \rangle$ and $u_2 = \langle v_2, H_2 \rangle$. There is an* ***inconsistency*** *at vertex t in a commitment tree if $v_t \neq v_1 + v_2$ or any of $\{v_1, v_2\}$ is negative.*

Informally, an inconsistency occurs at t if the sums don't add up at t, or if any of the inputs to t are negative. Intuitively, if there are no inconsistencies on a path from a vertex to the root of the sum hash tree, then the aggregate value along the path should be non-decreasing towards the root.

Definition 5. *Call a leaf-vertex u* ***accounted-for in*** *r if there is no inconsistency at any vertex on the path from the leaf-vertex u to the root r of the sum hash tree, including at the root vertex.*

Lemma 6. *Suppose there is a set of accounted-for leaf-vertices with distinct labels $u_1, \ldots, u_m$ and committed data values $v_1, \ldots, v_m$ in the sum hash tree. Then, either we can show a hash collision for H, or the final aggregation value at the root vertex is at least $\sum_{i=1}^{m} v_i$.*

Lemma 6 can be rigorously proven using induction on the height of the subtrees in the forest. Here we present a more intuitive argument.

Proof. (Sketch) We show the result for $m = 2$; a similar reasoning applies for arbitrary m. By assumption, u_1 (resp. u_2) is accounted-for in a hash tree T_1 (resp. T_2) with root v_r. Let the path from u_1 to v_r in T_1 be P_1. Let the path from u_2 to v_r in T_2 be P_2. Since both P_1 and P_2 end in v_r, there exists a longest suffix P_a that belongs in both P_1 and P_2 and ends in v_r. Since $u_1 \neq u_2$, P_a is shorter than either P_1 or P_2. Let the first vertex in P_a be t, and let its predecessor in P_1 be c_1, and its predecessor in P_2 be c_2. By definition of P_a, $c_1 \neq c_2$. Let the ordered set of children of t in T_1 be C_1, and the ordered set of the children of t in T_2 be C_2. Since t is generated by computing a hash on C_1 (or C_2), if $C_1 \neq C_2$ then we have a hash collision on H. Otherwise, let $C_1 = C_2 = C$. Since $c_1 \neq c_2$, c_1 and c_2 are distinct vertices in C. The value of c_1 must be at least v_1 since there are no inconsistencies on the path between u_1 and t. Similarly the value of c_2 is at least v_2. Since there was no inconsistency at t, vertex t must have aggregation value at least $v_1 + v_2$. Since there are no inconsistencies on the path from t to the root of the commitment tree, the root also must have aggregation value at least $v_1 + v_2$. Negative root aggregate values are detected by the querier at the end of the aggregate-commit phase, so the total sum of the aggregate values of the roots of all the trees is thus at least $v_1 + v_2$. □

8.2.3 Reducing Upper Bounds to Lower Bounds

The protocol of Section 8.2.1 ensures that the reported aggregate is no lower than the sum of all the values reported by the legitimate nodes. Recall that sensor node values are bounded by $[0, r]$. We re-use the lower bound functionality to create a corresponding upper bound. The basic intuition is that, for a computation which produces results in a bounded range (in this case, the sum aggregate should always be between 0 and nr), then applying an upper bound is equivalent to applying a lower bound "from the top" of the range.

Let the complement $\overline{v}_i$ of a value v_i be defined as $\overline{v}_i \overset{\text{def}}{=} r - v_i$. Using the protocol of Section 8.2.1 we simultaneously compute both the sum of all values $S = \sum_{i=1}^{n} v_i$ as well as the sum of the complements of all values $\overline{S} = \sum_{i=1}^{n} \overline{v}_i$. Specifically, we modify the sum hash tree to also keep track of the complement sum. Leaf vertices are thus defined as:

$$\langle v_i, \overline{v}_i, i \rangle$$

Each internal vertex in the sum hash tree contains a value v_u in addition to the standard cryptographic hash information:

$$u = \langle v_u, \overline{v}_u, H[c_1 \| c_2 \| \cdots \| c_m] \rangle$$

Where, $c_1, c_2, \ldots, c_m$ are the child vertices of the node u, and $v_u = v_{c_1} + v_{c_2} + \cdots + v_{c_m}$ and $\overline{v}_u = \overline{v}_{c_1} + \overline{v}_{c_2} + \cdots + \overline{v}_{c_m}$.

At the end of the aggregation process, the base station checks the constraint that the aggregated sum of all the complement values is the complement of the aggregated sum over the normal values, i.e., $S + \overline{S} = nr$. If this constraint does not hold then the result is rejected.

Theorem 7. *Let the final* SUM *aggregate received by the querier be S. If the querier accepts S, then $S_L \leq S \leq (S_L + \mu r)$ where S_L is the sum of the data values of all the legitimate nodes, μ is* *the total number of malicious nodes, and r is the upper bound on the range of allowable values on each node.*

Proof. Suppose the querier accepts the SUM result S. Let the COMPLEMENT SUM received by the querier be $\overline{S}$. The querier accepts S only if $S+\overline{S} = nr$ and the leaf vertices of each legitimate sensor node are accounted-for. The set of labels of the leaf vertices of the legitimate nodes is distinct since the labels contain the (unique) node ID of each legitimate node. Since all the leaf vertices of the legitimate sensor nodes are distinct and accounted-for, by Theorem 6, $S \geq S_L$ where S_L is the sum of the data values of all the legitimate nodes. Furthermore, $\overline{S} \geq \overline{S_L}$, where $\overline{S_L}$ is the sum of the complements of the data values of all the legitimate nodes. Let L be the set of legitimate sensor nodes, with $|L| = l$. Observe that $\overline{S_L} = \sum_{i \in L} r - a_i = lr - S_L = (n - \mu)r - S_L = nr - (S_L + \mu r)$. We have that $S + \overline{S} = nr$ and $\overline{S} \geq nr - (S_L + \mu r)$. Substituting, $S = nr - \overline{S} \leq S_L + \mu r$. Hence, $S_L \leq S \leq (S_L + \mu r)$. □

Note that nowhere was it assumed that the malicious nodes were constrained to reporting data values between $[0, r]$: in fact it is possible to have malicious nodes with data values above r or below 0 without risking detection as long as $S_L \leq S \leq (S_L + \mu r)$.

Theorem 8. *The* SUM *algorithm provides input-value-injection-equivalent integrity.*

Proof. Let the sum of the data values of all the legitimate nodes be S_L. Consider an adversary with μ malicious nodes which only performs direct input-value injection attacks. Recall that in a direct input-value injection attack, an adversary only causes the nodes under its control to each report a data value within the legal range $[0, r]$. The lowest result the adversary can induce is by setting all its malicious nodes to have data value 0; in this case the computed aggregate is S_L. The highest result the adversary can induce is by setting all μ nodes under its control to yield the highest value r. In this case the computed aggregate is $S_L + \mu r$. Since we assume that the set of legal inputs for a single node can be any number between $[0, r]$, any aggregation value between S_L and $S_L + \mu r$ is also achievable by direct input-value injection. The bound proven in Theorem 7 falls exactly on the range of possible results achievable by direct input-value injection, hence the algorithm is input-value-injection-equivalent by Definition 3. □

8.2.4 Building Other Aggregation Functions from SUM

In this section we briefly discuss how to use the SUM algorithm as a primitive for the COUNT, AVERAGE and Φ-QUANTILE aggregates. Note that in the subsequent section (Section 8.3) we will describe how to generalize the secure aggregation primitive for SUM to more general function classes.

The Count Aggregate

The query COUNT is generally used to determine the total number of nodes in the network with some property; without loss of generality it can be considered a SUM aggregation where all the nodes have value either 1 (the node has the property) or 0 (otherwise). More formally, each sensor node s has a data value $a_s \in \{0, 1\}$, and we wish to compute $f(a_1, \ldots, a_n) = a_1 + a_2 + \cdots + a_n$. Since count is a special case of SUM, we can use the basic algorithm for SUM without modification.

The Average Aggregate

The AVERAGE aggregate can be computed by first computing the SUM of data values over the nodes of interest, and then the COUNT of the number of nodes of interest, and then dividing the SUM by the COUNT.

The Φ-Quantile Aggregate

In the Φ-QUANTILE aggregate, we wish to find the value that is in the Φn-th position in the sorted list of data values. For example, the median is a special case where $\Phi = 0.5$. Without loss of generality we can assume that all the data values are distinct; ties can be broken using unique node IDs.

If we wished to verify the correctness of a proposed Φ-quantile q, we can perform a COUNT computation where each node s presents a value $a'_s = 1$ if its data value $a_s \leq q$ and presents $a'_s = 0$ otherwise. If q is the Φ-quantile, then the computed sum should be equal to Φn. Hence, we can use any insecure approximate Φ-quantile aggregation scheme to compute a proposed Φ-quantile, and then securely test to see if the result truly is within the approximation bounds of the Φ-quantile algorithm.

8.3 Generalizing the Algorithm for One-pass, One-output Functions

Given the algorithm for SUM as described in the prior sections, we can consider how the approach of commitment and distributed verification is applicable for general functions. This will allow us to better understand the capabilities and limitations of the core functionalities behind the algorithm.

8.3.1 The Generalized Algorithm

We consider the case where, in the aggregation tree, each node i computes the scalar result of some subfunction f_i over each of the scalar values reported by its children. As a first sketch, we show a possible approach to the problem given some very strong assumptions. We assume that each node knows the computations which are affected by its own inputs. Specifically, it knows all the intermediate subfunctions for all the nodes between itself and the root node (e.g., additions, multiplications, etc). It does not know the exact intermediate inputs to these computations (except for its own inputs); however it knows the exact *set* of possible (input-value-injection-equivalent) legal values each of these inputs can take. More precisely, the *legal set* of values for a function subtree is the set of values achievable by at the root of the subtree by setting legitimate values at the leaves.

Definition 9. *Let T be a function (sub)tree rooted at v_T where each vertex represents a (scalar) function computation over (scalar) inputs provided by its child vertices, the result of which is provided to the node's parent. The tree thus represents a function computation over the inputs at the leaf vertices; let this function be f_T. Let each of the k leaf vertices $l_1, \ldots, l_k$ be associated with some set of legal values $S_1, \ldots S_k$ respectively.*

If T is a single leaf vertex then let the ***legal set*** *of T just be the set of legal inputs at the vertex, i.e.,, S_1. Otherwise, we can define the* ***legal set*** *of T to be $\{x : x = f_T(s_1, \ldots, s_k)$ for $s_1 \in S_1, \ldots, s_k \in S_k\}$.*

Figure 8.2 shows an example of the kind of knowledge possessed by a given sensor node in the network.

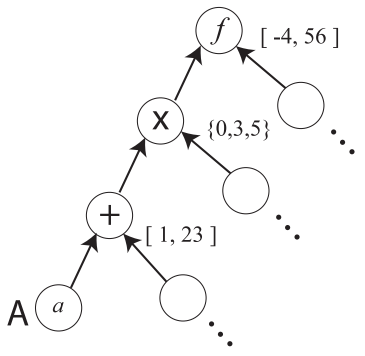


Figure 8.2: Example for general computations. Assume all ranges are in the reals. Node A knows that, in the computation, its input a is first summed with some value in $[1, 23]$ and then this result is multiplied with a value in $\{0, 3, 5\}$ and then finally the result of that is the first argument in the (known) function f where the other argument is a value in $[-4, 56]$.

We consider the straightforward method for applying the commitment-verification approach. We consider a simple extension of the sum hash tree to a more general *function hash tree*. The structural definition of the vertices remains identical as for the sum hash tree. Specifically, the vertex templates remain unchanged. For a leaf vertex containing the value v_i, we have:

$$\langle v_i, H[i, v_i] \rangle$$

For an internal vertex, we have:

$$\langle v_u, H[c_1 \| c_2 \| \cdots \| c_m] \rangle$$

In the above, $c_1, c_2, \ldots, c_m$ are the child vertices of the node. In the sum hash tree, v_u is the sum of the values in these child vertices, i.e., $v_u = v_{c_1} + v_{c_2} + \cdots + v_{c_m}$. For the function hash tree, we generalize this to $v_u = f_u(v_{c_1}, v_{c_2}, \ldots, v_{c_m})$, where f_u is the function that is evaluated at vertex u.

The protocol proceeds similarly to the process described in Section 4.2.1 (the sum algorithm without hash-tree restructuring operations). The leaf sensor nodes in the network topology report their respective hash tree vertices to their parents; internal sensor nodes then compute their respective internal hash tree vertices for the function hash tree based on the function f_u that they are respectively supposed to compute. The root of the function hash tree (containing the result of the network-wide computation) is passed on (or computed by) the base station; subsequently, nodes disseminate the information between their children such that each node receives its verification path between its leaf vertex and the root of the hash tree (i.e., all off-path vertices for their respective leaf nodes). Each individual node is then responsible for verifying the computations that occurred between itself and the root. It does this in the natural way by checking two conditions for each node on the path: first, all the inputs to the computation must lie within the set of a-priori-known legal possible values for the results of the intermediate computations resulting from each of the off-path subtrees; secondly, given the committed inputs and outputs, the computation must evaluate correctly at each node (i.e., the outputs are consistent with the computations evaluated on the inputs and results of each node on the path are correctly reported to that node's parent, and so on). Once the verification computations are complete, a network-wide confirmation phase is executed where each node returns a confirmation code based on the root vertex of the function hash tree that it computed (i.e., $M_i = \mathrm{MAC}_{K_i}(N||r)$). If all verifications passed correctly then the base station should receive the XOR of all the above messages; if any verification was incorrect then the aggregated confirmation code will not match what the base station expects and the base station should then reject the reported result. Note that, like the secure SUM algorithm of Section 8.2, there is no requirement for authenticated broadcast in this protocol.

8.3.2 Correctness of the General Algorithm

We can show that the algorithm provides input-value-injection equivalent integrity. Suppose every legitimate node correctly performs its verification. Then, each node traces a path of verified computation from itself to the root. Since the adversary must commit to exactly one commitment tree per query (i.e., the tree whose root vertex is reported to the base station), each of these paths are distinct and the union of these verified paths also forms a verified subgraph H which is also a tree, the structure of which is fixed by the (a-priori known) sequence of computations from each node to the root. The adversary is free to select the rest of the commitment tree G as long as it (a) contains H as a subgraph and (b) for any node v in $G - H$ that has a node in H as a parent, v must report a value that lies within the legal set for the (correct) subtree rooted at v. In this sense, v is constrained to behave correctly: since only values in the legal set for v can be injected without detection, and, in the correct function computation tree, v is the root of a subtree consisting of

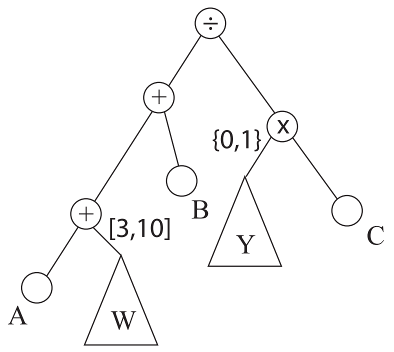


Figure 8.3: Example showing how the commitment tree ensures security.

inputs from only adversary-controlled nodes, any input that the adversary successfully injects at v is achievable simply reporting particular selected values for inputs contributed by nodes under its direct control. This implies input-value-injection-equivalent integrity.

A visual representation of this is shown in Fig 8.3. A, B, C are the only legitimate nodes. Since each of these nodes knows the computations between themselves and the root, the commitment tree must contain these correct paths to the root. The adversary is free to choose the structure of the subtrees W and Y subject to the respective constraints of $[3, 10]$ for the intermediate result from W and $\{0, 1\}$ for the intermediate result from Y; these constraints are assumed to be known to the legitimate nodes and checked in the verification process. Manipulation of the computations in W or Y cannot improve the adversary's ability to alter the computation result, since in the correct computation tree W and Y consist wholly of malicious nodes. Hence, by only altering the input data values of malicious nodes and not changing the correctness of the in-network computation, the adversary could already achieve any value in $[3, 10]$ for the intermediate result from W and likewise either 0 or 1 for the intermediate result from Y.

Note that in this case the congestion complexity of the protocol is determined by the structure of the computation tree and network topology; for the basic case where each sensor node computes a function over the results of its children, the congestion cost is $O(hd)$ where h is the height and d is the maximum degree of the network tree topology.

In the special case of computations where each subfunction is identical, associative, and commutative, the structural optimizations of Section 8.2 may be applied directly without change to balance the computation/hash tree. In this case, the congestion is identical to the case of the SUM algorithm, i.e., $O(\log n)$.

8.3.3 Making the General Algorithm Practical

The general algorithm of Section 8.3.1 requires two strong assumptions: each sensor node must know the set of computations between itself and the root, and it must also know the legal sets of all inputs to these computations. In this section we discuss several restrictions and methods to make these assumptions more practical.

Representation Size of Legal Sets and Functions

For the algorithm to be practical, the descriptions of both the functions and the legal sets must be relatively compact. This excludes arbitrary general functions from being practical in the framework. The functions computed at each node cannot be large multidimensional tables containing arbitrary values. In the most general case, each combination of legal input values at the leaves of a subtree results in a distinct, arbitrary result in the legal set of the subtree; this also results in a large table that grows exponentially with the number of leaf vertices in a subtree. Furthermore, the sensor nodes must have some authenticated way to access this information.

We consider a specific case of functions and values that retains much of the intuitive expressiveness of numeric computation on a tree while being a good practical fit for the framework described in Section 8.3.1. We consider the set of functions which will allow us to use simple range notation to describe the legal sets in the computation tree. First, assume that the legal set for each input value is some continuous range on the real numbers. In practice, this is impossible since it would require an infinite amount of precision in the representation of these numbers; we discuss these practical considerations later in Section 8.3.4.

In order for the legal sets of each sub-computation to also be a range on the reals, it is sufficient that each subfunction in the function tree be continuous. The inductive reasoning is based on the intermediate value theorem, i.e., for a continuous multivariate function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ defined for a continuous domain in $\mathbb{R}^n$, the image set of f must also be continuous. Since the legal sets of the leaf values are continuous range on $\mathbb{R}$, we can repeatedly apply the intermediate value theorem to show that the legal set of any subtree must also be a continuous range in $\mathbb{R}$.

We provide a constructive proof in the form of an algorithm for deriving the legal sets of each vertex in the computation tree. As the base case, let the legal set of each leaf vertex i in the computation tree be a continuous range S_i. Repeat the following until all the legal sets of all the vertices in the computation tree have been derived. For a vertex u computing a continuous function f_u with children $c_1, \ldots, c_k$ all with known legal sets $S_1, \ldots, S_k$, let a (resp. b) be the minimum (resp. maximum) value of f_u over the domain $S_1 \times S_2 \times \cdots \times S_k$. This maximum and minimum can be found by evaluating f_u at stationary and boundary points in the domain in the usual manner. Then, the legal set of u is $[a, b]$ since the intermediate value theorem ensures us that for any $c \in [a, b]$ we can find some $s_1 \in S_1, \ldots s_k \in S_k$ such that $f_u(s_1, \ldots, s_k) = c$.

The fact that all legal sets are continuous ranges ensures us that legal sets can be concisely communicated and stored. We assume that the subfunctions in the computation tree are also describable in a concise manner, i.e., we can consider simple algebraic functions such as short polynomials. As a further example, the implementer could exclude very long polynomials unless there is a concise rule which generates the coefficients of the polynomial.

Disseminating the Functions and Legal Sets

Given that both functions and legal sets can be stored and communicated in a concise way (e.g., a small constant number of bits), the problem remains of exactly how to bootstrap this information. If the topology and the computation is static, then this information can be stored statically on the sensor nodes; however if changes occur in the topology or the computation structure then these updates need to be efficiently and authentically disseminated from the central coordinating base station to the sensor nodes. The straightforward approach is to unicast this information to each individual sensor node from the base station. However, this incurs significant congestion since one of these notices must be sent each time the parameters of the problem changes (e.g., if a new, different computation is required, or if the legal sets of any input values change, or if the topology if the network changes).

An alternative to unicast is to use the information binding/credential dissemination primitive of Section 6. This assumes that the sensor nodes themselves know (or are instructed) when their own subfunctions or legal sets change, and the schedule for these changes is known by the base station. For example, if nodes change their computations as directed by the base station, or if the network topology changes due to additional node installations.

The protocol is a straightforward extension of the information binding/credential dissemination primitive of Section 6. We define a hash tree augmented with the function descriptions and legal sets of each vertex; this hash tree construction is identical to the sum hash tree construction, except that the various v_i in each vertex now instead describe the function computed at that vertex and the legal set of that vertex. The base station then uses its knowledge of the topology, functions and the legal sets in the network to construct a hash tree, the root of which is broadcast to the network. The network then collaborates to reconstruct the hash tree, disseminating verification paths to each sensor node. The verification path for each leaf vertex contains the hash tree vertices which describe the functions and legal sets of each vertex (and all its siblings) on the computation path from the vertex to the root, thus containing all the necessary information needed to enable the algorithm of Section 8.3.1. The congestion cost of this information dissemination phase is identical to the overall congestion of the generalized tree computation algorithm itself, i.e., $O(hd)$.

8.3.4 Accuracy of Representation

The input-value-injection-equivalent correctness of the general protocol holds under the assumption that the legal sets of inputs are continuous ranges in $\mathbb{R}$. In practice, when the accuracy of representation of numbers is limited, this may cause the strict definition of input-value-injection-equivalent integrity not to hold for the algorithm. For example, consider the "square" operation which takes a single value and returns the square. Let the legal set of inputs to this operation be $[0, 2]$. Then the legal set of this computation should be $[0, 4]$. The input-value-injection-equivalent correctness property tells us that for any output in $a \in [0, 4]$, we can choose some input $b \in [0, 2]$ such that $b^2 = a$; this is obviously true from the intermediate value theorem. Now suppose the input is represented using standard floating point notation with a fixed number of bits (while the output still retains infinite precision), and consider the output $a = 2$. Clearly, this is in the legal set, but we cannot find a precise floating point representation for $\sqrt{2}$ in the input. This seems to imply that the input-value-injection-equivalent property does not hold, because the function computation could potentially output 2, and have this result accepted, when no floating-point input value could have caused this precise output in practice.

We make the argument that allowing unachievable results within a legal set are not actual flaws in the practical usefulness of the protocol, but rather artifacts resulting from an over-strict definition of correctness. In the previous example, the "attack" was more based on the inability of the input to accurately represent $\sqrt{2}$ than the fact that the output of 2 in the computation of $f : [0, 2] \to [0, 4], f(x) = x^2$ is itself a vulnerability.

In light of this, we relax the definition of input-value-injection-equivalence to allow for correctness to be defined by policy rather than derived from specific function.

Definition 10. *Let f be a function computed on a set of inputs $x_1, \ldots, x_n$. In a computational system implementing f, the* ***legal set*** *of an input is the set of allowable values that it can take (as decided by the policy of the implementor or network supervisor). Let the legal sets of each input be $S_1, \ldots S_n$. A protocol implements f with* ***policy-compliant input-value-injection-equivalent integrity*** *if, by tampering with the aggregation process, an adversary is unable to induce the querier to accept any computation result which is not in the image set of f on the domain $S_1 \times \cdots S_n$.*

Essentially, this means that the protocol only accepts computation results y for which there exists some $x_1 \in S_1, \ldots, x_n \in S_n$ (i.e., some set of inputs which are "policy compliant"), such that $f(x_1, \ldots, x_n) = y$. This is a purely policy-based definition which allows y to be accepted even if some x_i cannot be represented in the computer system precisely. The idea is that we do not consider this to be an attack since we have explicitly allowed for this in our policy specification for S_i.

8.3.5 Continuous Function Computation in a Tree: Result

In summary, as the result of combining Sections 8.3.1, 8.3.3 and 8.3.3, we now have a protocol for providing policy-compliant input-value-injection-equivalent integrity for evaluating any computation tree comprising of continuous multivariate subfunctions (which have short descriptions) on the real numbers, (i.e., $f_i : \mathbb{R}^{d_i} \to \mathbb{R}$ for each computation tree vertex i). The legal sets for the inputs to this computation tree are intervals in $\mathbb{R}$. The overall congestion of the setup and the protocol itself is $O(hd)$. To the best of our knowledge, this is the first protocol ensuring computation integrity for arbitrary functions over a network, with a reasonable congestion bound.

8.4 Input Verification

The main limitations of the general techniques presented in Section 8.3.1 are due to the difficulty of accurately representing the legal sets of large subtrees. Hence, they are restricted to tree computations on subfunctions for which legal sets can be concisely represented, e.g., continuous functions. In the case of truly general functions, the legal set of a subtree can potentially be as large as the cross product of the legal sets of all the inputs to the subcomputation performed in a subtree. In such cases, rather than attempting to represent the legal set of a subcomputation, we can instead directly audit the inputs to the subcomputation; if all inputs are in their respective (small) legal sets, and the computation result is correct, then the result must also be legal.

8.4.1 Exhaustive Input Verification

The most straightforward method of performing this kind of verification for each node to replicate the computation of the *entire* network. To start, assume that each node contributes a single input value to the computation, and each internal node in the network computes a single subfunction over its input value and the intermediate results of all its children. Furthermore, assume that the legal sets of inputs of each node can be represented in a (small) constant amount of space.

The exhaustive input verification algorithm then proceeds as follows. We use the credential dissemination protocol of Section 6 to bind the legal sets and subfunction descriptions of every node in the network, and disseminate this information to all nodes in the network. Each node now has centrally-authenticated knowledge of the computations and legal sets in the entire network. Now we use the node-to-node signature algorithm of Section 7 to authentically disseminate the inputs of each node to every other node. Note that the signature functionality forces each node to commit to exactly one value of their input value for each run of the secure computation protocol; furthermore, since these values are signed, no malicious node can alter the input value of a legitimate node. This implies that all legitimate nodes have a consistent and authentic view of the structure of the computation in terms of subcomputations, inputs and legal sets of inputs. From this consistent picture it is a simple matter for each node to check that each input value is in its respective legal set, and then repeat the computations performed in the entire network to compute the final result R. Once the computation is complete, the nodes then use the network-wide confirmation protocol of Section 4.4 to confirm the value of R. The querier then checks the confirmation code to confirm that every node did indeed compute the same value R, and if so, it accepts R as correct and authentic.

The correctness of the protocol follows from the correctness of the subprotocols used; in effect we have put the pieces together in the natural way to construct an authenticated bulletin board of input values along with a correct copy of all computations and legal sets to every node. Each legitimate node can thus only compute a correct, consistent final result R or abort. The overheads of this exhaustive verification protocol are $O(n)$ communication and storage since every node needs to receive values from every other node and reconstruct several hash trees each built over n leaves. This makes it infeasible for networks where reducing node-to-node communications is the main goal. On the other hand, the congestion at the querier is only $O(1)$ since it only needs the result R, and the aggregated confirmation code confirming R, and a fixed number of runs of the subprotocols for credential dissemination and node-to-node signatures, each of which involves the communication of only a single hash value. This makes exhaustive input verification useful for networks where reducing querier-to-node communications is the primary metric.

8.4.2 Cluster-based Computation Verification

In Section 8.3.1, we noted that for a general one-pass function, the main complication in applying the hash-tree-based framework was in representing the legal sets of large subtrees. While we showed a promising approach for continuous real-valued multivariate functions, there is a still a large gap between this family of functions and truly general one-pass functions. In this section we show that if we can assume a fixed upper bound k on the number of malicious nodes, we can use input verification to ensure the authenticity of results of general one-pass functions.

We consider network computations where each node (internal or leaf) contributes exactly one input value to the computation. The idea is that the complexity of the legal sets of large subtrees arise directly from their size; a large subtree may contain a subfunction over a large number of inputs, and each combination of legal inputs may potentially produce a different legal result. However, in the hash-tree framework, we only have to worry about checking the legal sets of subtrees that are *wholly* malicious (see the analysis in Section 8.3.2). Hence, if we know that there are no more than k malicious nodes in the network, then we do not need to check the intermediate results of any subtree with $> k$ inputs. On the other hand, subtrees with $\leq k$ inputs are quite small and each of these at most k inputs can be checked for legality directly in a process similar to exhaustive verification (Section 8.4.1. Indeed, if we group these small subtrees into *clusters* of size at least $k + 1$, we can be assured that each of these clusters contains at least one legitimate node. By requiring all nodes in each cluster to verify each other, this ensures that every input is verified by at least one legitimate node and correctness follows in a similar argument to that provided in Section 8.3.2. In fact, this process can be considered a generalization of Section 8.4.1 parameterized with k being the number of assumed malicious nodes in the network.

The algorithm is described in more detail as follows. We assume that the adversary has compromised at most k nodes. We assume that the legal set of a single input is representable in size α where α can be a function of n; for ease of exposition we can assume α to be a constant (to be practical, α must be sublinear in n). We no longer restrict the range of each subfunction to be in $\mathbb{R}$ but simply require it to be representable in size β or less, where β is a function of n. Similarly to α, for the protocol to be practical, β should be sublinear in n.

Define the *size* of a subtree in the network computation as the number of *input values* for the subcomputation of the subtree; since each node must contribute exactly one input value, this is equivalently the number of nodes in the subtree rooted at the node responsibe for this particular subcomputation, in the network topology.

As a subprotocol, we use the information-dissemination functionality of Section 6 to authentically compute the size of each subtree in the network. The procedure where each internal node learns the size of its own subtree is identical to a SUM computation where each node contributes an input value of 1; since the trusted base station knows the exact topology of the network, it can then confirm this computation for every node in the network using the information dissemination functionality in Section 6. Let v be an internal node in the network topology, with parent node u.

The size of the subtree rooted at v is disseminated to all nodes that are descendants of u. At the end of this process, each node x in the topology knows the (authentic) size of all subtrees rooted at all nodes that are children of any node between x and the root (base station). This process causes at most $O(hd)$ congestion; the analysis is identical to that in Section 8.3.3.

Once the size of each subtree is known we can then decompose the entire network tree into "small" subtrees of size at most k which are joined via the internal nodes which are roots of the "large" subtrees of size $> k$. The original tree topology can thus be visualized as a "backbone" tree consisting of internal nodes which are the roots of the large ($> k$) subtrees; off this backbone hangs the small subtrees of k or less nodes. We can further break up the backbone tree into paths by assigning each parent node in the backbone to the leftmost path among its children. When this is done, the network is partitioned into distinct paths from which hang small subtrees of size k or less: we call each such path a *clustering-path*. Since every node knows the size of all subtrees off the path from itself to the root, every node in the network is also aware of the topology of its own clustering-path. An example partition is illustrated in Figure 8.4.

For each clustering-path, we then further group the small subtrees into *clusters* of size at least $k+1$. This ensures that each cluster contains at least one legitimate node. A straightforward way to assign clusters is to iterate across the subtrees left-to-right starting from the top of the clustering path, adding each subtree to the current cluster until it is larger than k, and starting a new cluster as needed. This creates clusters of size at most $2k$, except for the last cluster which contains any remainder subtrees and may be of size of up $3k$. This method is illustrated in Figure 8.5. For a given method of clustering the subtrees, it is straightforward for each node to determine the division of clusters in its own clustering-path.

Once the clusters have been determined, the protocol proceeds in two phases: first, exhaustive input verification is performed within each cluster. This process is identical to the exhaustive input verification of Section 8.4.1; each node in the cluster disseminates its input value and function computation to all other nodes in the same cluster. The credential dissemination method of Section 6 ensures that all nodes of the cluster can verify the inputs and computations performed in every other node in the cluster. Once exhaustive intra-cluster verification has been completed, nodes may then perform computation verification for their own respective computation paths from their input

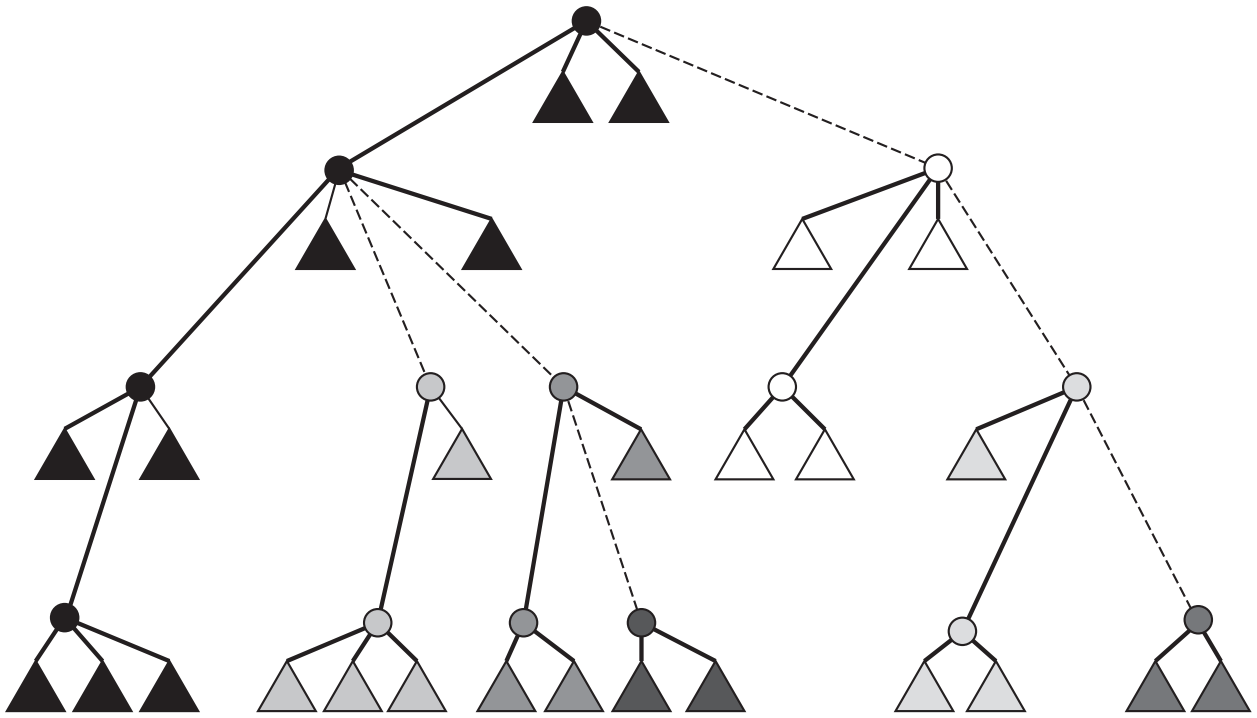


Figure 8.4: Partitioning the network into clustering-paths, or paths of small subtrees. Each triangle is a subtree of size k or less; distinct cluster-paths in the partition are highlighted with different colors.

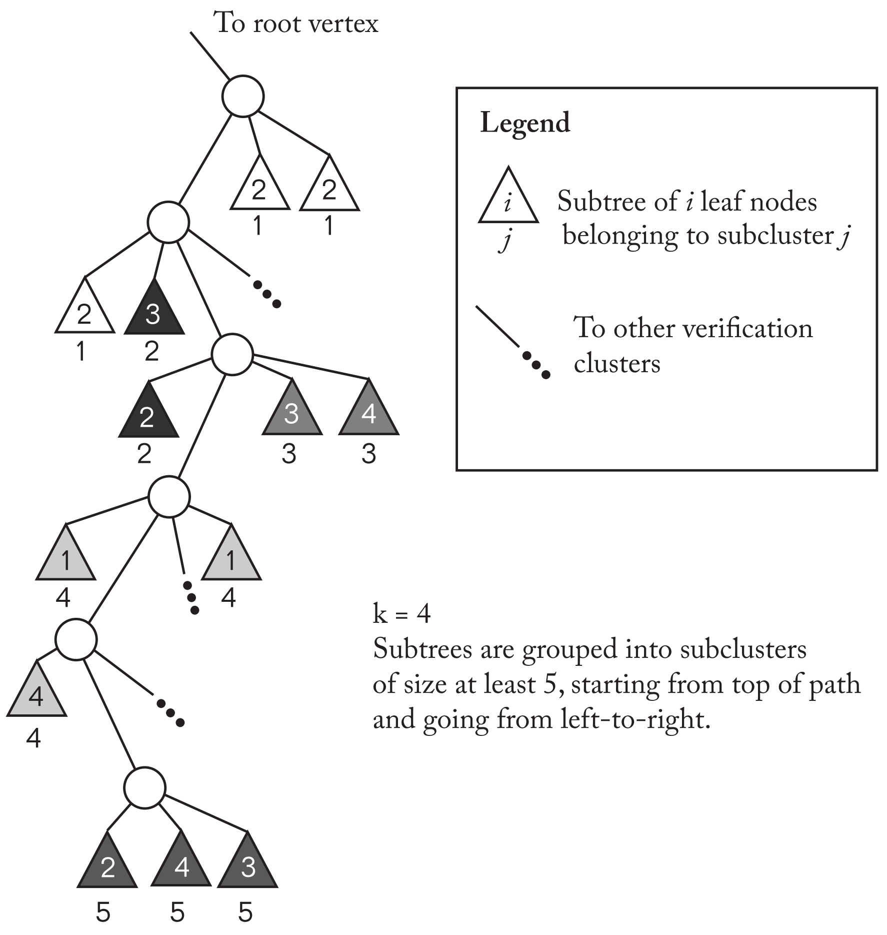

To root vertex
Legend
Subtree of *i* leaf nodes belonging to subcluster *j*
To other verification clusters
k = 4
Subtrees are grouped into subclusters of size at least 5, starting from top of path and going from left-to-right.

Figure 8.5: Partitioning a clustering-path into clusters of size at least $k+1$. In this example, $k = 4$

up to the root of the tree containing the final computation result. This process is similar to that in Section 8.3.1, except that legal sets of intermediate results do not need to be checked; each node only needs to ensure that all the intermediate computations between its own input and the final result are performed correctly. Finally, if all nodes were successful in their respective verifications, a network-wide acknowledgement is used to confirm this to the base station.

The correctness of the cluster-based scheme follow directly from the correctness of exhaustive verification (Section 8.4.1) combined with computation verification (Section 8.3.1). Specifically, as noted in Section 8.3.2, given computation verification is performed successfully, the only way for an adversary to cheat the computation is to forge an intermediate result from some subtree T consisting of wholly malicious nodes. By assumption, this subtree T does not have more than k nodes, and is thus contained in one of the subtrees in the clustering-paths decomposition; in particular all the nodes of T are in some cluster of size $> k$. This means that the computations and inputs of T must be exhaustively verified by some legitimate node; hence the adversary is unable to cheat in T, and the security property holds.

The cluster-based verification protocol induces additional overhead for the exhaustive input verification of each cluster; it is not hard to see that each edge in the network topology carries spans at most one cluster and is thus responsible for at most the traffic of one cluster in the exhaustive input verification phase. Since the largest cluster is of size at most $3k$, this yields an edge congestion of $O(3\alpha k)$ for the exhaustive input verification phase. Since each node is adjacent to at most $d+1$ edges (where d is the maximum fan-out or number of children for each internal node), this yields a node congestion of $O(3\alpha k(d+1))$. Combining this with the $O(\beta hd)$ overhead of verifying the results along the computation path, the final congestion overhead is $O(3\alpha k(d+1)+\beta hd)$. While this congestion is unattractive for networks that have to tolerate large numbers of compromised nodes, this protocol is nonetheless useful because it is the only known efficient protocol for ensuring the correct computation of *any* one-pass function.

Chapter 9

Rate-Limiting Using the *HT* Framework

We consider the problem of *traffic volume attacks*, an attack in wireless networks where malicious nodes inject spurious traffic to drain the resources of legitimate nodes in the network. More insidiously, the adversary could send large volumes of data down specific links or paths, crowding out legitimate data.

In this section, we describe a countermeasure against a packet-injection attacker by leveraging the *HT* framework. Our approach is to perform communications in epochs. Within each epoch, each node is allotted a fixed maximum rate of message origination. This is the upper bound on the amount of data origination that we wish to enforce at each node in the network, and is selected by the operator of the network as an input to the algorithm (note that different nodes may have different bounds). When each epoch is complete, the base station then audits the total amount of communications originated by each node in the network. Any node that exceeds its allotted rate will be unable to pass the audit and will have to pay a penalty; typically this means some set of malicious edges or nodes will be revoked. The mechanism for the audit will be similar to the SUM secure aggregation operation of Section 8.2.

9.1 Problem Definition

Informally, we wish to detect if the adversary has caused a large amount of traffic to be forwarded by legitimate nodes in the network. We define the problem formally as follows.

We associate with each node a, a maximum data origination rate R_a which represents its per-epoch origination bandwith allotment. In a given epoch, each link e in the topology carries some amount of traffic destined for the base station. Let b_e denote the total amount of traffic that traversed link e within the epoch. Visualising the set of b_e as weights on edges, the weight assignment for a given epoch is called the *communication pattern* of the epoch:

Definition 11. *A* ***communication pattern*** *is an assignment of some traffic volume b_e for each link e in the network.*

We assume that all traffic in the sensor network involves flows from individual nodes to the base station (i.e., all traffic only travels towards the base station). A communication pattern where every node conforms to its rate limits is called "well-behaved", defined as follows:

Definition 12. *A* ***well-behaved communication pattern*** *is a communication pattern in which, for every node a in the network, the total amount of outgoing traffic is no more than the sum of the total amount of incoming traffic and the node's maximum data origination rate R_a.*

Also note that the definition allows for nodes to have less outgoing traffic than incoming traffic, reflecting some packet loss at that node. Intuitively, we would like to allow only well-behaved communication patterns in each epoch; however such a security definition is too strict. For example, if two malicious neighboring nodes send an illegally large volume of traffic to each other, this traffic is invisible to the legitimate nodes in the network and yet the communication pattern would violate the "well-behaved" constraint. Hence we offer a more functional definition in terms of only what is visible to the legitimate nodes.

Definition 13. *The* ***view*** *of the legitimate nodes is an assignment of some traffic volume b_e for each link e in the network which is adjacent to a legitimate node.*

Since the view only assigns weights to a subset of the links in the network, we can consider if it is in fact possible to assign weights to the remaining links such that no node in the network exceeds its allotted rate. We formalise this approach as follows:

Definition 14. *For a given view W, let F be any communication pattern which assigns the same weight as W to all edges in W. Then F is a* ***completion*** *of W. A* ***permissible*** *view W is one where there exists some well-behaved communication pattern which is a completion for W.*

For a given epoch, if the view is permissible, then the existence of the well-behaved completion implies that exists some way for this view to have occured legitimately. In other words, the adversary could have achieved this amount of traffic on the legitimate nodes simply by obeying the rate limits on the nodes under its control. On the other hand, if the view is not permissible, then any communication pattern C which results in the view V must not be well-behaved: in particular, there must always be at least one node in the network which exceeds its maximum allowed data origination rate. Hence, we can say that the network is under data origination attack if and only if the view of the legitimate nodes is not permissible.

9.2 Naive Approach: Direct Audit

Based on the problem definition, the most direct method for determining if the view of the legitimate nodes is permissible is to check the amount of traffic across every link in the network. The base station can then directly check if the reported traffic weights reflect a well-behaved communication pattern.

The details of this direct audit process is as follows. At the end of each epoch, the base station broadcasts an end-of-epoch message containing the epoch identifier N_e. Every node a then composes a *traffic report* D_a of the total communications it has performed over the epoch on each of its incident links. The message is encrypted and authenticated using the secret key K_a that a shares with the base station.

The data portion of the report of node a is composed as follows, where b_{c_i} is the amount of traffic received from child c_i, and b_{p_i} is the amount of traffic sent to parent p_i, and M_{c_i} is the report message received from child c_i.

$$\begin{aligned} D_a &= (N_e, ID_a, \text{Incoming:}(c_1, b_{c_1}), \ldots, (c_k, b_{c_k}), \\ &\qquad \text{Outgoing:}, (p_1, b_{p_1}), \ldots, (p_q, b_{p_q})) \end{aligned}$$

This message is encrypted and authenticated using the unique secret key shared between the node and the base station.

Once all the traffic reports are received, the base station can then take the union of the reports to find the communication pattern. To resolve conflicting reports on the amount of traffic on a given link, we note that if link-layer encryption and authentication is implemented, an adversary cannot inject messages between legitimate nodes. Hence, for any link e from node c to node p, the downstream node must have received no more messages than was sent by the upstream node. The base station checks that the respective traffic reports reflect this fact, then assigns the traffic weight reported by p to the link. When each edge has been labelled with a traffic weight, the base station checks the well-behavedness property of the overall communication pattern C': specifically, for each node i, the amount of outgoing traffic from i must be no more than its maximum origination rate R_i more than the total amount of incoming traffic to i.

Theorem 15. *If C' is well-behaved, then the view of the legitimate nodes is permissible.*

Essentially, C' assigns the same traffic weight as the legitimate nodes to all edges in the legitimate view except it may assign smaller weights on outgoing edges from legitimate nodes. This discrepancy does not help the adversary exceed its rate.

If each of these checks pass, then the base station determines there was no packet-origination attack in this epoch. Otherwise the base station concludes that an adversary is present and takes appropriate remedial action as described in Section 9.2.1.

If reports from all the nodes are received by the base station, correctness is immediate. Suppose the data received by the base station passes all the checks, i.e., the views of each node is mutually consistent with that of the other nodes, and the overall communication pattern formed from the union of these views is well-behaved. Since the adversary is unable to falsify reports from legitimate nodes, the final communication pattern must be consistent with (i.e., it is a completion of) the view of the legitimate nodes. Since the final communication pattern is well-behaved, by definition the view of the legitimate nodes must be permissible.

By the contrapositive of this argument, if an adversary induces a communication pattern causing a non-permissible view at the legitimate nodes, then regardless of what the malicious nodes report, the final communication pattern assembled by the base station is not well-behaved and so will not pass the check.

9.2.1 Locating an Adversary

In the case of a failed audit, we would like to be able to locate at least one link or one node under adversarial control and revoke it, thus applying some kind of penalty to the adversary for the attack. Thus, if the adversary persists in performing such attacks, eventually all its points of presence in the network will be located and revoked.

The detection mechanism of Section 9.2 ensures that if an adversary performs an attack in an epoch, it cannot pass the requisite checks. However, rather than incriminate itself by revealing that one of its nodes is not obeying its rate limit, the adversary can simply refuse to respond to the audit. More importantly, it can choose to selectively drop the reports of various legitimate nodes, thus denying the base station any information which can be used to locate the adversary's point of presence in the network.

We modify the protocol to prevent arbitrary dropping of traffic reports. First, we select a subset of the network's edges that form a spanning tree directed towards the root (the base station). We assume that certain relevant details of the topology of the spanning tree are known to the nodes; in particular, each node is aware of its set of children in the spanning tree, and also the size of the subtrees associated with each child. Since this spanning tree is largely static, such details could be sent from the base station to each node on the initialisation of the network, and changes can be disseminated efficiently using authenticated broadcast to the affected subtrees.

When a node a receives the end-of-epoch message, it waits to receive traffic report messages from its children. The receiving node a checks the traffic reports for size (the report from a child c_i should be at most proportional in size to the number of nodes in subtree rooted at the c_i). If the traffic report is too large, a refuses to accept the message. This prevents the adversary from using the audit phase as an opportunity to flood the network with a large volume of traffic. Once a has received the reports from all its children, it then concatenates all the received information with its own traffic report into a single message, which secured with the MAC using the key K_a shared between a and the base station. This message is then sent to the parent of a which then repeats the process. Formally, this process is described as follows, where $c'_i, \ldots, c'_r$ are the children of a in the spanning tree and D_a is the traffic report of node a:

$$M'_a = D_a || M_{c'_1} || \cdots || M_{c'_r}, \quad M_a = M'_a || \mathrm{MAC}_{K_a}(M'_a)$$

This method of repeated encapsulation ensures that a malicious adversary cannot selectively drop the reports of nodes at arbitrary points in the network. The adversary can only drop the messages of an entire subtree in the spanning tree; furthermore it can only do this if it controls the link between the root of the subtree and its parent (for example, one of the two endpoints of the link may be malicious). Hence, by observing which subtrees did not successfully transmit their traffic reports, the base station can determine which communication links are under adversary control. The base station may then revoke these links in a subsequent epoch.

Specifically, in a given epoch where the audit fails, the base station revokes any link e between parent node p and child node c if (a) p claims to have received more messages than c claims to have sent out or (b) e was a spanning tree edge and no traffic report was received from c. We assume that during the audit phase we can allow for a sufficiently large number of retransmission attempts such that a link between two legitimate nodes should eventually succeed in delivering the message in the absence of adversarial interference.

Links may be revoked by having the base station unicast specific revocation instructions to the affected nodes; these messages will be encrypted and authenticated using the key shared between the respective nodes and the base station. The revocation instructions will also have to include updates to the structure of the new spanning tree if a spanning tree edge was revoked.

9.2.2 Bounds on Adversarial Injection Capabilities

We have established that if the network passes a full audit, then the adversary cannot have performed an attack in that epoch. It remains to show that if the adversary decides to take an opportunity to perform an attack (and thus deliberately fail the audit), the total amount of damage can be bounded.

We label each communication link in the network with a *capacity*, which represents the maximum amount of data that can be sent over that link in an epoch. Each parent node is aware of the capacity of each of its incoming links; any data in excess of this rate is discarded by the parent.

Based on this capacity, a set of adversary nodes can inject at most some volume of data F into the network. Each time the adversary fails the audit, it can inject at most F data but loses one of the edges under its influence. Hence if E_m is the set of all edges incident to any malicious node then the total amount of extra data that can be injected in a series of failed audits is at most $|E_m|F$.

9.3 Distributed Detection

The detection algorithm of Section 9.2 allows us to determine if the view of the legitimate nodes is not permissible, but it requires the transmission of n traffic reports to the base station causing $O(n)$ congestion at the nodes closest to the base station even when no traffic injection attack is taking place. We describe a method for detecting non-permissible views with $O(dn_a)$ congestion and memory overhead (where d is the maximum indegree in the network and n_a is the maximum number of ancestors for any node).

Our algorithm is motivated by the observation that the problem of auditing incoming and outgoing traffic is closely related to the problem of secure SUM aggregation. Hence, we can use the HT framework of Chapter 4.2 to perform distributed audit for rate monitoring.

Two important complications arise making the adaptation non-trivial. First, in the SUM aggregation problem, only the final sum computed at the base station needs to be correct; however for the rate monitoring problem the subsums at every intermediate node must also be correct. We address this in a similar way to the generalized secure aggregation algorithm of Section 8.3: we use the credential dissemination function of Chapter 6 to make the capacity of the links incident to each node (correctly) known to all its children; effectively allowing us to perform the rate consistency check at every node instead of just at the base station. The second complication is that, in the SUM aggregation problem, aggregation is performed with respect to a spanning tree; on the other hand rate monitoring needs to consider the topology of the entire DAG network since traffic can flow along any edge. To address this problem we require each node to check the consistency of the operations at each one of its parents rather than just a single path. Hence, we need to modify the *HT* protocol to not build a hash tree but rather a hash DAG (directed acyclic graph).

9.3.1 Assumptions

To provide the properties that are necessary for the correctness of the distributed audit scheme, we make two additional assumptions, as follows:

Every node knows the full details of the topology between itself and the base station. Specifically, for a node x, let A_x denote the set of nodes on any path between x and the base station. We will call A_x the set of all ancestors of x. For any node $a \in A_x$, the node x knows every edge incident to a as well as its associated capacity. Furthermore x knows the rate limit R_a of a. This information can be distributed in a direct application of the credential dissemination functionality described in Chapter 6. The credentials of various nodes along a single path share a common verification paths in the hash DAG used by the credential dissemination functionality. Hence, the total amount of congestion caused by this phase is $O(dn_a)$ (where d is the maximum indegree in the network and n_a is the maximum number of ancestors for any node).

Capacities are assigned conservatively. In a distributed audit setting, malicious nodes without legitimate descendants are never audited. To prevent them from exceeding their rate limits, we need to directly restrict their ability to inject data by not providing them with edges of large capacity into the network (this is similar to the generalization for secure function computation presented in Section 8.3). Specifically, we require that, for each node, the total capacity on its outgoing edges should be no more than the total capacity on its incoming edges, plus the origination rate of the node itself. For example, if a node has two incoming edges with capacities of 2 and 3, and it has an allowed origination rate of 1, then its total outgoing capacity should be no more than 2+3+1=6.

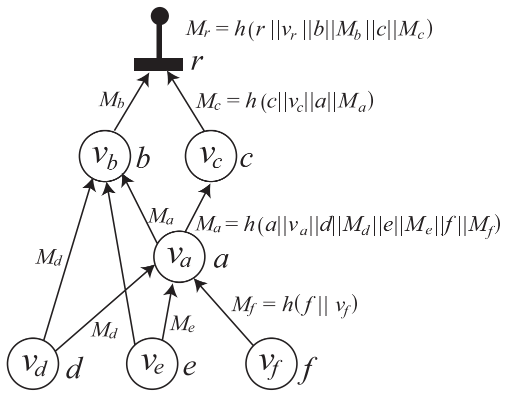

$M_r = h(r||v_r||b||M_b||c||M_c)$
r
M_b
$M_c = h(c||v_c||a||M_a)$
v_b b
v_c c
M_a
$M_a = h(a||v_a||d||M_d||e||M_e||f||M_f)$
M_d
v_a a
$M_f = h(f||v_f)$
M_d
M_e
v_d d
v_e e
v_f f

Figure 9.1: Topological Hash DAG

9.3.2 Algorithm Overview

The distributed audit algorithm is related to the secure SUM algorithm of Section 8.2. Specifically, it proceeds in three phases: commitment, verification and confirmation. We describe each in turn.

Commitment phase. In the first phase of the algorithm, each node commits to the details of its own traffic report. The commitment structure is constructed such that each node can only commit to a single traffic report; hence a malicious node is unable to selectively reveal different traffic reports to different nodes in the hope of evading detection.

The commitment structure is a *topological hash DAG*, which is a natural generalisation of a *hash tree*. Consider a network topology in the form of a directed acyclic graph (DAG) with a single sink (i.e., the base station) which we call the root. Suppose each node x is associated with some (not necessarily distinct) label v_x (in our algorithm, the label v_x represents the traffic report of node x). An example of this network topology is shown in Figure 9.1.

It is straightforward to modify the *HT* protocol to construct a DAG instead of a tree. The topologicial hash DAG is constructed recursively. An example is shown in Figure 9.1. First the nodes in the network which have no children set $M_x = H[x||v_x]$ and report their respective M_x to all of their parents. Internal nodes then compute hash values recursively. In the following formula,

a is the internal node computing its value, and it has children $c_1, \ldots, c_k$ in order of increasing ID.

$$M_a = H[a||v_a||c_1||M_{c_1}|| \cdots ||c_k||M_{c_k}]$$

The root value M_r is computed in the same way. Considering Figure 9.1, note that the structure of the hash DAG follows the network topology exactly. For a given M_r, it is computationally infeasible for an adversary to find another topology or assignment of values to node labels that will create a hash DAG with the same M_r.

M_r is sent by the base station to the entire network using authenticated broadcast. Nodes then relay the necessary information for each other to verify their presence in the topological hash DAG. Specifically, each node a sends all the hash inputs needed for the computation of M_a to all of its descendants. Each node thus receives the inputs to the hash computation for each one of its ancestors. The dissemination process causes $O(N_a)$ congestion on the links of the network where N_a is the number of ancestor nodes for the node with the greatest number of ancestors.

Note that an adversary is unable to perform additional data injection while the hash inputs are being disseminated. This is because each node, knowing the exact downstream topology of itself and its parent nodes, is able to calculate the exact size of the data it is expecting to receive. Any additional data is simply dropped.

Verification Phase. Once a node x has received the inputs to the hash computations of each of its ancestors, it may perform verification to ensure that none of its ancestors exceeded their respective rates. First, node x repeats the relevant computations to verify that the received values do indeed result in the correct M_r after repeated hashing, and the reconstructed structure of the downstream traffic reports exactly matches the known downstream topology of x. Once x is certain that the received data was indeed the same values that resulted in the commitment M_r, it then checks the consistency of the traffic reports of each ancestor node a. Specifically, x checks that (a) no node exceeded the capacity on its links; (b) for any link $e = (c \rightarrow p)$, the downstream node p does not report that more packets were received than the upstream node c claimed to have sent; and (c) none of the traffic reports show an ancestor node originated more than its allowed rate of traffic in an epoch. Node x only passes the verification process if all of the above conditions are met.

Confirmation phase. In the final phase all nodes proceed to notify the base station of their respective successes in verification. We use the network-wide acknowledgement functionality of Section 4.4. If the acknowledgement is received successfully, then all nodes have obeyed their traffic limits; if not, then an adversary is present somewhere in the network. The base station then runs the detection protocol of Section 9.2.1 to identify some edge of the adversary to revoke.

9.3.3 Correctness

We show that, if all the legitimate nodes successfully perform distributed verification, then the view of the legitimate nodes must be permissible. For the remainder of this subsection, assume that every legitimate node has successfully performed distributed verification; our goal is to exhibit a well-behaved communication pattern as a completion to the view of the legitimate nodes.

Lemma 16. *Let a be any common ancestor of any two legitimate nodes i, j. Then it must provide the same traffic report to both i and j.*

Proof. The constructed hash DAG commits to exactly one traffic report per vertex position. Since each node is associated with a given fixed hash tree position which corresponds to its position in the network topology (via the credential dissemination protocol), the adversary can only create one traffic report per node. This property is related to the non-repudiation property of node-to-node signatures (see Chapter 7). □

Lemma 17. *Consider a DAG network where edge capacities are assigned conservatively, Consider a set of nodes $U = \{v_1, \ldots, v_k\}$. Let $S(U)$ be the set of U and all descendant vertices of any vertex in U. Then, for any assignment of (traffic) weights on the outgoing edges of $S(U)$ that respect the capacities of those edges, there exists an assignment of traffic weights on the internal edges of $S(U)$ which respects edge capacities such that no node in $S(U)$ exceeds its allotted origination rate limit.*

Proof. (Sketch) This is a restatement of the principle of conservative capacity assignment. It is easy to see that if capacities are assigned conservatively then the outgoing rate limit at a given internal node v is no more than the sum of the origination rate limits of all descendants of v plus the origination rate limit of v itself. Hence, if v is obeying its rate limit, then there exists an legal assignment of origination rates to v and all nodes that are descendants of v. The lemma generalizes this to a arbitrary sets of nodes v. □

Theorem 18. *If all legitimate nodes pass the distributed verification, then the view of the legitimate nodes must be permissible.*

Proof. (Sketch) Let the set of all legitimate nodes be L. Let $A(L)$ be the set which includes L and all ancestors of the nodes in L. By Lemma 16, every node in $A(L)$ reports the same traffic report to all its legitimate descendants. All the origination rates reported by any node in $A(L)$ are within the allotted bounds since some legitimate node must have checked it. As for the remaining nodes, Lemma 17 ensures they do not exceed their respective rates. This implies a well-behaved communication pattern that is a completion of the legitimate view. □

Theorem 19. *The distributed audit protocol induces $O(dn_a)$ congestion and requires $O(dn_a)$ memory overhead on sensor nodes, where d is the maximum indegree in the network and n_a is the maximum number of ancestors of any node.*

Proof. The $O(dn_a)$ memory overhead arises from the necessity of each node knowing its entire downstream topology. For the congestion bound, the commitment phase causes $O(d)$ congestion while the verification phase causes $O(dn_a)$ congestion due to the fact that the hash computations from each ancestor are of size $O(d)$ and $O(n_a)$ of them must reach the nodes with the greatest number of ancestors. The final confirmation phase causes $O(1)$ congestion. □

Chapter 10

Low Latency Broadcast Authentication

Topology knowledge can not only yield efficient new protocols for various applications, but can also be applied to enable speedups of these protocols for known special-case topologies. In this chapter we examine latency optimizations for the broadcast authentication protocol of Section 5 as well as for a family of schemes known as hash-chain-keyed broadcast authentication protocols.

Latency is of particular importance in delay-sensitive applications such as alarms, node revocation, and time synchronization. We examine specific optimizations for latency and communication congestion in two specific topologies: linear and fully-connected. The linear topology occurs frequently in wireless networks both directly in such examples as networks deployed along corridors and roadways and indirectly as a multi-hop routing path between a sender and receiver. The fully connected topology occurs less frequently in our standard application scenario of sensor networks, but it often occurs in Internet-scale distributed systems such as grid computing and peer-to-peer networks, where any node can communicate (via the Internet) with any other node. We also provide a latency lower bound for broadcast authentication in trees without time synchronization.

10.1 Problem Definition

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 entire 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 cryptography.

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 arbitrary unknown subset of the receivers may be arbitrarily malicious, i.e., they are colluding byzantine nodes under control of a single adversary. 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 actions of the adversary). For brevity, we also ignore replay attacks in the discussion; we use the standard assumption that each message contains the necessary sequence numbers or time stamps to make replay attacks ineffective.

We consider two metrics 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 time between when the sender s initiates the protocol (by sending the first message) and the latest time at which any receiver node authenticates 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

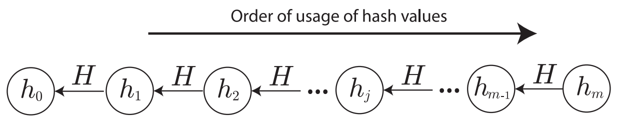


Figure 10.1: Hash chain

messages or failing to respond as expected. To estimate latency, we divide time into *steps*. In each time step, every node is allowed to exchange information (both ways) from only 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 can only send a small amount of information in each time step. Time steps are simply an estimate of latency and not a synchronization assumption, i.e., nodes are not globally synchronized to uniform time step boundaries.

Conceptually, it can be easier to think of all nodes performing the protocol in lock-step, with all nodes switching connection partners simultaneously at each time step boundary: for clarity, this is how we will describe our protocols; however no global synchronization is necessary for these protocols. Nodes can attempt to establish connections as soon as they are able to do so in the protocol; if their partner is not ready (e.g., it is busy with an existing connection, or waiting on a dependency), then this connection attempt will fail or block until the partner node is ready to connect. In general the notion of "time steps" should be considered more as a measure of latency than a form of protocol synchronization.

10.1.1 Background: Hash Chain Broadcast Authentication

Hash chains are used for broadcast authentication in such protocols as the Guy Fawkes schemes (an example of which is the Chained Stream Authentication algorithm or CSA) [4] and in TESLA [66]. The basic structure of these algorithms are similar and can be sketched as follows.

On initialization, the sender generates a hash chain of some predetermined length k by randomly selecting a secret seed value h_k and then iterating a pre-image resistant hash function H to generate a chain of k values such that $h_{i-1} = H[h_i]$ for $i = k-1, k-2, \ldots, 0$. Figure 10.1 shows a hash chain. Receivers are loaded with the final hash chain value h_0 during initialization.

To authenticate the ith message m_i, the sender broadcasts $(m_i, \mathrm{MAC}_{h_i}(m_i))$. Once all receivers

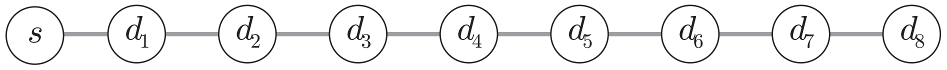


Figure 10.2: Linear Topology with 8 receivers

have received $\text{MAC}_{h_i}(m_i)$, the sender broadcasts h_i. Receivers can check that $H[h_i] = h_{i-1}$ and also that the MAC is correct given m_i and h_i. If so, they accept the message.

The hash chain value h_i is released only after all receivers have stopped accepting candidates for $\text{MAC}_{h_i}(m_i)$. The protocol thus temporally separates MAC generation from MAC verification by releasing h_i for verification only after it becomes useless for MAC generation. To achieve this separation, TESLA uses loose time synchronization which implies additional assumptions and protocol overheads, while CSA/Guy Fawkes uses explicit acknowledgments which are expensive in terms of communication overhead. In Section 10.2.2 we describe several optimizations to speed up and to reduce the communication overhead of explicit acknowledgments.

10.2 Linear Topology

We now examine optimizations of the hash tree and hash chain protocols for the linear network topology. In the linear topology, 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 nth receiver is n hops away from the sender, any protocol for disseminating M must take at least n time steps. Figure 10.2 shows an example of the linear topology for 8 receivers.

There are numerous applications of broadcast authentication in a linear topology. For example, wireless networks deployed in a linear environment naturally assume a linear topology. Examples of such networks include sensor and wireless networks deployed along corridors, roadways, tunnels and pipelines. In such environments, all applications involving broadcast or multicast would benefit from our optimizations. These may include data query messages, control messages (e.g. turn on all the lights, or temporarily shut down all sensors), and coordination messages (e.g., disseminate new global sleep/wake schedules, or open the network to allow new nodes to join).

General sensor networks are often logically arranged in a tree with the base station at the root [54]. Paths naturally occur in tree topologies as the subgraph of all the nodes between the root and any leaf. Hence our algorithms are useful for securing broadcast communications to all nodes along any root-leaf path. A specific example of such communications include node join/leave notifications. In the event of a new node joining a tree topology as a leaf, all its ancestors up to the root may need to be authentically notified of the change in membership of their respective subtrees; the notification would take the form of a broadcast along a linear topology from the base station to all affected ancestors. A similar argument applies for nodes leaving the topology and reorganizations of the tree due to node redeployments.

More generally, the linear topology also occurs as a *routing path* between any two endpoints. Broadcast authentication on the routing path can then be used for functions related to route maintenance and data transport. For example, endpoints may wish to inform forwarders that they are still actively using this routing path, or to request a certain traffic rate on the route. Endpoints may also send out authenticated packet loss statistics to help forwarders determine if they are responsible for packet dropping bottlenecks. In scenarios where a group key is shared between all nodes on the path (e.g., for the purposes of preventing injection of packets from nodes outside of the forwarding path), authenticated broadcast may be used to maintain consistency in sequence numbers or to update the path key periodically to prevent key overuse.

10.2.1 Hash Tree Based Schemes

In the basic hash tree based scheme of Chapter 5, the protocol proceeds in three passes. The first pass consists of a broadcast message front expanding outwards from the sender; this disseminates the message M and the hash tree root r to all nodes. The second pass is a convergecast message front starting at the leaves of the network topology and converging towards the sender; this allows internal nodes in the topology to compute internal hash tree vertices. The third front is another outgoing message front from the sender and facilitates the dissemination of the computed internal hash tree vertices to the rest of the network for the reconstruction of each node's respective authentication paths.

A naive implementation of the basic protocol on a linear topology will result in a latency of $3n$ as each of the three passes takes n hops to reach the farthest node from the sender and vice-versa. The communication congestion of the basic protocol is $O(\log n)$ hash values. We show an optimization that reduces the latency to $2n$ with $\lceil \log n \rceil + 1$ congestion and a further optimization that with optimal n latency with $2\lceil \log n \rceil + 1$ congestion overhead.

Optimization for $2n$ latency

The first optimization is based on the following observation: as soon as (M, r) is received by a node i, the value of $\mathrm{PRF}_{K_i}(M)$ can immediately be released to the (untrusted) network to facilitate the derivation of authentication paths for other nodes. In other words, each node can individually start to reconstruct the hash tree as soon as it has received the payload message M and does not have to wait for other nodes to also receive M. This allows us to "piggy-back" the dissemination of (M, r) onto the two passes that reconstruct the hash tree, thus reducing latency to $2n$.

The optimization is as follows. For a linear topology of n receivers numbered $1, \ldots, n$ with node 1 being closest to the sender, the hash tree is constructed with leaf vertices, from left to right, $(v_1, \ldots, v_n)$, with $v_i = \mathrm{PRF}_{K_i}(M)$ where K_i is the key shared between the sender and the ith receiver. We define the *left off-path vertices* L_i for a hash tree leaf v_i as the set of all sibling vertices that are to the left of the path from i to the root vertex r; similarly the *right off-path vertices* R_i of v_i is the set of all sibling vertices that are to the right of the path. For example, in Figure 10.3, the left off-path vertices of vertex 4 are vertices 3 and 9; similarly, there is one right off-path vertex, 14. Given v_i, L_i and R_i, the authentication path from v_i to the root can be computed.

We note that L_{i+1} can be easily computed from L_i and v_i. This is done in the natural way by adding v_i to L_i and repeatedly computing hash tree vertices as far up the authentication path of v_i as possible. This yields the deepest left-child in the authentication path of v_i. For example, in Figure 4, given L_4 (vertices 3,9) and vertex 4, we can first compute vertex 10 and then vertex 13, which is the first left-child in the authentication path $(4, 10, 13, 15)$. Vertex 13 is exactly the left off-path vertex of vertex 5. More generally, let h be the height of the hash tree. If L_i has no vertex at maximum depth h, then we can just add v_i to L_i and obtain L_{i+1}. Otherwise, let k be the deepest level of the hash tree which has no vertices in L_i. Then L_i has vertices at all depths from $k+1$ to h. Let them be $u_{k+1}, u_{k+2}, \ldots, u_h$ respectively. Now L_{i+1} has a vertex u_k at level k: this vertex is root of the largest subtree that contains v_i as the right-most child, i.e., it is the deepest left-child in the authentication path of v_i. It can be computed as:

$$u_k = H[u_{k+1} \| H[u_{k+2} \| \cdots H[u_{h-1} \| H[u_h \| v_i]] \cdots]]$$

All vertices of L_{i+1} at depths less than k are exactly the same as those at the same depth in L_i. A symmetrical reasoning applies to computing R_{i-1} from R_i and v_i. The algorithm thus completes in two passes: in the outgoing pass, nodes compute left off-path vertices for their downstream neighbors; in the incoming pass, nodes compute right off-path vertices for their upstream neighbors. The message and authenticator (M, r) are piggybacked onto the outgoing pass. The algorithm is shown in Algorithm 4; an example is illustrated in Figure 10.3.

Algorithm 4 Hash Tree Protocol with $2n$ Latency

```
Sender $s$ sends $(M, r)$ to $d_1$
for $i = 1$ to $n$ do
    If Node $d_i$ has seen $M$ before, abort.
    Node $d_i$ computes $L_{i+1}$ from $L_i$ and $v_i = \mathrm{PRF}_{K_i}(M)$
    Node $d_i$ sends $(M, r), L_{i+1}$ downstream to Node $d_{i+1}$
end for
for $i = n$ down to 1 do
    Node $d_i$ computes $R_{i-1}$ from $R_i$ and $v_i = \mathrm{PRF}_{K_i}(M)$
    Node $d_i$ sends $R_{i-1}$ upstream to Node $d_{i-1}$
end for
```

The communication congestion of this protocol is at most $\lceil \log n \rceil + 1$ hash tree vertices. The nodes with the most number of transmissions are the two inner vertices of every subtree of 4 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.

Optimization for n latency

The optimization of Section 10.2.1 reduced the number of passes of the broadcast authentication protocol from three passes down to two. We now present an additional optimization which reduces the latency of the algorithm to the equivalent of a single pass, i.e., n time steps. This optimization makes the protocol optimally fast, since all broadcast protocols on the linear topology require at least n time steps.

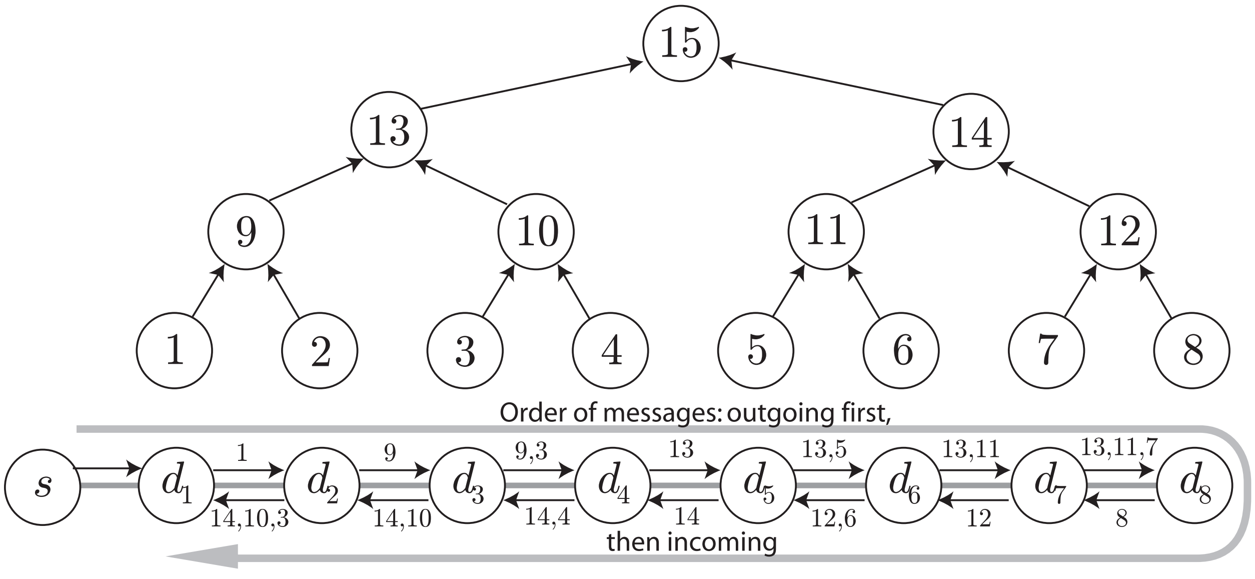


Figure 10.3: Example of Algorithm 1 with 8 nodes. Numbers indicate transmitted hash tree vertices.

The optimization is parameterizable to trade off latency and congestion, and is applicable to all 2-pass authentication protocols. The intuition is that we can perform a variant of divide-and-conquer on the receiver set. We observe that a 2-pass protocol takes $2n - 1$ latency due to the round-trip time from the sender to the farthest receiver and back. To reduce the size of this round-trip, we can partition the set of n receivers into two contiguous subsets R_1 and R_2. The first subset R_1 contains the first m receivers $\{r_1, \ldots, r_m\}$ (for some selected value of m) and R_2 contains the rest of the receivers $\{r_{m+1}, \ldots, r_n\}$. We then run the protocol separately for each subset. The sender computes different authenticators T_1, T_2 for each receiver subset R_1, R_2. Receiver subset R_1 performs the protocol using T_1, and forwards T_2 to R_2 as part of its own outgoing pass; once the last node r_m receives T_2, it is immediately forwarded to r_{m+1} in the next time step to allow R_2 to perform the protocol in a timely fashion. Receiver subset R_1 will take $2m$ latency: $2(m - 1)$ time steps to pass messages within the receiver set and an extra time step to receive the first message from s and another for r_m to forward T_2 to R_2. Receiver subset R_2 will take no more than $2(n-m)+m = 2n-m$ latency since it has $n-m$ receivers and needs m time steps before it receives T_2 forwarded across the m nodes of set R_1. Setting $m = \frac{2}{3}n$ causes both subsets to complete at the same time ($\frac{4}{3}n$ time steps) and minimizes the worst case latency.

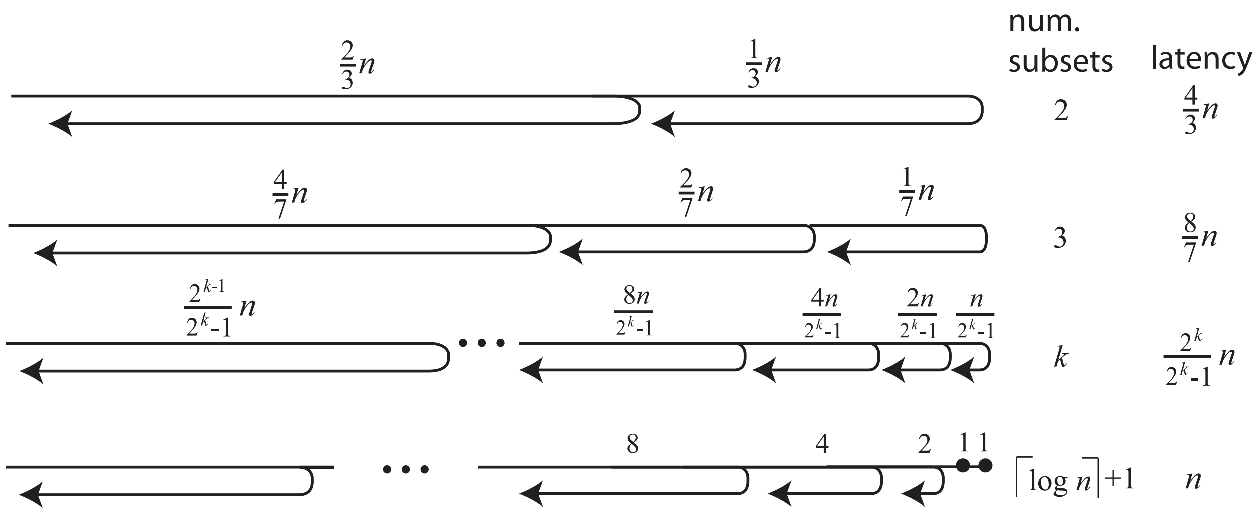


Figure 10.4: Latency optimization for 2-pass protocols

Figure 10.4 shows how partitioning the receiver set yields lower overall latency. The more subsets we use, the lower the latency, at the cost of greater congestion due to the larger number of authenticators that have to be forwarded to the respective receiver groups. For k subsets $R_1, R_2, \ldots, R_k$, worst case latency is minimized by having each successive receiver subset be half the size of the previous subset, i.e., $|R_{i+1}| = \frac{1}{2}|R_i|$ for $i = 1, \ldots, n-1$, with the smallest subset having $n/(2^k - 1)$ receivers. This results in an overall latency of approximately $n + n/(2^k - 1) + 1$ time steps. For values of n which are not divisible by $2^k - 1$, we need to round down the subset boundaries to the next largest integer value. This may introduce up to one extra time step in each subset. Hence the latency is bounded by $n + n/(2^k - 1) + 1$ time steps.

Each receiver subset R_i must receive a separate authenticator T_i to execute its respective authentication protocol; these authenticators must be forwarded through the first group R_1, adding to its congestion. In the hash tree based protocol, the sender sends one hash tree vertex to each receiver group; this causes an increase in communication congestion of k hash values for the nodes in the first receiver group, thus we have a bound of $\log n + k$ hash values on the communication congestion.

Since the optimization achieves $n + n/(2^k - 1) + 1$ latency for k subsets, setting $k = \log n$ suffices to achieve a latency of at most $n + 3$. To achieve optimal n latency, we can use a slightly different way of partitioning the receiver subsets as follows. Let $|R_k| = |R_{k-1}| = 1$, and each preceeding subset double in size, e.g., $|R_{k-2}| = 2, |R_{k-3}| = 4$, and so on until R_1 contains the remaining receivers. For example, a receiver set of size $n = 35$ can be divided into 7 subsets of size $3, 16, 8, 4, 2, 1, 1$ respectively. It can be seen that, except possibly for R_1 (which may complete early if n is not a power of 2), all subsets complete their respective protocols at the same time: after n time steps. This method of partitioning yields $\lceil \log n \rceil + 1$ subsets and involves a total of $2\lceil \log n \rceil + 1$ congestion in the hash tree protocol.

10.2.2 Hash Chain Based Schemes

As described in Section 10.1.1, in hash chain based schemes, to authenticate the ith message m_i, the sender first broadcasts $(m_i, \mathrm{MAC}_{h_i}(m_i))$ where h_i is the ith hash chain value. Once the sender has ensured that all receivers have received $\mathrm{MAC}_{h_i}(m_i)$, then the sender broadcasts h_i; receivers then check h_i against known values of the hash chain to verify h_i and verify the MAC to authenticate the message.

The main property that distinguishes different protocols is how to achieve *temporal separation* between MAC reception and MAC verification. In TESLA, the temporal separation is implicit by assuming that nodes are loosely time synchronized, and hence are aware of when a given MAC is no longer "fresh" and acceptable. Time synchronization may not be desirable for networks with high clock drift and where receivers may undergo long periods of sleep where desynchronization might occur; the resynchronization protocol itself must also be secure which raises another set of overheads and problems. In the Guy Fawkes/CSA protocol, temporal separation is provided through explicit coordination; each receiver uses a similar hash chain based protocol to return authenticated acknowledgements back to the sender to indicate that they have received the message. This causes the protocol to take three passes through the network.

We address efficient mechanisms for explicit coordination improving upon the properties of CSA. As a preliminary discussion, we first describe an application of a known construction [17, 15, 44]for authenticated acknowledgements that reduces the congestion overhead of the second phase (authenticated acknowledgement) to a constant; this has better properties than a previously-described authenticated acknowledgement construction described by Yao et al. [80]. Building on this, we then introduce an optimization which reduces the number of passes of the protocol from 3 to 2 passes.

Efficient Acknowledgements

In CSA, each receiver transmits a separate ACK message back to the sender. This causes n messages in total to be received by the sender, thus the communication congestion of the scheme is n cryptographic values. Yao et al. improve this for tree-based networks by proposing the construction of an authenticated aggregate acknowledgement via construction of a hash tree over the acknowledgements [80]. In their scheme, internal nodes in the tree topology wait to receive all (aggregated) acknowledgements from their children; they then compute a hash over these values and their own acknowledgement and forward the result to their respective parents. This approach reduces congestion to a single hash value but has some drawbacks: first, internal nodes must buffer all aggregated acknowledgements from their children before computing their own aggregated acknowledgement; second, the sender must know the exact topology of the tree to verify the aggregated acknowledgement (instead of just the receiver set). We note that if the hash function is instead replaced with the XOR-based construction of Section 4.4, all the security properties remain intact but the above disadvantages are removed: for example, in the linear topology, the sender does not need to know the sequence of receivers in the topology.

Construct a hash chain $h_0, \ldots, h_m$. 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. Assume that all nodes have received the previous value h_{j-1} on the previous stage and are waiting for the jth stage message M_j (in general, knowing any previous value h_k for $k < j$ will suffice [1, 66]). 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 some fixed length. Once each receiver d_i has received the message tuple, it extracts the value of j from N_j, and checks that j is larger than the latest stage number that d_i has seen so far, and if so, d_i releases $\text{PRF}_{K_i}(M_j \| N_j \| T_j)$ to the network. The acknowledgements occur in reverse order of reception of the broadcast in the first pass, i.e., starting from receiver r_n

and proceed towards the sender. Each receiver d_i computes the XOR of its own PRF value with all the PRF values of the receivers $d_{i+1}, \ldots, d_n$. Finally, the sender receives:

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

The sender can verify that the result was indeed composed of the XOR of all the respective PRFs 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 $T_j = \mathrm{MAC}_{h_j}(M_j \| N_j)$. If so, then they accept M_j. The algorithm for the linear topology is shown in pseudo-code in Algorithm 5.

Algorithm 5 Hash Chain Protocol with $O(1)$ congestion

```
Require: Hash chain $h_0, \ldots, h_m$
  $M_j = j$th Message ; $N_j = j$th Nonce
  $s$ computes: $T_j \leftarrow \mathrm{MAC}_{h_j}(M_j \| N_j)$
  $s$ multihop broadcasts $(M_j, N_j, T_j)$ to all receivers
  $A_j \leftarrow 0$
  for $i = n$ down to 1 do
    If $d_i$ has already processed stage $j$ or later, abort
    $d_i$ computes: $A_j \leftarrow A_j \oplus \mathrm{PRF}_{K_i}(M_j \| N_j \| T_j)$
    $d_i$ sends $A_j$ to $d_{i-1}$ (or to $s$ if $i = 1$)
  end for
  $s$ checks: $A_j = \bigoplus_{i=1}^{n} \mathrm{PRF}_{K_i}(M_j \| N_j \| T_j)$
  $s$ multihop broadcasts $h_j$ to all receivers
  for $i = 1$ to $n$ do
    $d_i$ checks that $H[h_j] = h_{j-1}$
    $d_i$ checks $T_j = \mathrm{MAC}_{h_j}(M_j \| N_j)$; if success, accept $M_j$.
    $d_i$ forwards $h_j$ to $d_{i+1}$
  end for
```

The security of the algorithm follows from the temporal separation of MAC reception with MAC verification. By the correctness of the subprotocol involving authenticated aggregated acknowledgements, the hash chain value h_j is released only after all nodes have received a message containing N_j. Since each node stops accepting messages from stage j or earlier as soon as it receives a candidate for stage j, when h_j is released, it is not useful for creating a new forged MAC for period j; deriving earlier keys from h_j is likewise not useful since all receivers have stopped accepting messages from those periods.

An adversary may 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 cannot 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 implemented for 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.

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.

Optimization for latency

In the original 3-pass protocol, the third pass where h_j is disseminated starts only after the reception of the aggregated acknowledgement A_j by the sender. This allows the sender to verify A_j before releasing h_j. However, passing A_j back to the sender, and then passing h_j out to all receivers, involves a round-trip between the sender and the farthest node 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 $A_j = \bigoplus_{i=1}^{n} \mathrm{PRF}_{K_i}(M_j \| N_j \| T_j)$ and includes $E_j = E_{A_j}(h_j)$ in the broadcast, where $E_K(X)$ denotes the encryption of string X 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 each A_j is used as a key to encrypt only one fixed-length plaintext h_j, if we set the PRF length and the hash length to be equal, we can simply use A_j as a pseudorandom masking factor for h_j as follows:

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

From this construction, it is clear that if even one PRF value is unknown, the adversary finds it infeasible to recover any bits of h_j from E_j. Hence, the adversary can only feasibly recover h_j after all legitimate nodes have released their PRF acknowledgement values 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 (a sketch of this proof is in Appendix 10.2.2). The modified protocol is described in pseudocode in Algorithm 6.

Algorithm 6 Hash Chain Protocol with 2 passes

```
Require: Hash chain $h_0, \ldots, h_m$
  $M_j = j$th Message ; $N_j = j$th Nonce
  $s$ computes: $T_j = \mathrm{MAC}_{h_j}(M_j \| N_j)$
  $s$ computes: $E_j = \bigoplus_{i=1}^{n} \mathrm{PRF}_{K_i}(M_j \| N_j \| T_j) \oplus h_j$
  $A_j \leftarrow 0$
  $s$ sends $(M_j, N_j, T_j, E_j, A_j)$ to $d_1$:
  for $i = 1$ to $n$ do
    If $d_i$ has already processed stage $j$ or later, abort
    $d_i$ computes: $A_j \leftarrow A_j \oplus \mathrm{PRF}_{K_i}(M_j \| N_j \| T_j)$
    $d_i$ forwards $(M_j, N_j, T_j, E_j, A_j)$ to $d_{i+1}$
  end for
  $d_n$ computes: $h_j \leftarrow A_j \oplus E_j$
  for $i = n$ downto 1 do
    $d_i$ checks $T_j = \mathrm{MAC}_{h_j}(M_j \| N_j)$; if success, accept $M_j$.
    $d_i$ forwards $h_j$ to $d_{i-1}$
  end for
```

The resultant protocol terminates in 2 passes and thus results in a latency of $2n$ for a linear topology. The communication overhead increases by one for the additional encrypted value, hence 6 cryptographic values are sent by each receiver in the network.

More importantly, since this protocol is now a 2-pass protocol, the speedup tradeoffs of Section 10.2.1 apply directly. For example, it is possible to speed up the latency to $\frac{4}{3}n$ by splitting the receiver set into two receiver subsets each with an separate tag and hash chain value, thus raising the communication overhead to 7 cryptographic values transmitted per node. In general, $n + n/(2^k - 1) + 1$ latency is achievable with k subsets implying a communication overhead of $2k + 4$ cryptographic values. Optimal n latency is achievable with $k = \lceil \log n \rceil + 1$ receiver sets, at the cost of $2\lceil \log n \rceil + 6$ communication congestion.

Security Proof for 2-Pass Hash Chain Scheme

The security proofs of TESLA and CSA can be found in their respective papers[66, 4]. In the unoptimized three-phase hash chain protocol, the message and tag are broadcast, then authenticated acknowledgements are received, and finally the hash chain value is released. This sequence of events is functionally identical to TESLA and CSA except with the addition of the use of aggregated authenticated acknowledgements, the correctness of which was shown by Chan et al and Katz et al [17, 44]. We sketch a proof of the security of our optimization where the hash value is encrypted using the aggregated PRFs of each receiver, i.e.,

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

We define the attacker-challenger Game 0 for the optimized scheme as follows. The game is described for $i \neq n$; the case where $i = n$ is straightforward and omitted for brevity.

- The protocol is initialized as normal for n nodes and all receiver nodes are given their respective keys.
- The adversary selects a target receiver i. The challenger provides the adversary with the keys and states of all receivers that are not i.
- The adversary sends $q - 1$ adaptive queries M_j to the challenger which responds by running the protocol as normal, i.e., the sender sends out correctly computed tuples (M_j, N_j, T_j, E_j) as per the protocol, using successive pre-images h_j on the hash chain for each successive j; receiver i also behaves correctly by producing $\mathrm{PRF}_{K_i}(M_j)$ on reception of each tuple. The number of queries is selected by the adversary.

- The qth query is an attack query. The adversary sends M_q to the challenger and receives (M_q, N_q, T_q, E_q).
- The adversary then chooses $M'_q \neq M_j$ for $j = 1, \ldots, q$. It constructs (M'_q, N'_q, T'_q, E'_q) which is sent to the challenger.
- The challenger implements the response of receiver i which gives $\mathrm{PRF}_{K_i}(M'_q)$ to the adversary.
- The adversary then produces h'_q to the challenger. The challenger simulates the operation of receiver i in verifying T'_q. If it accepts, then the adversary wins.

We first analyze Game 1 which is identical to Game 0 except we replace each PRF_{K_i} with a random function f_i. In game 1, if $f_i(M_j \| N_j \| T_j)$ is not known, then E_j leaks no information about h_j. Given a game for the unoptimized scheme, and an attacker A for game 1, we construct an attacker B that simulates the optimized Game 1 for attacker A thus using it to attack the unoptimized scheme. Attacker B answers attacker A's queries M_j by forwarding the queries to its challenger, thus receiving (M_j, N_j, T_j). It completes the response by picking each E_j as a random bitstring of the required length. When receiver i is given message M_j by attacker A, then attacker B also allows its own receiver i to receive M_j and respond with its authenticated acknowledgement. This allows that round of the protocol to complete, thus sender in the unoptimized scheme releases the hash chain value h_j. Using this value, attacker B can then reconstruct $f_i(M_j \| N_j \| T_j) = E_j \oplus h_j$ and release it to attacker A. The attack query of attacker A is simulated in an identical fashion, except that attacker B discards the values of E'_q provided by attacker A, and forwards the h'_q provided by attacker A to its own challenger. Clearly, the simulation implemented by attacker B is identical to Game 1. Furthermore, if attacker A succeeds in Game 1, then attacker B succeeds in attacking the unoptimized protocol. Hence Game 1 is at least as secure as the unoptimized protocol, which is as secure as TESLA and CSA.

It remains to show that converting from Game 1 to Game 0 confers an additional advantage no more than the distinguishability advantage ϵ of the PRF. We prove this in the usual way by constructing a distinguisher D for PRF_{K_i} and random function f_i from an adversary A for Game 0. Given oracle access to a function F, we implement Game 0 substituting F for PRF_{K_i}; hence

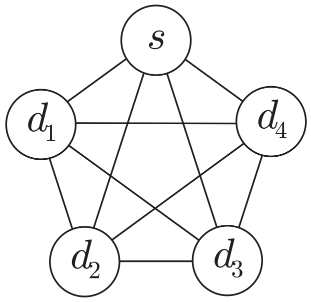


Figure 10.5: Fully-connected Topology

if F is pseudorandom then the game is equivalent to Game 0, if F is a random function then it is equivalent to Game 1. The distinguisher D runs adversary A on the game; and outputs 1 if A succeeds and 0 otherwise. If adversary A has a significantly different chance of success in game 0 vs game 1 then D has a significant chance of distinguishing the PRF from a random function; hence the advantage of A in Game 0 must be no more than ϵ higher than the advantage bound on adversaries in Game 1.

10.3 Fully Connected Topology

We now examine bounds and optimizations for the fully connected topology. In the fully connected topology, G is the complete graph of $n + 1$ vertices (including the sender s and the n receivers). Figure 10.5 shows an example of the fully-connected topology for 5 receivers. Fully-connected topologies are important in both theory and practice. In terms of theoretical importance, latency lower bounds proven on fully connected topologies apply to *all* topologies. Since any graph G of n nodes is a subgraph of K_n, any algorithm for G is also an algorithm for K_n; hence if we can show that no algorithm can be faster than $f(n)$ on the complete graph K_n, the bound also applies directly to arbitrary topologies. In terms of practical importance, fully-connected topologies occur whenever a network of nodes can achieve any-to-any addressability; this commonly occurs on Internet-based distributed systems such as peer-to-peer networks and grid computing. Although our algorithms assume uniform latency between any two nodes, this does not hinder the practical applicability of our algorithms. Our constructions specifically target resource-limited applications by not using asymmetric cryptography; such applications are unlikely to perform exhaustive latency

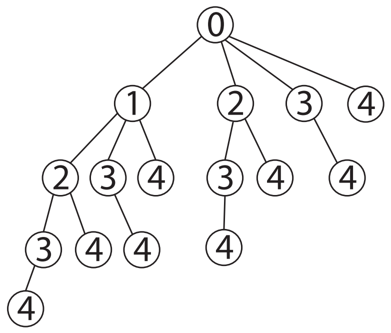


Figure 10.6: Maximum Dissemination Tree

measurements between pairs of nodes. In the absence of latency estimates, a reasonable course of action is to minimize the total number of connections made to other hosts in the course of the protocol. This also allows the protocol to bound the worst case protocol execution time as the number of time steps incurred multiplied by the worst case connection latency between hosts.

10.3.1 Doubling Broadcast / Maximum Dissemination Trees

As a preliminary, we first consider the problem of disseminating a message M from one node to as many other nodes as possible within t time steps. A straightforward approach is *doubling broadcast*: in each time step, every node which has knowledge of M communicates it to a different node that has not yet received M. An inductive reasoning on t shows that this schedule reaches the maximum number of nodes for each $t \geq 0$; since each node can only communicate with one other node in each time step, the number of nodes that have received M can at most double at each time step. Thus, after t time steps, up to 2^t nodes have received M. Figure 10.6 shows this process; the numbers in the circled nodes indicate in which time step the node first receives the message. We define the communication pattern caused by this optimal dissemination schedule the *maximum dissemination tree*. A consequence of this schedule being optimally fast is that any dissemination of a message from one node to n nodes must take at least $\lceil \log_2(n) \rceil$ time steps on any topology.

10.3.2 Hash Tree Based Schemes

The hash tree based broadcast authentication scheme of Chapter 5 takes three message passes to and from the sender to the rest of the network. This implies a $3 \log n$ latency for a naive

Algorithm 7 Doubling Broadcast (for $n = 2^k$ nodes)

```
Require: Message $M$ for broadcast from $d_0$
  for $h = 0$ to $k - 1$ do
    {The next loop happens simultaneously for all $i$}
    for $i = 0$ to $2^h$ do
      $d_i$ sends $M$ to $d_{i+2^h}$
    end for
  end for
```

implementation of the algorithm in a fully connected topology. As an improvement, we present a schedule which allows $2 \log n$ time steps. 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 sender computes the hash tree root r for the message M using the HT construction of Chapter 4.2. Then the message tuple (M, r) is disseminated using the repeated doubling broadcast method of Algorithm 7. This takes $1 + \log n$ time steps: one time step for the sender to send the message to the first receiver, and $\log n$ time steps for the receivers to disseminate the message. In the second phase, nodes collaborate to reconstruct authentication paths. This phase of messages is illustrated in Figure 10.7. There are two time-steps of message exchanges for each level of the hash tree, starting from the leaves. At each level j, every node exchanges the hash tree vertex at level j on its authentication path with that of its counterpart in the neighboring subtree at level j. 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 time step. Within $2\lceil \log n \rceil$ time steps all nodes will have computed their authentication paths to the root. This process is shown in Algorithm 8. The communication pattern is similar to all-to-all broadcast on a hypercube.

It remains to address the case where n is not a power of 2. In this case, we can partition the receiver set into up to $\log n$ subsets where each subset is a distinct power of 2. The subsets correspond to the binary representation of n, e.g., for $n = 13$, we have subsets of size $8, 4$ and 1. The protocol then runs, in parallel, one completely separate instance of the protocol for each subset. Since the sender can only start one instance of the protocol in each time step, it starts the protocol for the largest subset first, then it starts the protocol for the second largest subset in the

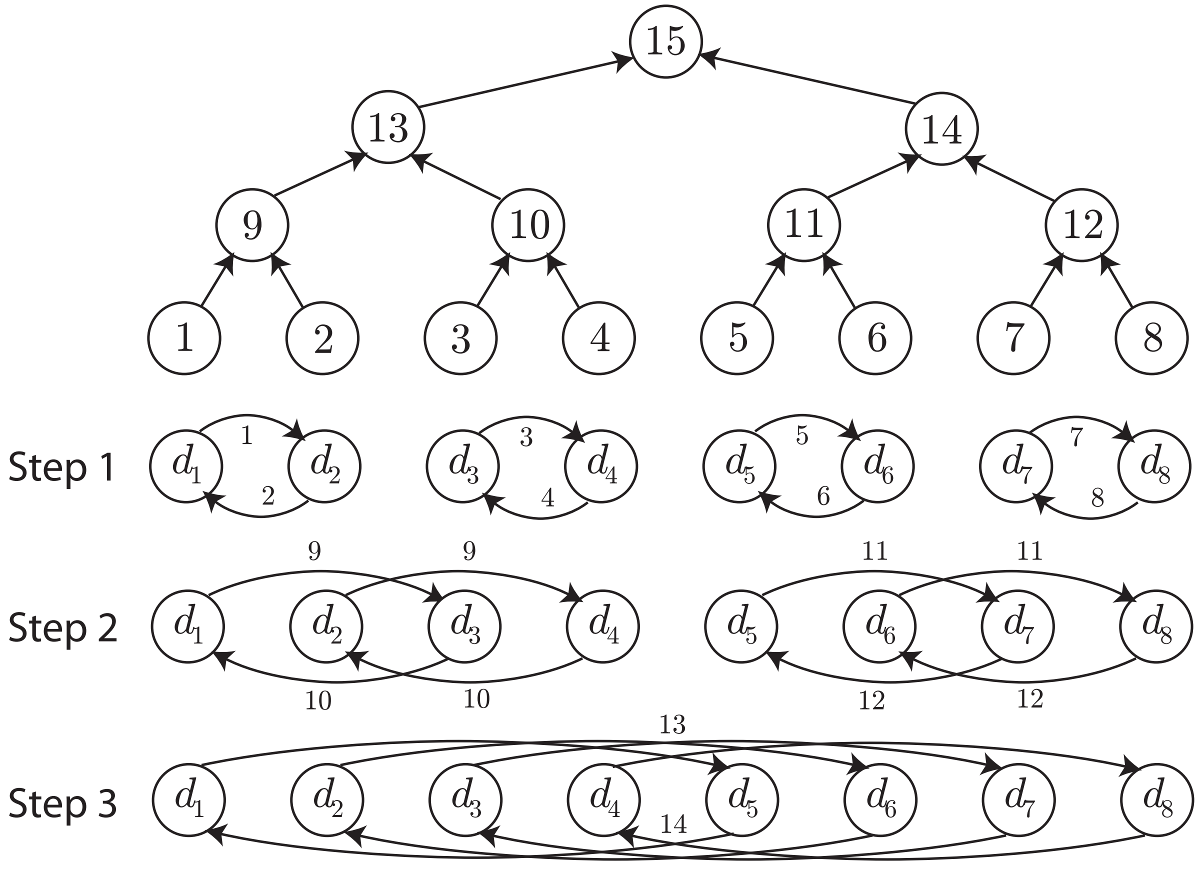


Figure 10.7: Path Reconstruction Phase

Algorithm 8 Hash Tree Scheme with $2\log n$ latency

```
$s$ sends $(M, r)$ to all receivers using Algorithm 7
If any receiver $d_i$ has seen $M$ before, abort
Otherwise, $d_i$ sets $v_i[0] \leftarrow \mathrm{PRF}_{K_i}(M)$
**for** $h = 0$ to $(\log n) - 1$ **do**
  **for** $i = 1$ to $n$ **do**
    $j \leftarrow ((i-1) \oplus 2^h) + 1$
    $d_i$ sends to $d_j$: $v_i[h]$
    $d_i$ receives from $d_j$: $v_j[h]$
    **if** $i < j$ **do** $v_i[h+1] \leftarrow H[v_i[h] \| v_j[h]]$
    **if** $i > j$ **do** $v_i[h+1] \leftarrow H[v_j[h] \| v_i[h]]$
  **end for**
**end for**
Each $d_i$ checks authentication path in $v_i$.
```

next time step, and so on. Even with this head start, the largest subset has the largest hash tree and will be the last subset to complete verification; it has $2^{\lfloor \log n \rfloor}$ nodes and thus takes $1+2\lfloor \log n \rfloor$, or fewer than $2\lceil \log n \rceil$ time steps to complete verification.

The communication congestion overhead for this scheme is at most $3\lceil \log n \rceil$ values: $2\lceil \log n \rceil$ for the dissemination of the tuple (M, r), and $\lceil \log n \rceil$ for the authentication path.

10.3.3 Hash Chain Based Schemes

One of the main advantages of the hash chain family of authentication schemes over the hash tree scheme 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 a constant communication congestion authentication protocol for the fully connected topology albeit with somewhat higher latency.

The doubling broadcast (Algorithm 7) requires $\log n$ congestion at the first node to achieve $\log n$ latency; hence we must consider a different schedule for constant-congestion broadcast. We define a *Fibonacci Tree* T over the topology, rooted at s, and use it for both broadcast dissemination and acknowledgement aggregation. A Fibonacci tree of height h is defined recursively as a binary tree with a root vertex r which has, as a left subtree, a Fibonacci tree of height $h-1$, and as a right subtree, a Fibonacci tree of height $h-2$. For base cases, Fibonacci trees of height 0 and 1 are defined as the single vertex and the a root vertex with a single left-child respectively. A (complete) Fibonacci tree of height h has $F(h+3)-1$ nodes where $F(x)$ is the xth Fibonacci number. We assume that each receiver knows its position in the tree T. For brevity we only describe the algorithm where the Fibonacci tree embeds directly in K_n, i.e., for $n = F(h+3)-1$; the case for arbitrary n can be easily generalized by setting h be the smallest integer such that $n \leq F(h+3)-1$.

When using the Fibonacci tree for broadcast, each internal node disseminates the message first to its left child, and then to its right child. When performing a convergecast, the opposite happens, i.e., first the right child sends a message to the parent, and then the left. This timing requirement need not be explicit since each node can simply respond to its parent as soon as all its children have responded; the structure of the Fibonacci tree ensures that typically no two children will respond to the same parent at the same time.

The 3-pass protocol of Section 10.2.2 can thus be implemented in the straightforward way on the Fibonacci Tree subgraph of K_n. For the jth message, the sender first broadcasts (M_j, N_j, T_j) on the Fibonacci Tree; then all nodes perform a convergecast on the same tree, aggregating authenticated acknowledgements $(\mathrm{PRF}_{K_i}(M_j, N_j, T_j))$ of the reception of this message using XOR. Finally the sender checks the aggregated ACKs and then broadcasts h_j on the Fibonacci tree. The result is three traversals of the Fibonacci Tree. The algorithm is shown in Algorithm 9.

Algorithm 9 Hash Chain Protocol with $4.32 \log n$ latency

```
Require: Public embedding of Fibonacci Tree $T$ on $K_n$
Require: Hash chain $h_0, \ldots, h_m$
  $M_j$ = $j$th Message ; $N_j$ = $j$th Nonce
  $s$ computes: $T_j = \mathrm{MAC}_{h_j}(M_j \| N_j)$
  $s$ broadcasts $(M_j, N_j, T_j)$ along $T$
  repeat
    for all $d_i$ that have received messages from all their children (if any) do
      If $d_i$ has already processed stage $j$ or later, abort
      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. $v_r$) $= 0$.
      $d_i$ sends to its parent: $v_l \oplus v_r \oplus \mathrm{PRF}_{K_i}(M_j \| N_j \| T_j)$
    end for
  until $s$ has received messages from all its children
  $s$ checks: $A_j = \bigoplus_{i=1}^{n} \mathrm{PRF}_{K_i}(M_j \| N_j \| T_j)$.
  If so, $s$ broadcasts $h_j$ along $T$
  Each receiver $d_i$ authenticates $M_j$ with $h_j, T_j$.
```

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 $\lceil 1.44 \log n - 0.328 \rceil$. The algorithm performs three passes along the tree, 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.

10.4 Congestion-Latency Lower Bounds

Several of our schemes for the linear topology are latency-optimal: they take n rounds, which is the lower bound on the time for message dissemination. It is less clear what lower bounds exist for the fully connected topology. The requirements for message dissemination ($\log n$ rounds for doubling broadcast) provide a loose bound, which can be achieved by attaching a digital signature to every message. However digital signatures represent a fundamentally harder problem than authentication, being both more computationally expensive and providing additional unnecessary properties such as non-repudiation. Therefore, we investigate what bounds can be derived for protocols which do not involve the digital signature problem.

We prove that any network broadcast authentication protocol that completes in at most $(2 - \rho) \log n$ time steps (for $\rho > 0$) must imply a digital signature or have communication overhead at least proportional to n^ρ. By implication, any protocol that completes in time asymptotically less than $2 \log n$ cannot have communication complexity polylogarithmic in n unless it also provides signature properties (i.e., it solves a fundamentally harder problem).

We build on the result of Boneh, Durfee and Franklin [7] which showed that any *multicast MAC* (or MMAC) for n receivers must either imply a digital signature or have overhead proportional to n. An MMAC is defined by three algorithms $\mathsf{key\text{-}gen}$, $\mathsf{mac\text{-}gen}$, $\mathsf{mac\text{-}ver}$. Algorithm $\mathsf{key\text{-}gen}$ takes a security parameter s and a number of receivers n and generates the sender key $\mathbf{sk}$ and receiver keys $\mathbf{rk}_1, \ldots, \mathbf{rk}_n$ for each receiver $d_1, \ldots, d_n$. Algorithm $\mathsf{mac\text{-}gen}$ takes a payload message M and a sender key $\mathbf{sk}$ returns a fixed-length tag $T = \mathsf{mac\text{-}gen}(M, \mathbf{sk})$. Algorithm $\mathsf{mac\text{-}ver}$ takes a payload message M, a tag T, and a receiver key K and returns a bit indicating whether the tag T verified correctly for payload message M, under the constraint that correctly generated tags from the sender always verify successfully for all receivers, i.e., $\mathsf{mac\text{-}ver}(M, \mathsf{mac\text{-}gen}(M, \mathbf{sk}), \mathbf{rk}_i) = \text{'yes'}$ for all $i = 1, \ldots, n$.

MMACs are static codes; in this sense they can be considered a special case of the set of broadcast authentication protocols under the restriction that the sender only produces a single, fixed-length tag T for all receivers. Each receiver must then perform verification by inspection of T without interacting with the sender or other receivers. General broadcast authentication protocols, on the other hand, allow the sender to send different protocol messages to different receivers, which can perform rounds of interaction with each other or the sender to determine if the payload is legitimate. Boneh, Durfee and Franklin's result holds only for MMACs and do not apply to broadcast authentication protocols in general. We extend their result to show that if a broadcast authentication protocol is fast, then it implies an MMAC. The basic idea is that if the protocol is fast, then there must exist a (large) set of receivers that did not interact with each other in the protocol; in the absence of interaction between receivers, a broadcast authentication protocol can be reduced to an MMAC.

The details of the proof are as follows. To begin with, assume that all receivers are deterministic, i.e., they do not perform any purely random coin tosses during the protocol, although they may simulate this with pseudorandom values generated from a preloaded seed. We assume that the communication pattern is static for each execution of the protocol, i.e., the set of nodes that communicate with each other in each round is fixed regardless of the message or tag originated by the sender. These assumptions are consistent with all known broadcast authentication protocols in the literature. We do not count setup overhead (such as key establishment, etc) in the rounds of the protocol; thus we can assume that no receiver starts the protocol until it receives a message that contains information from the sender.

The proof involves a reduction from a fast broadcast authentication protocol to an MMAC; we define the security games for each class of problems as follows. For the MMAC security game: (1) The adversary *adaptively* selects a proper subset S_A of S for compromise, and the challenger provides A with all secret information known by these receivers as they are selected; the adversary can expand her choice of S_A based on this information. (2) The adversary can query the challenger to provide valid MMACs for some adaptively chosen $M_1, \ldots, M_q$. (3) The adversary constructs a forged data-message $M' \neq M_i$ for all $i, \ldots, q$ and a tag T'. The adversary wins the game if some receiver in $S - S_A$ accepts message M'. We say the MMAC is ϵ-secure if the probability of any adversary winning is at most ϵ.

The security game for existential forgery on the broadcast authentication protocol is similarly structured: (1) The adversary adaptively selects a subset of receivers C for compromise, and the challenger provides all secret information known by these receivers. (2) The adversary can query the challenger to execute the authentication protocol for some adaptively chosen $M_1, \ldots, M_q$. (3) The adversary constructs a forged data-message $M' \neq M_i$ for all $i, \ldots, q$, and sends a (polynomially bounded) series of messages to some receivers not in C. The adversary wins if any of these uncompromised receivers accept M' because of these messages. We say the protocol is ϵ-secure if the probability of any adversary winning is at most ϵ.

The construction of a MMAC from a broadcast protocol is based on finding a set of nodes that do not interact directly or indirectly in the protocol. More formally:

Definition 20. *For a given sequence of messages in a protocol, Node A is an* upstream *node of node B if A could have sent any information to Node B in the protocol. Let B be a* downstream *node of node A iff node A is upstream of B.*

More precisely, A is upstream of B if there exists a sequence of nodes $X_1, \ldots, X_k$ and messages $m_1, \ldots, m_k$ in successive (not necessarily consecutive) time steps that start with A and end with B (e.g., $A \rightarrow X_1 : m_1$, $X_1 \rightarrow X_2 : m_2$, …, $X_k \rightarrow B : m_k$ in successive time steps).

Theorem 21. *Let P be a broadcast authentication protocol that is ϵ-secure against arbitrary coalitions of receivers that has congestion at least $c(n)$ bits. Let S be a subset of m receivers in the protocol P such that no two members of S are upstream (or downstream) of each other. Then there exists an MMAC for m receivers with tag length at least $c(n)$ bits that is also ϵ-secure.*

Proof. We can construct a MMAC for receivers $s_1, \ldots, s_m$ in S by defining the three subroutines key-gen,mac-gen and mac-ver in terms of the broadcast authentication protocol P. Let key-gen be defined as follows: for each receiver s_i in S, let $\mathbf{rk}_i$ be the receiver key of s_i in P; let the information $\mathbf{sk}$ known to the sender be all the information known by all principals in the broadcast protocol. This allows the sender to fully simulate the execution of any run of the protocol; in particular it can predict *a-priori* the set of messages it will send in any run of the protocol for a given message M. Hence, let the tag T generated by mac-gen$(M, \mathbf{sk})$ be the ordered set of all messages sent by the sender for a given broadcast of message M. We define mac-ver$(M, T, \mathbf{rk}_i)$ such that it simulates the operation of P to the point of verification by receiver s_i. Let R_S be the set of all nodes that are upstream of any node in S. By definition of S, $R_S \cap S = \emptyset$. Encode (publicly) inside mac-ver all the protocol information associated with the nodes in R_S (including their receiver keys). By definition of R_S, this information allows any MMAC receiver s_i to accurately simulate the entire broadcast protocol from the point of view of node s_i in the broadcast protocol; in other words, all messages sent and received by node s_i in the broadcast protocol can be generated from this information by MMAC receiver s_i. We can thus define mac-ver$(M, T, \mathbf{rk}_i)$ to return 'yes' if s_i accepts the payload

M in the simulation of the protocol P where the sender sends the messages encoded in T, and 'no' if s_i does not accept M.

Since the protocol P has at most $c(n)$ bits of congestion, the tag T is no more than $c(n)$ bits since it consists of all messages from a single node (the sender). It remains to show that the MMAC construction is at least as secure as P.

Let adversary A be an adversary for the MMAC game that wins with probability ϵ. We define Adversary B for attacking the broadcast authentication protocol that runs A as a subroutine. Adversary B first compromises R_S, the set of all nodes upstream of S. From this, B constructs the key-gen,mac-gen and mac-ver for the MMAC, which is fed to A. Now A adaptively selects some set S_A for compromise; these nodes are also adaptively compromised by B in the broadcast authentication protocol. By definition of S, $R_S \cap S = \emptyset$, so the set $S - S_A$ remain uncompromised for both adversaries. During the chosen message attack phase, queries of A are forwarded to B's challenger, and the corresponding MMACs are reconstructed by inspection of the sender's messages for M_i. Now when attacker A produces M' and T', since Adversary B has compromised all receivers upstream of S, it can use this information as a script to (interactively and independently) simulate the protocol P to each member of S. Since mac-ver(M', T', rk_i) succeeds with at least ϵ probability for some receiver in $S - S_A$, by construction, the chance of Adversary B successfully causing a receiver in $S - S_A$ to accept the forged broadcast message M' is also ϵ. □

Lemma 22. *In any broadcast authentication protocol, for any topology, at least $n/2$ nodes start executing the protocol only after $\lfloor logn \rfloor$ time steps.*

Proof. The doubling broadcast of Sec. 10.3.1) is the fastest rate at which information can spread in the network. At time step $\lfloor \log n \rfloor - 1$, no more than $2^{\lfloor \log n \rfloor - 1} \leq n/2$ nodes have received their first message of the protocol; the remaining nodes must start the protocol after $\lfloor \log n \rfloor$ time steps. □

Lemma 23. *Let G be a network topology and $L \subseteq V$ be a subset of the nodes in G. Let $U(v, t)$ be an upper bound on the number of nodes in L that can receive information from a node $v \in L$ within t time steps. Then $U(v, t)$ is also an upper bound on the number of nodes in L that can send information to v within t time steps in the topology.*

Proof. Any schedule for converge-casting information to v can be converted to a schedule for disseminating information from v (and vice-versa) by reversing the schedule chronologically and reversing the direction of all communications. This establishes a one-to-one correspondence between schedules that disseminate information and schedules that collect information. Hence, in a fixed topology, the maximum possible number of upstream and downstream nodes from a given node v are the same. □

Lemma 24. *If a broadcast authentication protocol takes no more than $(2-\rho)\lfloor \log n \rfloor$ time steps, then there exists a set S of at least $\frac{1}{4}n^{\rho}$ receivers that are not mutually upstream or downstream of each other.*

Proof. Let L be the set of receivers which started the protocol after time step $\lfloor \log n \rfloor$. By Lemma 22, $|L| \geq n/2$. The nodes in L only have $(1-\rho)\lfloor \log n \rfloor$ time steps remaining in which to exchange information. By the doubling broadcast of Section 10.3.1, any node in L can reach at most $2^{(1-\rho)\log n} \leq n^{1-\rho}$ other members of L within that time; by Lemma 23, likewise less than $n^{1-\rho}$ of the other nodes in L can be upstream nodes of l_i. Hence, we can construct S as follows: starting with empty S, select any node l_i in L; remove it from L and add it to S; also remove all nodes in L that are upstream or downstream of l_i. After each selection, the size of L decreases by less than $2n^{1-\rho}$ nodes. Hence, we can perform such a selection at least $\frac{1}{2}n/2n^{1-\rho} = \frac{1}{4}n^{\rho}$ times. The resultant S has at least $\frac{1}{4}n^{\rho}$ nodes each of which are neither upstream nor downstream of each other. □

Theorem 25. *Any deterministic ϵ-secure broadcast protocol taking at most $(2-\rho)\lfloor \log n \rfloor$ time steps either implies an $(\epsilon + 1/2^m)$-secure signature mechanism or requires at least $\frac{1}{4}n^{\rho} - m$ bits of congestion where m is a constant security parameter (e.g., 128).*

Proof. By Lemmas 24 and 21, any deterministic ϵ-secure broadcast protocol taking at most $(2-\rho)\lfloor \log n \rfloor$ time steps implies an ϵ-secure MMAC for $\frac{1}{4}n^{\rho}$ receivers with tag length equal to the congestion at the sender. Boneh, Durfee and Franklin showed that an ϵ-secure MMAC for q receivers with tag length no more than $q-m$ bits implies an $(\epsilon+1/2^m)$-secure signature scheme [7]. Setting $q = \frac{1}{4}n^{\rho}$, the result follows. □

More significantly, this implies that no deterministic broadcast protocol can take asymptotically less than $2 \log n$ time steps while achieving congestion complexity polylogarithmic in n. This means that the hash tree based algorithm of Section 10.3.2 achieving $2 \log n$ time steps is optimally fast for protocols with polylogarithmic congestion complexity; in other words the existence of the algorithm of Section 10.3.2 shows that bound we have proved is tight. As noted earlier in the beginning of Section 10.3, this lower bound holds for all topologies.

10.5 Extending the Lower Bound to Tree Topologies

The method of proving the latency lower bound of $2 \log n$ for arbitrary topologies can be extended to trees. The proof for trees follows the same structure as the proof for fully connected topologies (Section 10.4). The key insight is that while information spreads at the rate of 2^t in fully-connected topologies, we can show that it must spread at a slower rate from *maximum-depth* leaf nodes in trees. Specifically, we prove the following lemma:

Lemma 26. *Let v be a vertex at the* greatest depth *in an arbitrary tree topology with a designated root r. The number of nodes that can be reached from v within t time steps is no more than $\phi^{t+3}/\sqrt{5}$, where ϕ is the golden ratio (1.618).*

Proof. We consider the set of all nodes reachable from each node at each depth from u to the root. We define a term $T(t, h)$ as the size of the h *height-bounded maximum dissemination tree* (see Section 10.3.1, Figure 10.6) of t time steps. This can be defined as the maximum size of the set of nodes reached from a single node v_0 given t time steps such that no node is more than h hops away from v_0. This can be obtained from the maximum dissemination tree $T(t)$ by truncating off all levels with depth greater than h. We observe that $T(t, h) = T(t-1, h) + T(t-1, h-1)$. This follows from the observation that in the first time step, the original node v_0 can send to at most one other node v_1; in the remaining $t-1$ time steps, each of these two nodes then continue disseminating into their respective subtrees, except that the height bound of the maximum dissemination tree rooted at v_1 is one less than the original height bound h. This reasoning is shown in Figure 10.8. For base cases, we note that, for any x, $T(x, 0) = 1$ (a height bound of 0 does not allow the original node to disseminate to any other node) and $T(0, x) = 1$ (with 0 time steps, no dissemination takes place).

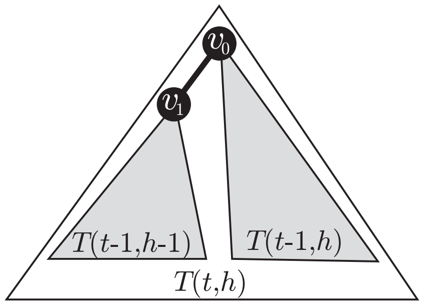


Figure 10.8: Recursive structure of $T(t, h)$

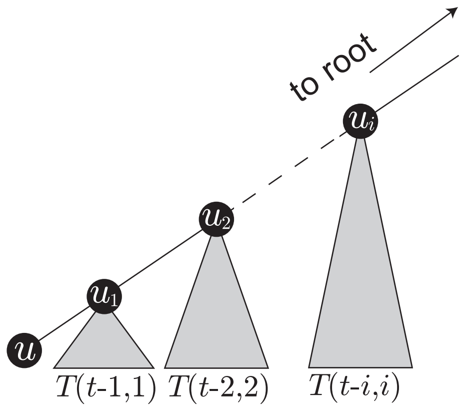


Figure 10.9: Nodes reachable from u

We can now consider the maximum number of nodes reachable from a max-depth leaf vertex u within t time steps. The argument is illustrated in Figure 10.9. The figure shows the nodes between u and the root r. The node u_i is i levels above u. It can be reached no earlier than at time step i. Thus, u_i has $t-i$ time steps remaining to disseminate into its subtree. Since u is a node at maximum depth in the tree, the subtree rooted at u_i has a height bound of i. Hence, the maximum number of nodes reached from u_i is $T(t-i,i)$. A bound on the total number of nodes reachable from u in time t is thus $F(t)=\sum_{i=0}^{t} T(t-i,i)$. Using the recurrence $T(t,h)=T(t-1,h)+T(t-1,h-1)$ we can expand $F(t)$ as follows:

$$\begin{aligned}
F(t) &= \sum_{i=0}^{t} T(t-i,i)\\
&= T(t,0)+\sum_{i=1}^{t-1} T(t-i,i)+T(0,t)\\
&= 1+\sum_{i=1}^{t-1}\left[T(t-i-1,i)+T(t-i-1,i-1)\right]+1\\
&= T(t-1,0)+\sum_{i=1}^{t-1} T(t-1-i,i)+\sum_{i=1}^{t-1} T(t-2-(i-1),i-1)+1\\
&= \sum_{i=0}^{t-1} T(t-1-i,i)+\sum_{j=0}^{t-2} T(t-2-j,j)+1\\
&= F(t-1)+F(t-2)+1
\end{aligned}$$

By substituting $F'(t)=F(t)+1$, we can see that $F'(t)=F'(t-1)+F'(t-2)$ yielding a Fibonacci series. From the base cases $F(0)=1, F(1)=2$ we can show $F(t)=\mathrm{Fib}(t+3)-1$ where $\mathrm{Fib}(x)$ is the xth number in the standard Fibonacci series. From Binet's formula $\mathrm{Fib}(x)=\mathrm{round}(\phi^x/\sqrt{5})$, we have $F(t)<\phi^{t+3}/\sqrt{5}$. □

Lemma 27. *If a broadcast authentication protocol on a tree topology takes no more than $(2.44-\rho)\log n-2.33$ time steps, then there exists a set S of at least $\frac{1}{4}n^{0.694\rho}$ receivers that are not mutually upstream or downstream of each other.*

Proof. Let L be the set of receivers which started the protocol after time step $\log n-1$. By

Lemma 22, $|L| \geq n/2$. After these first $\log n - 1$ time steps, there are $(1.44 - \rho)\log n - 1.33$ steps remaining. Consider the *maximum-depth* node u in L relative to the sender at the root of the tree topology. We can bound the number of members of L reachable by u in the remaining time by Lemma 26 because any nodes that are at a greater depth than u in the tree are all not in L and are hence irrelevant to the count. Let t be the time needed for u to reach up to n receivers. We can set $\phi^{t+3}/\sqrt{5} = n$, and solve $t = \log_\phi n + (\log_\phi \sqrt{5} - 3) = 1.44 \log n - 1.33$. Hence, in the remaining time steps, u can reach at most $\phi^{(1.44-\rho)\log n + 1.67}/\sqrt{5} \leq n^{1-0.694\rho}$ total members of L (including u itself); by Lemma 23, likewise less than $n^{1-0.694\rho}$ of the other nodes in L can be upstream nodes of l_i. Hence, we can construct S as follows: starting with empty S, select a node l_i that is at the greatest depth in the tree among all the nodes in L; remove it from L and add it to S; also remove all nodes in L that are upstream or downstream of l_i. Repeat until L is empty. After each selection, the size of L decreases by no more than $2n^{1-0.694\rho}$ nodes. Hence, we can perform such a selection at least $\frac{1}{2}n/2n^{1-0.694\rho} = \frac{1}{4}n^{0.694\rho}$ times. The resultant S has at least $\frac{1}{4}n^{0.694\rho}$ nodes each of which are neither upstream nor downstream of each other. □

Theorem 28. *Any deterministic ϵ-secure broadcast protocol on a tree topology taking at most $(2.44-\rho)\log n - 2.33$ time steps either implies an $(\epsilon + 1/2^m)$-secure signature mechanism or requires at least $\frac{1}{4}n^{0.694\rho} - m$ bits of congestion where m is a constant security parameter (e.g., 128).*

Proof. Identical to the proof of Theorem 25, except using Lemma 27 instead of Lemma 24. □

Theorem 28 implies that any scheme with at most $(2.44-\rho)\log n - 2.33$ time steps must involve either a signature scheme or a non-polylog amount of congestion. In particular this rules out the possibility of strictly 2-pass schemes with polylog congestion. From the discussion at the beginning of Section 10.3.2, we note that a 3-pass broadcast authentication protocol can be as fast as $3\log n$ time steps; we conjecture that the small gap factor between 2.44 and 3 seems to indicate that the current 3-pass schemes are likely latency-optimal in terms of the factors of $\log n$.

10.6 Low Latency Authenticated Broadcast: Results

A summary of the results of this Chapter is presented in Table 10.1. The new protocols avoid the high computational overhead of digital signatures and the high communication overhead of one-time signatures and multi-receiver MACs and the time synchronization needed by TESLA. In terms of latency and communication congestion, our protocols provide points in the design space that are not achievable by previously published protocols. In particular, for the linear topology, the hash tree scheme achieves the fastest possible latency n with only $O(\log n)$ congestion; the hash chain scheme achieves constant congestion with only $2n$ latency. For the fully connected topology, the hash tree scheme takes at most $2\lceil\log n\rceil$ latency with only $O(\log n)$ congestion; since we also show that this is the fastest possible latency achievable for schemes with polylogarithmic congestion, this also functions as an existence proof that the $2\log n$ bound is tight. We also show previously uninvestigated latency bounds for symmetric-key-based broadcast authentication in fully connected and tree topologies. We show a tight latency bound of $2\log n$ time steps for broadcast authentication schemes with polylogarithmic congestion in any topology and another latency lower bound of $2.44\log n - 2.33$ time steps for trees. These lower bounds show the fundamental latency limitations of broadcast authentication protocols, and can inform the design constraints of new authentication protocols in networks; for example, the $2.44\log n - 2.33$ time bound shows it is impossible for any polylog-congestion broadcast authentication protocol to make (asymptotically) only two or less passes up and down the height of a tree. The new bounds also significantly extend the relevance of the result on multicast message authentication by Boneh, Durfee and Franklin [7]. The previous result applied only to static *authentication codes* and did not address *protocols* involving multiple interactions between receiver/sender nodes, which are in fact the primary methods of multicast message authentication in practice. In our new latency lower bounds, we greatly improve the relevance of this result by showing how it can be extended to apply to general authentication protocols by considering the time dimension. This significantly expands the existing level of understanding about the fundamental limits in signature-free multicast authentication.

	Latency	Congestion
Linear Topology		
Unsecured Broadcast	n	1
HT (Sec.10.2.1)	$2n$	$\lceil \log n \rceil + 1$
HT, k subsets (Sec.10.2.1)	$n + \frac{n}{(2^k - 1)} + 1$	$\lceil \log n \rceil + k$
HT, $\lceil \log n \rceil + 1$ subsets	n	$2\lceil \log n \rceil + 1$
HC, 3-pass (Sec.10.2.2)	$3n$	5
HC, 2-pass (Sec.10.2.2)	$2n$	6
HC, k subsets (Sec.10.2.2)	$n + \frac{n}{(2^k - 1)} + 1$	$2k + 4$
HC, $\lceil \log n \rceil + 1$ subsets	n	$2\lceil \log n \rceil + 6$
Tree Topology		
Lower Bound (Sec.10.5)	$(2.44 - \rho) \log n - 2.33$	$\Omega(n^{0.694\rho})$
Full Topology		
Lower Bound (Sec.10.4)	$(2 - \rho) \log n$	$\Omega(n^{\rho})$
HT (Sec.10.3.2)	$2\lceil \log n \rceil$	$3\lceil \log n \rceil$
HC (Sec.10.3.3)	$4.32 \log n + 2$	9

Table 10.1: Latency and congestion bounds of our protocols. HT: Hash Tree based scheme; HC: Hash Chain based scheme.

Chapter 11

Summary and Conclusion

In this dissertation we show how the existence of a trusted authority combined with topology knowledge can be used to derive efficient and secure algorithms for communication and computation. Using an approach based on the distributed construction and dissemination of hash trees, we demonstrated new efficient primitives for general communications and computation problems such as authenticated broadcast, information binding, node-to-node signatures, and SUM data aggregation. All these primitives incur $O(\log n)$ overhead and only require that the base station shares a unique pairwise key with each sensor node, and are secure against arbitrary fractions of adversarial node compromise.

In addition, we describe a further extension of these techniques to implement a secure algorithm for performing computations on a tree for *arbitrary* continuous functions. This algorithm incurs congestion $O(hd)$ where h is the height and d is the degree of the computation tree. To the best of our knowledge, this is the first practical algorithm for ensuring the integrity for an entire general class of computations (rather than for a fixed functional primitive like SUM).

As a further extension, we also describe a protocol for the computation of arbitrary fan-in function computations where the range of each subfunction need not be continuous (or indeed even in $\mathbb{R}$). The only limitation on the type of function is that the legality of each input value should be checkable against a description of size α, and the output of each subfunction must be of size at most β, where α and β can be functions of n. This scheme is secure against up to k adversarial nodes (where k is a tunable parameter). The congestion overhead of the scheme is $O(3\alpha k(d+1)+\beta hd)$. This is the only known protocol in the literature that provides security against completely arbitrary function computations and still ensures sub-linear congestion on a tree topology.

As an application of our primitives, we describe a scheme for detecting and enforcing message origination rates in structured networks. The rate-limiting protocol induces $O(dn_a)$ congestion and requires $O(dn_a)$ memory overhead on sensor nodes, where d is the maximum indegree in the network and n_a is the maximum number of ancestors of any node, and is the only known rate-limiting protocol in the literature.

Finally, we describe optimizations for our hash-tree-based broadcast authentication scheme and the well-known hash-chain-based broadcast authentication protocol family that leverage the trusted authority and the known topology to reduce the latency of these protocols several-fold in certain special-case topologies. Our optimizations provide points in the design space that are not achievable by previously published protocols. These results are shown in Table 10.1. In particular, for the linear topology, our optimizations for the hash tree scheme achieves the fastest possible latency n with only $O(\log n)$ congestion; our optimizations for the hash chain scheme achieves constant congestion with only $2n$ latency. For the fully connected topology, our optimizations for the hash tree scheme takes at most $2\lceil\log n\rceil$ latency with only $O(\log n)$ congestion; we prove that it is impossible to achieve lower latency than $2\log n$ without an asymptotic increase in congestion. We also present an efficient schedule for three-pass hash-chain protocols on the fully-connected topology taking $4.32\log n$ latency with constant congestion. Finally, we show an asymtotic $2.44\log n$ time steps lower bound for trees.

In summary, our results highlight the fact that leveraging the existence of a trusted authority and the pre-knowledge of network topology yields extremely efficient and general secure algorithms that outperform existing algorithms with weaker assumptions. Since these properties are fundamental properties of structured networks with a single "owner" such as sensor networks and peer-to-peer networks, they represent a large and important practical application area. Our algorithms and analytical results bridge the gap between the theoretical work in secure multiparty computation and byzantine-resilient distributed algorithms, which have yielded highly powerful and general algorithms but unfortunately have impractically high communication overheads for the specific problems of computation and communication examined in this dissertation; and the work in ad-hoc networks which have produced many practically useful algorithms (such as techniques for secure routing, time synchronization and localization) but which have not addressed the fundamental general problems of in-network communication and computation.

Our results seem to indicate that security in structured networks with a trusted authority may be fundamentally easier than in general networks. It is our hope that further exploration of this unique security model can yield additional fundamental insights into the nature of secure computation on network models that closely resemble real-world applications.

Bibliography

[1] R. Anderson, F. Bergadano, B. Crispo, J. Lee, C. Manifavas, and R. Needham. A new family of authentication protocols. *SIGOPS Oper. Syst. Rev.*, 32(4):9–20, 1998.

[2] M. Ben-Or, E. Pavlov, and V. Vaikuntanathan. Byzantine agreement in the full-information model in o(log n) rounds. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 179–186, 2006.

[3] M. Ben-Or and D. Ron. Agreement in the presence of faults, on networks of bounded degree. *Inf. Process. Lett*, 57(6):329–334, 1996.

[4] F. Bergadano, D. Cavagnino, and B. Crispo. Chained stream authentication. In *Proceedings of the International Workshop on Selected Areas in Cryptography*, pages 144 – 157, 2001.

[5] P. Berman and J. Garay. Fast consensus in networks of bounded degree. *Distributed Computing*, 7(2):67–73, 1993.

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

[7] D. Boneh, G. Durfee, and M. Franklin. Lower bounds for multicast message authentication. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 437–452, 2001.

[8] J. Buchmann, E. Dahmen, and M. Schneider. Merkle tree traversal revisited. In *Proceedings of the 2nd International Workshop on Post-Quantum Cryptography*, pages 63–78, 2008.

[9] H. Cam, S. Ozdemir, P. Nair, D. Muthuavinashiappan, and H. O. Sanli. Energy-efficient secure pattern based data aggregation for wireless sensor networks. *Computer Communications*, 29(4):446–455, 2006.

[10] R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on the Foundations of Computer Science*, pages

136–145, 2001.

[11] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: a taxonomy and some efficient constructions. In *Proceedings of the Conference of the IEEE Computer and Communications Societies*, volume 2, pages 708–716, 1999.

[12] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 494 – 503, 2002.

[13] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. In *Proceedings of The Second Annual International Conference on Mobile and Ubiquitous Systems*, pages 109– 117, 2005.

[14] H. Chan, V. Gligor, A. Perrig, and G. Muralidharan. On the distribution and revocation of cryptographic keys in sensor networks. *IEEE Transactions on Dependable and Secure Computing*, 2(3):233–247, 2005.

[15] H. Chan and A. Perrig. Efficient security primitives derived from a secure aggregation algorithm. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, pages 521–534, 2008.

[16] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 197–215, 2003.

[17] H. Chan, A. Perrig, and D. Song. Secure hierarchical in-network aggregation for sensor networks. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 278–287, 2006.

[18] D. Chaum, C. Crepeau, and I. Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 11–19, 1988.

[19] Y. Chen, I. Lin, C. Lei, and Y. Liao. Broadcast authentication in sensor networks using compressed bloom filters. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems*, pages 99–111, 2008.

[20] Cheng, Wang, and Liang. Zone-oriented byzantine agreement on zone-based wireless adhoc network. In *Proceedings of the Second International Conference on High Performance Computing and Communications*, pages 853–862, 2006.

[21] Y. Desmedt, Y. Frankel, and M. Yung. Multi-receiver/multi-sender network security: efficient

authenticated multicast/feedback. In *Proceedings of the Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 2045 – 2054, 1992.

[22] Q. Dong, D. Liu, and P. Ning. Pre-authentication filters: Providing DoS resistance for signature-based broadcast authentication in wireless sensor networks. In *Proceedings of ACM Conference on Wireless Network Security*, pages 2–12, 2008.

[23] V. Drabkin, R. Friedman, and M. Segal. Efficient byzantine broadcast in wireless ad-hoc networks. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 160–169, 2005.

[24] W. Du, J. Deng, Y. Han, and P. K. Varshney. A witness-based approach for data fusion assurance in wireless sensor networks. In *Proceedings of the IEEE Global Telecommunications Conference*, volume 3, pages 1435– 1439, 2003.

[25] W. Du, R. Wang, , and P. Ning. An efficient scheme for authenticating public keys in sensor networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 58 – 67, 2005.

[26] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. J. Strauss, and R. N. Wright. Secure multiparty computation of approximations. *ACM Trans. Algorithms*, 2(3):435 – 472, 2006.

[27] P. Feldman and S. Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput*, 26(4):873 – 933, 1997.

[28] K. B. Frikken and J. A. Dougherty. An efficient integrity-preserving scheme for hierarchical sensor aggregation. In *Proceedings of the first ACM conference on Wireless network security*, pages 68–76, 2008.

[29] G. Gaubatz, J. Kaps, and B. Sunar. Public key cryptography in sensor networks - revisited. In *Proceedings of the European Workshop on Security in Ad-Hoc and Sensor Networks*, pages 2–18, 2004.

[30] R. Gennaro and P. Rohatgi. How to sign digital streams. In *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, pages 180 – 197, 1997.

[31] C. Georgiou, D. R. Kowalski, and A. A. Shvartsman. Efficient gossip and robust distributed computation. *Theoretical Computer Science*, 347(1):130 – 166, 2005.

[32] J. Girao, M. Schneider, and D. Westhoff. CDA: Concealed data aggregation for reverse multicast traffic in wireless sensor networks. In *Proceedings of the IEEE International Conference*

on Communications, volume 5, pages 3044– 3049, 2005.

[33] A. Giridhar and P. R. Kumar. Towards a theory of in-network computation in wireless sensor networks. *IEEE Communications Magazine*, 44(4):98– 107, 2006.

[34] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 218–229, 1987.

[35] M. B. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 275 – 285, 2004.

[36] I. Gupta, R. van Renesse, and K. P. Birman. Scalable fault-tolerant aggregation in large process groups. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 433 – 442, 2001.

[37] V. Gupta, M. Millard, S. Fung, Y. Zhu, N. Gura, H. Eberle, and S. C. Shantz. Sizzle: A standards-based end-to-end security architecture for the embedded internet. In *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communication*, pages 247 – 256, 2005.

[38] K. Hildrum and J. Kubiatowicz. Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In *Proceedings of the 17th International Symposium on Distributed Computing*, pages 321–336, 2003.

[39] L. Hu and D. Evans. Secure aggregation for wireless networks. In *Proceedings of the Symposium on Applications and the Internet*, pages 384– 391, 2003.

[40] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 457– 458, 2002.

[41] P. Jadia and A. Mathuria. Efficient secure aggregation in sensor networks. In *Proceedings of the 11th International Conference on High Performance Computing*, pages 40–49, 2004.

[42] M. Jakobsson, T. Leighton, S. Micali, and M. Szydlo. Fractal merkle tree representation and traversal. In *Proceedings of the Cryptographers Track at the RSA Conference (CT-RSA)*, pages 314–326, 2003.

[43] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: Attacks and counter-

measures. In *Proceedings of First IEEE International Workshop on Sensor Network Protocols and Applications*, pages 293–315, 2003.

[44] J. Katz and A. Lindell. Aggregate message authentication codes. In *Proceedings of The Cryptographers Track at the RSA Conference (CT-RSA)*, pages 155–169, 2008.

[45] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, 2000.

[46] V. King, J. Saia, V. Sanwalani, and E. Vee. Scalable leader election. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 990 – 999, 2006.

[47] V. King, J. Saia, V. Sanwalani, and E. Vee. Towards secure and scalable computation in peer-to-peer networks. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 87–98, 2006.

[48] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382 – 401, 1982.

[49] F. T. Leighton, B. M. Maggs, and R. K. Sitaraman. On the fault tolerance of some popular bounded-degree networks. *SIAM J. Comput*, 27(5):1303 – 1333, 1998.

[50] A. Liu and P. Ning. TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks*, pages 245–256, 2008.

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

[52] B. Lu and U. Pooch. A lightweight authentication protocol for mobile ad-hoc networks. In *Proceedings of the International Conference on Information Technology: Coding and Computing*, volume 2, pages 546 – 551, 2005.

[53] 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*, pages 147 – 156, 2006.

[54] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131 – 146, 2002.

[55] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the ACM International Conference on*

Management of Data, pages 491 – 502, 2003.

[56] A. Mahimkar and T. Rappaport. SecureDAV: A secure data aggregation and verification protocol for sensor networks. In *Proceedings of the IEEE Global Telecommunications Conference*, volume 4, pages 2175– 2179, 2004.

[57] D. Malan, M. Welsh, and M. Smith. A public-key infrastructure for key distribution in TinyOS based on elliptic curve cryptography. In *Proceedings of IEEE Conference on Sensor and Ad hoc Communications and Networks*, pages 71– 80, 2004.

[58] A. Manjhi, S. Nath, and P. B. Gibbons. Tributaries and deltas: efficient and robust aggregation in sensor network streams. In *Proceedings of the ACM International Conference on Management of Data*, pages 287 – 298, 2005.

[59] M. Manulis and J. Schwenk. Provably secure framework for information aggregation in sensor networks. In *Proceedings of the International Conference on Computational Science and Its Applications*, pages 603–621, 2007.

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

[61] R. C. Merkle. A digital signature based on a conventional encryption function. In *Proceedings of the Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, pages 369 – 378, 1988.

[62] S. Miner and J. Staddon. Graph-based authentication of digital streams. In *Proceedings of the IEEE Symposium on Security and Privacy*, page 232, 2001.

[63] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 250 – 262, 2004.

[64] P. Ning, A. Liu, and W. Du. Mitigating DoS attacks against broadcast authentication in wireless sensor networks. *ACM Transactions on Sensor Networks*, 4(1):1–35, 2008.

[65] J. M. Park, E. Chong, and H. Siegel. Efficient multicast packet authentication using signature amortization. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 227–240, 2002.

[66] A. Perrig, R. Canetti, J. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In *Proceedings of the IEEE Symposium on Security and Privacy*,

pages 56–73, 2000.

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

[68] 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.

[69] B. Przydatek, D. Song, and A. Perrig. SIA: Secure information aggregation in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 255 – 265, 2003.

[70] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 73 – 85, 1989.

[71] R. Safavi-Naini and H. Wang. New results on multi-receiver authentication codes. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 527–541, 1998.

[72] M. Szydlo. Merkle tree traversal in log space and time. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 541–554, 2004.

[73] E. Upfal. Tolerating linear number of faults in networks of bounded degree. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 541–554, 1992.

[74] D. Wagner. Resilient aggregation in sensor networks. In *Proceedings of the 2nd ACM Workshop on Security of Ad-hoc and Sensor Networks*, pages 78 – 87, 2004.

[75] S. Wang, Y. Chin, and K. Yan. Byzantine agreement in a generalized connected network. *IEEE Transactions on Parallel and Distributed Systems*, 6(4):420–427, 1995.

[76] S. Wang, W. Yang, and C. Cheng. Byzantine agreement on mobile ad-hoc network. In *Proceedings of the IEEE International Conference on Networking, Sensing and Control*, volume 1, pages 52–57, 2004.

[77] C. K. Wong and S. S. Lam. Digital signatures for flows and multicasts. *IEEE/ACM Trans. Netw.*, 7(4):502–513, 1999.

[78] A. Wood and J. Stankovic. Denial of service in sensor networks. *IEEE Computer*, pages 54–62, 2002.

[79] Y. Yang, X. Wang, S. Zhu, and G. Cao. Sdap: a secure hop-by-hop data aggregation protocol for sensor networks. In *Proceedings of the 7th ACM International Symposium on Mobile Ad-hoc Networking and Computing*, pages 356–367, 2006.

[80] T. Yao, S. Fukunaga, and T. Nakai. Reliable broadcast message authentication in wireless sensor networks. In *Proceedings of the Workshops on Emerging Directions in Embedded and Ubiquitous Computing*, pages 271–280, 2006.

[81] Y. Yao and J. Gehrke. The COUGAR approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, 2002.

[82] F. Ye, H. Luo, S. Lu, and L. Zhang. Statistical en-route filtering of injected false data in sensor networks. *IEEE Journal on Selected Areas in Communication*, 23(4):839– 850, 2004.

[83] H. Yu. Secure and highly-available aggregation queries in large-scale sensor networks via set sampling. In *Proceedings of the Eighth International Symposium on Information Processing in Sensor Networks*, pages 1–12, 2009.

[84] L. Zhang. Fault tolerant networks with small degree. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 64–69, 2000.

[85] S. Zhu, S. Setia, S. Jajodia, and P. Ning. An interleaved hop-by-hop authentication scheme for filtering false data in sensor networks. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 259–271, 2004.

[86] S. Zhu, S. Xu, S. Setia, and S. Jajodia. LHAP: a lightweight hop-by-hop authentication protocol for ad-hoc networks. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, pages 749– 755, 2003.

[87] Zigbee Alliance. Zigbee specification document 053474r17, 2008.