Efficient Security Primitives Derived from a Secure Aggregation Algorithm *

Haowen Chan
Carnegie Mellon University, USA
haowenchan@cmu.edu

Adrian Perrig
Carnegie Mellon University, USA
perrig@cmu.edu

ABSTRACT

By functionally decomposing a specific algorithm (the hierarchical secure aggregation algorithm of Chan et al. [3] and Frikken et al. [7]), we uncover a useful general functionality which we use to generate various efficient network security primitives, including: a signature scheme ensuring authenticity, integrity and non-repudiation for arbitrary node-to-node communications; an efficient broadcast authentication algorithm not requiring time synchronization; a scheme for managing public keys in a sensor network without requiring any asymmetric cryptographic operations to verify the validity of public keys, and without requiring nodes to maintain node revocation lists. Each of these applications uses the same basic data aggregation primitive and thus have $O(\log n)$ congestion performance and require only that symmetric secret keys are shared between each node and the base station. We thus observe the fact that the optimizations developed in the application area of secure aggregation can feed back into creating more optimized versions of highly general, basic security functions.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*

General Terms

Security, Algorithms

Keywords

Sensor Networks, Data Aggregation, Broadcast, Authentication, Integrity, Public Key Management, Signatures

*This research was supported in part by CyLab at Carnegie Mellon under grant DAAD19-02-1-0389 from the Army Research Office, grant MURI W 911 NF 0710287 from the Army Research Office, and grant CAREER CNS-0347807 from the National Science Foundation, and by a gift from Bosch. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, Bosch, CMU, NSF, or the U.S. Government or any of its agencies.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CCS'08, October 27–31, 2008, Alexandria, Virginia, USA.
Copyright 2008 ACM 978-1-59593-810-7/08/10 ...$5.00.

1. INTRODUCTION

Sensor networks represent an interesting emerging applications area where distributed computation takes place in a well-defined network topology. One example of a common structured distributed computation is *data aggregation*. In the framework first defined in Tiny Aggregation (TAG) by Madden et al. [16], data aggregation occurs over a tree rooted at the base station; data is sent in a convergent pattern from the leaf sensor nodes to the root with internal nodes performing intermediate computations to summarize the data along the way. *Secure data aggregation* algorithms aim to ensure the integrity of the results computed by this distributed computation in the presence of a small number of malicious ("Byzantine") nodes which may be attempting to influence the result.

In this paper, we make the following high-level observation: because data aggregation represents a general class of distributed computation over a specific kind of structured network (e.g., a tree) utilizing a specific pattern of communication (e.g., to and from the base station via the tree topology), therefore, algorithms that provide integrity for data aggregation tend to contain, as part of their design, primitives for providing certain useful general integrity properties for those specific communication patterns over those topologies. These general integrity properties may be used to construct efficient security primitives specifically optimized to the topology or communication pattern for which the original aggregation application was designed. Hence, secure data aggregation is a useful central research area for security in resource-constrained structured networks, feeding useful ideas for the creation of both general primitives and specific application protocols.

We consider generalized applications of the secure hierarchical aggregation algorithm with distributed verification proposed by Chan et al. [3] and Frikken and Dougherty [7]. Specifically, we analyze the original algorithm by decomposing it into modular, generalized functionalities. One functionality of particular importance is the ability to efficiently generate and disseminate network-wide cryptographic *hash trees*. We call this generalization the "HT functionality". We show that the HT functionality is applicable to more general and broader problems than secure data aggregation. We describe efficient solutions to three specific open problems for tree-based sensor networks derived from the HT functionality: authenticated broadcast, public key management, and node-to-node message signatures providing authentication, integrity and non-repudiation. Each of these derived algorithms requires only one symmetric key per node shared with the base station and incurs $O(\log n)$ communication congestion per link in the network.

The remainder of the paper is structured as follows. We describe the relevant details of the original hierarchical secure aggregation framework in Section 2. We then provide a generalized functional decomposition of the algorithm in Section 3. The assumptions used

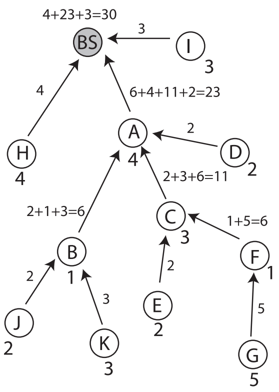
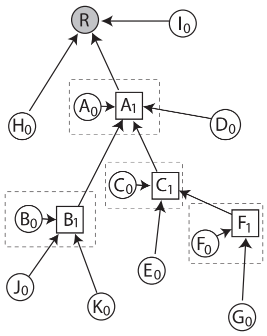


Figure 1: Standard tree-based SUM aggregation as described in TAG. Numbers next to nodes represent data/readings; Numbers and sums on edges represent intermediate sums sent from nodes to parents.



$$R = \langle 30; H[30||H_0||A_1||I_0] \rangle$$

$$A_1 = \langle 23; H[23||A_0||B_1||C_1||D_0] \rangle$$

$$C_1 = \langle 11; H[11||C_0||E_0||F_1] \rangle$$

$$F_1 = \langle 6; H[6||F_0||G_0] \rangle$$

$$G_0 = \langle 5; G \rangle$$

Figure 2: Non-optimized commitment tree for the aggregation of Figure 1, 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 representing the aggregate computation at X. On the right we list the labels of the vertices on the path of node G to the root.

in our application domains are summarized in Section 4. We then show how the general functionalities derived can be used to construct algorithms for broadcast authentication (Section 5), public key management (Section 6) and node-to-node message signatures (Section 7).

2. THE CPS ALGORITHM

In this paper we will focus on using the the algorithm proposed by Chan et al. [3] and improved by Frikken and Dougherty [7] to derive new algorithms for our security applications. Alternative secure aggregation algorithms with distributed verification may possibly be adapted for the applications listed in this paper – when describing each application, we will specify the functionalities required of the secure aggregation algorithm. This section provides a summary of the original algorithm (which we will call the CPS algorithm for brevity) and references to subsequent optimizations. For a detailed description with proofs, please refer to the original publications [3, 7]. Readers already closely familiar with the algorithm are encouraged to read Section 2.2 for some updated terminology before skipping to Section 3.

2.1 Problem Definition: Secure Aggregation

For some context to the secure aggregation problem, we review the standard aggregation framework as proposed in Tiny Aggregation (TAG) by Madden et al. [16]. We consider the example of computing a sum over all the sensor readings in the network. First, a spanning tree rooted at the base station is constructed over the network topology. All communications occur over the spanning tree subgraph; thus the remainder of the edges of the topology are ignored for this algorithm. Next, each sensor node that is a leaf in the spanning tree reports its 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 1.

The goal of the CPS algorithm is to guarantee the integrity of the SUM computation under the attacker model where a certain unknown subset of the nodes in the network is malicious and intends to skew the computed result (the original paper describes further generalizations of using SUM to compute other aggregates like counts, averages and quantiles). We assume that the base station shares a unique symmetric secret key with each sensor node. The correctness goal is that if no attacker is present and no errors occur in the algorithm, then the result is accepted; if an attacker tampers with the aggregation computation then the result (if any) is discarded and the algorithm reports that an adversary (or fault) must be present in the system. Specifically, the adversary is bounded by the algorithm such that any result it causes the system to accept is achievable by just reporting *legal* input values (i.e., values within a predetermined fixed range, e.g., room temperature sensors only report values between 0 to 50° C) at the malicious nodes that it controls; in other words, tampering with the aggregation mechanism gives the adversary no added ability to influence the set of accepted results. Furthermore, if the adversary attempts to cheat or disrupt the algorithm, its presence will be detected (although the exact malicious node is not pinpointed). Specific countermeasures may then be deployed to eliminate the adversary from the system; the specifics of the security reaction is outside the scope of the problem.

2.2 General Overview

A high level overview of the CPS secure hierarchical data aggregation scheme is as follows. The algorithm proceeds in four phases. Chan et al. and Frikken et al. originally used names for each phase which referred to the aggregation computation process [3, 7]; in anticipation of the generalization of the algorithm beyond data aggregation, we rename each phase with a more general label. The four phases and their short descriptions are as follows:

1. **Commitment Tree Generation Phase:** A structure similar to a hash tree, called a *commitment tree* (see Figure 2), is generated by the nodes in a distributed manner, committing to the set of inputs and the intermediate operations in the distributed aggregation computation. The root vertex of the commitment tree contains the aggregation result and also acts as an overall commitment to the operations and inputs leading to that result. Typically it is computed at the base station at the conclusion of this phase.
2. **Result Dissemination Phase:** The base station distributes the root vertex of the commitment tree using an authenticated

broadcast to the entire network.

3. **Distributed Verification Phase:** The nodes exchange information which allows each node to verify that their respective contributions were indeed correctly incorporated into each of the intermediate results computed during the aggregation process.
4. **Verification Confirmation Phase:** Once each node has successfully performed verification, it must notify the base station of success. Verification success confirmation messages are generated at each node and efficiently aggregated towards the base station. If the base station detects that all nodes succeeded in verification then the result is accepted, otherwise it is rejected.

2.3 The Commitment Tree

The foundational data structure of the CPS secure data aggregation algorithm is the commitment tree. In this section we describe this structure in more detail, describing the operation of the four phases with reference to the data structure. For clarity, we first focus on the basic (non-optimized) version of the commitment tree, which has a structure reflecting the network topology. In Section 2.4 we briefly discuss optimizations by Frikken and Dougherty which bound the depth of the commitment tree to $O(\log n)$ where n is the total number of leaves in the tree [7].

For clarity, we focus on a general simplification of the commitment tree and omit essential details in the original CPS publication such as nonces, node counts, and complement sums.

Figure 2 shows an example of how a basic commitment tree can be constructed. Figure 1 shows the logical network topology as well as the aggregation operations. For clarity, we will refer to the *vertices* of the commitment tree and the sensor *nodes* in the topology. Comparing Figures 1 and 2 side by side, it is apparent that the commitment tree follows the shape of the network topology. Each commitment-tree vertex is a hash-based commitment to the aggregation inputs and result occuring at the corresponding node in the topology. For example, vertex C_1 contains a hash on the result (i.e., 11) computed at node C as well as the set of inputs used to compute this result (i.e., commitment-tree vertices C_0, E_0, F_1).

2.3.1 Commitment Tree Generation

In the *commitment-tree generation phase*, the commitment tree is constructed from the bottom up in a distributed fashion, with new internal vertices added on each aggregation operation which are then transmitted on to the aggregating node's parent node in the network topology. The leaf vertex of the commitment tree for a node with ID i contains its input to the computation x_i as well as its identity i:

$$v_i = \langle x_i; i \rangle$$

For example, in Figure 2, sensor node G constructs a leaf vertex consisting of its input value x_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).

Each internal (non-leaf) sensor node i in the topology receives from each of its children a commitment tree vertex. The parent node i then performs its own aggregation operation over its own leaf vertex and the vertices supplied by its children, and generates a new internal vertex in the commitment tree.

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

Where u_i is an internal vertex created by node i, y_i is the result of the intermediate aggregation operation performed on the data contained in the commitment tree vertices $u_1, \cdots, u_k$ received from the children of i (or generated by i itself). For example, in Figure 2, the sensor node A performs an aggregation over the received internal vertices B_1 and C_1 and the leaf vertex D_0 as well as its own leaf vertex A_0. Based on the received inputs, it generates the aggregated result value 23 (as the sum of the values from all its children and itself) and commits to the set of inputs used in the computation, by computing a hash over $23||A_0||B_1||C_1||D_0$. The result is a new internal vertex in the commitment tree which is the parent of all the vertices received by A. Once each node i has computed all its internal commitment 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.

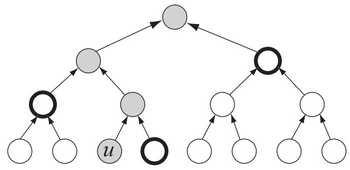


Figure 3: Off path vertices of vertex u

2.3.2 Result Dissemination

At the conclusion of the aggregate commit phase, the base station disseminates the root vertex of the commitment tree via an authenticated broadcast to all nodes in the network. It may also include a unique nonce N in the broadcast message; this nonce is used in the verification confirmation phase.

2.3.3 Distributed Verification

Each node is then required to verify that its own contribution to the network-wide aggregation computation was incorporated correctly in the commitment tree. Specifically, the node must check that its leaf vertex is indeed a descendant of the root vertex that was broadcast by the base station; furthermore, it must check the correctness of all the aggregation operations computed which involve its input value as an upstream input. To check this, each node must recompute the sequence of commitment tree vertices between its leaf vertex and the root. For example, node G in Figure 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. The off-path vertices of a tree vertex u are the sibling vertices of all nodes on the path from u to the root of the tree. A graphical depiction is shown in Figure 3. In Figure 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 in the network topology broadcasts the set of commitment-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 1.

2.3.4 Verification Confirmation

Once each node has successfully performed its respective verifications, they have to notify the base station of their success. Each node performs this operation by forwarding a specific *verification confirmation message* which is the message authentication code over a specific "OK" string and a nonce N specific to the current query, computed using the secret key shared between the node and the base station: $M_i = \mathrm{MAC}_{K_i}(N||OK)$. Individually forwarding each of these confirmation messages would be too costly in terms of communication; hence the algorithm makes use of the fact that the base station only wishes to know if *all* of the sensor nodes have released their respective confirmation messages. Thus the confirmation messages are aggregated using the XOR operation: each internal node in the topology waits to receive the confirmation messages from each of its children, then computes the XOR of all the 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 the confirmation messages, i.e., $\text{MAC}_{K_1}(N||OK)\oplus\cdots\oplus\text{MAC}_{K_n}(N||OK)$. 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 succesfully perform distributed verification. The base station can then compare this expected value with the aggregated confirmation message that it receives from the network: if the values are the same then it can accept the aggregation result; if not then it discards the aggregation result.

2.4 Optimizations

In aggregation algorithms (as for most sensor network applications), we measure communication overhead by *congestion*, which is defined as the worst-case heaviest communication load on any single link in the network. The reason for this metric is that for sensor networks communication is costly because it drains battery power, which affects node lifetimes. Algorithms with low congestion tend to spread out the communication load, leading to a longer lifetime until the first sensor node death.

The limiting step in the congestion bound is the distributed verification phase where each sensor node i must verify the path in the commitment-tree from each of the vertices it created to the root vertex r. To do this, node i must collect (or recompute) all the child vertices of each vertex that is an ancestor of each of the vertices that node i is verifying. Thus, the height and degree of the commitment tree affects the cost of the algorithm. Fortunately, the structure of the commitment tree does not necessarily have to follow the shape of the network topology. Aggregation operations can be re-ordered such that the resultant commitment trees formed are well-balanced. The original CPS algorithm balanced its commitment trees using a rule which ensured that only trees of similar size could be joined by creating a parent vertex over them. This resulted in a congestion bound of $O(\log^2 n)$ [3]. Frikken and Dougherty greatly improved this optimization by using a more sophisticated heuristic which ensured that all the new vertices created by a sensor node always shared the same verification path [7]. This improved the congestion bound to $O(\log n)$.

Further optimizations may be possible in future work to further reduce the commitment tree congestion overhead; the applications described in this paper are viable independently of such optimizations.

3. FUNCTIONALITY DECOMPOSITION

In this section we extract and analyze the general properties provided to the protocol designer by the CPS algorithm.

To understand the usefulness of the CPS algorithm, we decompose its four phases into three distinct functionalities. Phase 1 and 3, the commitment tree generation and distributed verification phases, correspond to the ability to efficiently generate commitment trees and disseminate them such that each sensor node which contributed a vertex can verify that vertex is properly included in the commitment tree. We call this functionality the CT (for "commitment tree") functionality. Phase 2, the result dissemination phase, corresponds to an authenticated broadcast functionality. Phase 4, the verification confirmation phase, corresponds to an

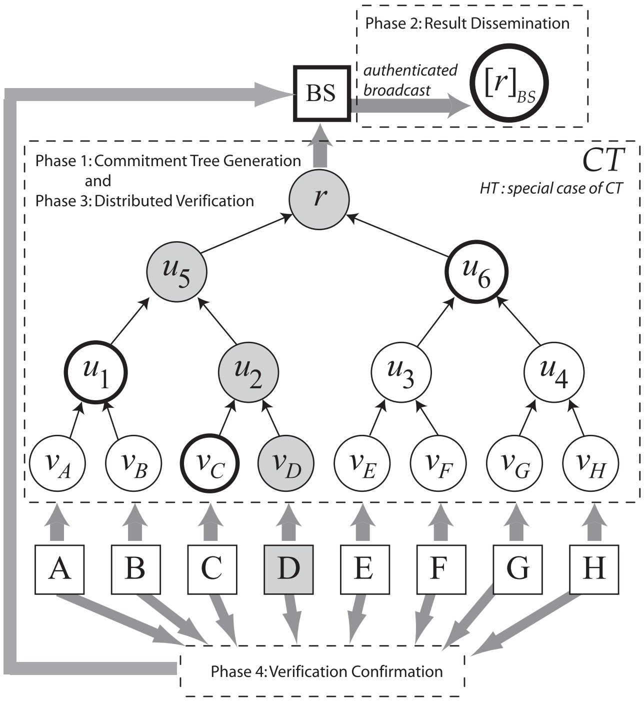


Figure 4: Functional decomposition of the CPS algorithm. Dashed boxes represent phases/functionalities in the original CPS scheme, with the Commitment Tree Generation and Distributed Verification phases combined to form the CT functionality. Square lettered nodes represent sensor nodes and round vertices represent hash tree vertices. Thick arrows represent communication to/from the functionalities, thin arrows represent hash function inputs. The verification path of node D is highlighted by the grey shaded nodes; D receives all nodes outlined in bold.

VERIFICATION CONFIRMATION FUNCTIONALITY
Inputs: (from each sensor node i)
Confirmation message: C_i
Outputs:
To Base Station:
XOR of all confirmation messages: $C_1 \oplus C_2 \oplus \cdots \oplus C_n$

Table 1: Interface of the Verification Confirmation Functionality

efficient network-wide acknowledgement functionality. The three functionalities are illustrated in relation to one another in Figure 4. Note that since each of these functionalities is network-based, they are considered *untrusted* functionalities, i.e., it is up to any nodes communicating with these functionalities to perform the necessary verifications of integrity based on the authenticating information returned by the functionalities.

To more rigorously define these modular functionalities, we define their respective interfaces (i.e., parameters, inputs and outputs) in turn. We skip discussion of the functionality for the result dissemination phase: it is simply authenticated broadcast which is well-understood.

The verification confirmation functionality is an efficient method for collecting a set of acknowledgements; Section 2.3.4 describes the verification confirmation phase of CPS which performs exactly

CT FUNCTIONALITY
Inputs: (from each sensor node i)
Data Value x_i
Computes:
Tree via the following:
Leaf vertex v_i (one per node i): $v(x_i, i)$
Internal vertex u_0 with child vertices $u_1, \ldots, u_k$: $u(u_1, \ldots, u_k)$
Outputs:
To Base Station: Commitment tree root vertex r
To each sensor node i: Off path vertices for v_i

Table 2: Definition of the CT Functionality

this function. Table 1 shows the interface of this functionality.

The CT (for "commitment tree") functionality is the ability to form and disseminate arbitarily defined trees. It is derived as a generalization of the process of forming and disseminating the specific type of commitment-tree defined in the CPS algorithm. After defining the general CT functionality, we will instantiate it to operate on simple hash trees; this is the more practical HT functionality we will use for the practical applications in subsequent sections.

The CT functionality is defined as follows: consider a network consisting of some number of nodes and a trusted base station (or central authority). Define an untrusted functionality CT which performs the following function: it queries each of the nodes in turn, receiving from each node i a data value x_i. Functionality CT then constructs a tree. First a leaf vertex v_i is created for each node i; then CT repeatedly adds internal vertices until a tree is created over the given leaves. The shape of the tree can be freely decided by the algorithm implementing the CT functionality. The internal vertices are computed using the fixed parent vertex creation rule $u()$, i.e. for each internal vertex u_0 with child vertices $u_1, \ldots, u_k$, we compute $u_0 = u(u_1, \ldots, u_k)$. Once the tree has been constructed, functionality CT reports the root r of the hash tree to the base station. Then, for each node in the network, CT provides enough information for the node to verify the inclusion of its provided leaf values in the tree; specifically, each node i receives the siblings of all the vertices in the commitment tree from their leaf vertex v_i to the root vertex r. The CT functionality is summarized in Table 2.

Note that since the nodes do not trust the functionality CT, typically they need to have some prior knowledge of what root vertex value to expect (i.e., r) when performing distributed verification. One method of providing this knowledge is for the base station to disseminate the value r with an authenticated broadcast (i.e., via the result dissemination phase of CPS). In certain applications (e.g., Section 5) the value r is already known prior to the invocation of the CT functionality, hence authenticated broadcast is not necessary.

For reference, the specific parameterization of the CT functionality as used in the (simplified) CPS algorithm as described in Section 2 is shown in Table 3. We make the following observation:

Observation 1 *The CPS algorithm contains an $O(\log n)$ - congestion implementation of the general CT functionality.*

This is because the parameterization of the CT functionality only changes how the commitment tree vertices are defined; for any such definition we can replace the standard definitions of CT_{CPS} directly without affecting the operation of the algorithm.

In subsequent applications, we move away from the function-computation aspect of the CT functionality and use it purely for its

CT_{CPS} FUNCTIONALITY
Inputs: (from each sensor node i)
Data Value x_i
Computes:
Tree via the following:
Leaf vertex v_i (one per node i): $\langle x_i; i \rangle$
Internal vertex u_0 with child vertices $u_1, \ldots, u_k$ containing values $y_1, \ldots, y_k$ respectively: $\left\langle y_0 = \sum_{j=1}^{k} y_j; H[y_0 \| u_1 \| \cdots \| u_k] \right\rangle$
Outputs:
To Base Station: Commitment tree root vertex r
To each sensor node i: Off path vertices for v_i

Table 3: The instance of CT implemented in CPS

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

Table 4: Definition of the HT Functionality

commitment properties. In other words, we use a version of CT that does not perform data aggregation computations and simply computes a conventional hash tree instead of a commitment tree. We call the CT functionality parameterized in this fashion, the HT (hash-tree) functionality.

As mentioned, the commitment tree generation and distributed verification phases of the CPS algorithm can realize the HT functionality simply by changing the definition of the commitment-tree vertices to the ones in Table 4. All the other operations of Section 2.3.1 and 2.3.3 remain unchanged; in particular the optimizations of Section 2.4 still apply; the hash trees created by this realization of the HT functionality are binary trees of $O(\log n)$ height, and the cost of distributed verification is $O(\log n)$ congestion.

Observation 2 *Given the HT functionality, the network can efficiently generate and disseminate hash trees.*

In other words, HT functionality gives us the ability to construct protocols which use network-wide authenticated hash trees with very little overhead. The most common use of a hash tree is as a *batched signature* by the base station over a set of values: this functionality will drive most of the applications for the subsequent sections.

3.1 CPS Without Authenticated Broadcast

Our decomposition of the CPS algorithm into four phases shows that the authenticated broadcast of Phase 2 is not intrinsically part of the CT (or HT) functionality of Phase 1 and 3. Specifically, consider the removal of the authenticity property from Phase 2, i.e., by using normal broadcast instead of authenticated broadcast to disseminate the root r of the commitment tree. Without authentication, the distributed verification of Phase 3 may be performed on a falsified root hash value value $r' \neq r$. However, as long as the protocol eventually aborts (instead of accepting) on the injection of a falsified root hash value, the correctness of the overall algorithm remains unchanged. More precisely, instead of using authenticated broadcast to provide integrity for the root hash value in Phase 2, we can defer this check to later phases. In particular, we can easily make the success of Phase 4 dependent on each node receiving the correct root hash value r. This yields a version of CPS that does not require an authenticated broadcast primitive. Subsequently, in Section 5 we show that in some applications it is possible to cause Phase 3 to abort if any legitimate node is given a falsified root hash value: this itself yields an authenticated broadcast algorithm.

The modification to the CPS algorithm to remove the need for authenticated broadcast involves modifying the "verification successful" message in the final phase of the algorithm to include data about the root vertex that the verification process was computed against. Hence, if the wrong root vertex was used, then the "verification successful" message simply becomes invalid and the base station will (correctly) discard the aggregation result.

The specific modification is as follows. After the base station receives (or computes) the root vertex of the commitment tree, it disseminates the value using conventional (unauthenticated) broadcast. The sensor nodes then carry out the distributed verification phase as usual against the (possibly invalid) root vertex value that they receive. In verification confirmation phase, each node then replies with an authentication code that is efficiently aggregated using XOR and delivered to the base station. In the original protocol, each node i releases an authentication code $\text{MAC}_{K_i}(N||OK)$ where K_i is the key that i shares with the base station and N is a nonce associated with the aggregation query. We modify this message to also include the value of the root vertex that was used in the verification, i.e. node i will reply with $\text{MAC}_{K_i}(N||OK||r)$ where r is the root vertex that the successful verification was computed against. The rest of the protocol continues unchanged, i.e. the base station collects the XOR of all the authentication codes and compares it with the value that it expects.

An intuition for the correctness of the scheme is as follows. The reason why r needs to be authenticated in the original protocol was to prevent the adversary from injecting some arbitrary $r' \neq r$ which may cause certain nodes to fail to detect that their values were not correctly incorporated in the commitment tree sent to the base station; in other words, these nodes could be fed an alternative, false commitment tree with root r' which contained their input values but which has no relation with the actual commitment tree reported to the base station. The modified authentication code message removes the ability to perform this attack since any node verifying against the wrong root value r' will also (with high probability under standard MAC unforgeability assumptions) release the wrong authentication code reply $\text{MAC}_{K_i}(N||OK||r) \neq \text{MAC}_{K_i}(N||OK||r')$. This will cause the base station to reject the (possibly incorrect) aggregation result.

4. ASSUMPTIONS

We are now ready to start deriving applications from the HT functionality. First, we state the operating assumptions of the new protocols.

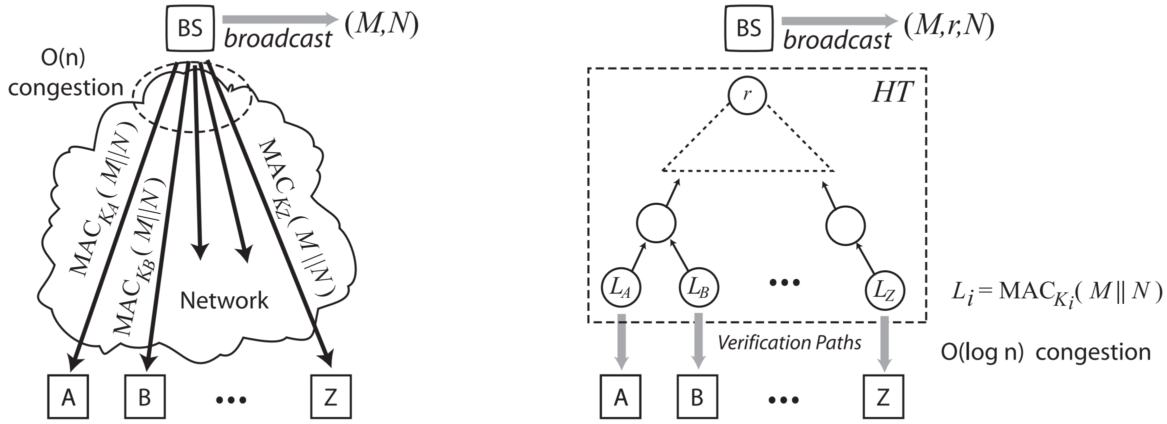
- **Preloaded Keys.** We require only that each sensor node share a unique symmetric key with the trusted base station.
- **Limited Resistance to Denial-of-Service (DoS) Attacks.** In our protocols we will mainly be concerned with data integrity rather than availability. Our definition of correctness is based on a tight definition of soundness and a looser interpretation of completeness, as follows. Soundness: data is accepted only if it is authentic. Completeness: if no adversary is present and no messages are lost, then the data must be accepted. In particular we allow the case where the adversary is present and the data is authentic, but the protocol rejects it. In practical terms this means we inherit the original CPS algorithm's vulnerability to denial-of-service, where a single malicious node can cause the protocol to abort (without a result) by behaving badly. The argument for this is that in each disrupted round of the protocol, the adversary nodes must deviate from the protocol in some way (by either sending the wrong messages or not sending a message when expected), and if this behavior persists then it is a clear indication of an error condition and an out-of-band remedial action can be taken. One particular form of DoS that we do attempt to prevent is long-lived DoS, where misbehavior in a single round causes the protocol to stop functioning for an extended number of subsequent rounds. This will be noted wherever appropriate.
- **Fixed known network topology.** We assume that the sensor network is mostly static, with a topology that is a-priori known to the base station, and changes in the network topology are sufficiently infrequent to be ignored in the estimation of communication and memory overheads. This appears to be true of many modern sensor network applications such as building and home instrumentation and automation. We assume that only reliable links are included in the network topology and retransmissions are performed to a sufficient degree that message loss is a negligible factor.

5. AUTHENTICATED BROADCAST

In this section we show how the HT functionality of the CPS algorithm 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.

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 MAC using the key they share, e.g., $\text{MAC}_{K_i}(M||N)$ (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). 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(a)).

Alternatively, we can use a hash tree to "batch" the entire set of MACs into a single structure. Specifically, construct a hash tree whose set of leaves is the set of MACs of M using each of the keys shared with the nodes in the network, i.e., $\{\text{MAC}_{K_1}(M||N), \ldots, \text{MAC}_{K_n}(M||N)\}$. 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 $\text{MAC}_{K_i}(M||N)$ in the past, exhibiting the value r suffices as proof of the ability to independently compute $\text{MAC}_{K_i}(M||N)$ since r was computed via a sequence of collision-resistant hash functions evaluated with $\text{MAC}_{K_i}(M||N)$ as an input. In other words, it is computationally infeasible for an adversary that does not know K_i or $\text{MAC}_{K_i}(M||N)$ to produce a new pair M', r', N' in such a way that $\text{MAC}_{K_i}(M'||N')$ is a leaf vertex in some hash tree with root r' (the adversary has to break either the collision resistance of the hash function or the unforgeability of the MAC). Note that in such a construction, each node only needs to verify the inclusion of its own MAC as a leaf in the hash tree; other MACs are irrelevant.



(a) Using unicast

(b) Using the HT functionality

Figure 5: Disseminating per node MACs: Naive method and HT method

Hence, the HT functionality is perfectly suited for efficiently generating and disseminating exactly such a hash tree. The details are shown on Figure 5(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, it 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 $\{\mathrm{MAC}_{K_1}(M||N), \ldots, \mathrm{MAC}_{K_n}(M||N)\}$ where K_i is the key shared with node i and N is a nonce that is never re-used (e.g., a sequence number). 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 releases $\mathrm{MAC}_{K_i}(M||N)$ to the HT functionality. The HT functionality recomputes the hash tree and releases the relevant verification information to each node i allowing it to verify that $\mathrm{MAC}_{K_i}(M||N)$ is a leaf of the hash tree with root r. If the node successfully verifies this then it can accept M as authentic.

The specifics of the broadcast algorithm as it relates to the CPS algorithm are evident from the way the CPS algorithm implements the HT functionality. Specifically, we assume that the topology is static and known to the base station: this allows the base station to anticipate exactly the hash tree that will be generated. The base station then simulates (internally) the commitment tree generation phase (where the leaf vertex associated with each node is $\mathrm{MAC}_{K_i}(M||N)$), and derives the root of the commitment tree r, and broadcasts the triplet (M, r, N). The nodes each check that the nonce N has not been previously used, then collaborate to perform the commitment tree generation phase and the distributed verification phase to verify that their respective MACs were included in the computation of r. Note that there is no need for a root vertex broadcast phase because the root vertex r was already known from the original broadcast. If the nodes successfully complete distributed verification, then they accept M as authentic. The final acknowledgement phase is optional; its inclusion can help the base station detect protocol failure due to malicious injection, node error or message loss, but does not affect correctness. Omitting this phase does not allow a node to accept an inauthentic triplet (M', r', N'). The algorithm is summarized in Algorithm 1.

Algorithm 1 Authenticated Broadcast using HT Functionality

Input: Nonce N not previously used; Message M

1. Base station simulates the operation of HT on the leaf vertex definitions $L_i = \mathrm{MAC}_{K_i}(M||N)$, 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 $L_i = \mathrm{MAC}_{K_i}(M||N)$.
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 L_i is a leaf vertex in the hash tree with root r.
8. If verification successful, node i accepts M.
9. (Optional) Base station can request a network-wide ACK by implementing verification confirmation functionality over confirmation messages $C_i = \mathrm{MAC}_{K_i}(M||r||N||ACK)$ where ACK is a unique identifier indicate broadcast authentication success.

5.1 Analysis and Refinements

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 $\mathrm{MAC}_{K_i}(M'||N')$ (since the MAC is hard to forge and K_i is unknown to the adversary), the adversary is computationally unlikely to be able to (a priori) deduce an r' that is the root of a hash tree containing $\mathrm{MAC}_{K_i}(M'||N')$. On the other hand, once a triple (M', r', N') with a valid nonce N' is received by node i, the node will release $\mathrm{MAC}_{K_i}(M'||N')$ 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. 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 MAC 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 huge 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 presents 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 total 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 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. Obviously a *continuous* DoS by a stubborn adversary in every round of broadcasts can still incapacitate the broadcast mechanism; as discussed in Section 4 we assume that such persistent adversarial behavior can be addressed out-of-band.

Like all protocols in this paper, the overhead of this protocol is a bound of $O(\log n)$ congestion on all links in the network. Furthermore, it is the only known broadcast authentication scheme that is efficient and works using only unique symmetric keys shared between the base station and the nodes and which does not require time synchronization.

6. PUBLIC KEY MANAGEMENT

With the authenticated broadcast primitive described in Section 5, we now have access to 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. The general formulation of the problem is as follows: suppose that each node i has a label L_i which is known to itself and to the base station. Our goal is to be able 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).

6.1 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 PK Management

We show that public key dissemination can be performed similarly to the process described in Section 5. The general primitive is very similar to the process described in Section 5. 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 (if so, the protocol rounds of the authenticated broadcast may be merged with the protocol rounds of this section; the details are straightforward and are omitted for brevity). 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 receipt of this broadcast is performed using the verification confirmation functionality. This ensures that all nodes receive the most up-to-date root vertex r. Subsequently, each node releases its L_i. Using these as 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

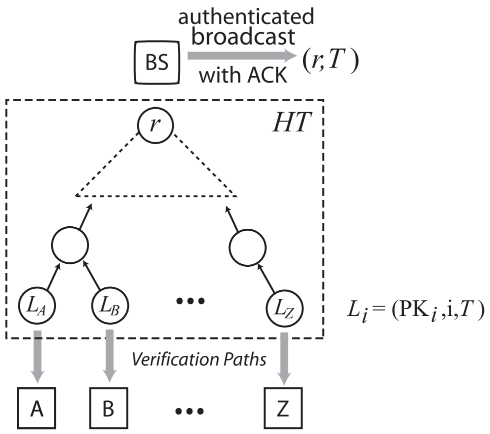


Figure 6: Use of HT as a Public Key distribution primitive

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 verification confirmation functionality is performed. The algorithm is summarized in Algorithm 2.

Algorithm 2 Public Key Dissemination using HT Functionality

Input: Nonce T not previously used; 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 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.

6.3 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.

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.
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.

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 CPS 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.

6.4 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 even 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.

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 (univerally trusted) base station.

The 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 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 nonrepudiation property (given the signature as evidence, i cannot deny that it was responsible for L_i).

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.

7.1 Algorithm Description

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 is changed. 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 self-validates 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 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 reports its L_i to the HT functionality, 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 is pre-image resistant. Note that the method of Section 5 can be used for the authenticated broadcast. The HT functionality also provides the requisite information to the sensor nodes for each node to 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 $\text{MAC}_{K_i}(r||s_b||h(N')||OK)$. 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 $\text{MAC}_{K_i}(r||s_b||N'||OK)$ 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.

7.2 Analysis

A proof sketch of unforgeability follows. Suppose for contradiction that an adversary has a non-negligible probability of being able to forge a message $L_i' \neq L_i$ purportedly from some legiti-

Algorithm 3 Signature Scheme using HT Functionality

Input: Replay-resistant messages L_i for each node i

Input: 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 L_i is a leaf vertex in the hash tree with root r via its fixed, known verification path.
6. Verification confirmation functionality (with $C_i = \text{MAC}_{K_i}(r||s_b||h(N')||OK)$) used to check that all nodes succeeded in verification. If not, stop.
7. Base station broadcasts N'
8. Upon receipt of N', nodes store r as being usable for authentication.
9. Verification confirmation functionality (with $C_i = \text{MAC}_{K_i}(r||s_b||N'||OK)$) is used to ensure all nodes received the broadcast. If not, stop.

Message Authentication Procedure:

Input: 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.

mate node i that successfully passed the checks of some legitimate node j. Let r be the hash tree root computed by j. Since j accepted the message authentication, the base station must have released N', $\langle r, s_b, h(N')\rangle$ in a prior authenticated broadcast from the base station. This means that (assuming that the nonce N' was not repeated, and MAC_{K_i} is a MAC whose forgeability is negligible) in some prior execution of the algorithm, node i has (with almost-certainty) 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. Since $L_i' \neq L_i$ this implies a hash collision somewhere in the verification path that j computed and the verification path that i computed (i.e. the two sequences are the same length, but started with different pre-images and resulted in the same image r). This implies that the adversary was able to engineer a hash collision with non-neglible probability.

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.

7.3 Applications

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.

7.3.1 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 [20]). 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 sign a much larger number of messages without byte-length constraints. We describe the details below.

7.3.2 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. [22]. 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.

7.3.3 Distributed Node Revocation.

One application of this authentication primitive is in the distributed revocation protocol of Chan, Gligor, Perrig and Muralidharan [1]. 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 [2]). 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 x without needing any additional preloaded information except for the single key x 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 7.3.1. 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.

8. RELATED WORK

Many secure data aggregation mechanisms for sensor networks have been proposed [3, 5, 7, 10, 11, 17, 19, 23–25], we review the CPS and Frikken and Dougherty's scheme that the mechanisms in this paper build on in Section 2.

Since we are not aware of other work that studies uses of secure data aggregation mechanisms to applications other than data aggregation, we discuss other work that addresses authenticated broadcast, public key management, and signature schemes for sensor networks.

In the area of broadcast authentication for sensor networks, Perrig et al. propose μTESLA [22], which unfortunately requires lose time synchronization. Improvements to μTESLA have been proposed, but they all require lose time synchronization [13]. Luk et al. propose families of broadcast authentication mechanisms [15], but the communication overhead of their one-time signature schemes can be quite substantial.

To provide resistance against computational DoS attacks for signature-based broadcast authentication in sensor networks, Ning et al. propose several mechanisms [4, 21]. Fortunately, our approaches for authentication and signature are inherently robust against computational DoS attacks.

Several works also target the problem of preventing the injection of false information into the sensor network, for example the work by Ye et al. [25] or Zhu et al. [26]. The problem we consider in this work is orthogonal.

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 distribution of node certificates [6, 14]. By updating the hash trees they provide a mechanism to revoke entities. Chan et al. develop mechanisms for node revocation in sensor networks [1]. However, their approach is not applicable for public key management.

Many researchers have studied the problem of efficient sensor network signatures, for example Liu and Ning [12], Malan et al. [18], Gupta et al. [9], and Gaubatz et al. [8]. 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.

9. CONCLUSION

We describe how the HT functionality encapsulated in the CPS algorithm for secure data aggregation is useful for developing a variety of useful and efficient security applications. In particular, we show fast and efficient primitives for broadcast authentication, public key management, and node-to-node signatures, each of which has important properties superior in some way to the current best known protocols in the literature. The reason for this performance is because they directly inherit, from the original secure data aggregation protocol, specific optimizations that work best in the structured tree network on which they operate.

These results highlight the significance of the secure data aggregation problem. As data aggregation represents a general description of the specific distributed computation pattern common in sensor networks, secure algorithms for this problem also represent secure versions of the communication and computation patterns that are most useful in sensor networks. Hence, secure data aggregation should not be considered simply a secure version of a sensor net application, but may be recognized as an important foundational problem closely related to the ideal of efficient secure computation and communication in structured networks.

10. ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their helpful comments and suggestions.

11. REFERENCES

[1] 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 (TDSC)*, 2(3):233–247, 2005.

[2] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *IEEE Symposium on Security and Privacy*, May 2003.

[3] 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*, Oct. 2006.

[4] 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 (WiSec)*, Apr. 2008.

[5] 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*, 2003.

[6] W. Du, R. Wang, , and P. Ning. An efficient scheme for authenticating public keys in sensor networks. In *Proceedings of Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 58–67, May 2005.

[7] K. B. Frikken and J. A. Dougherty. An efficient integrity-preserving scheme for hierarchical sensor aggregation. In *Proceedings of ACM Conference on Wireless Network Security (WiSec)*, pages 68–76, Apr. 2008.

[8] G. Gaubatz, J. Kaps, and B. Sunar. Public keys cryptography in sensor networks - revisited. In *Proceedings of European Workshop on Security in Ad-Hoc and Sensor Networks (ESAS)*, 2004.

[9] 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 (PerCom)*, 2005.

[10] L. Hu and D. Evans. Secure aggregation for wireless networks. In *Workshop on Security and Assurance in Ad hoc Networks*, 2003.

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

[12] A. Liu and P. Ning. TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In *Proceedings of Conference on Information Processing in Sensor Networks (IPSN), SPOTS Track*, Apr. 2008.

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

[14] D. Liu, P. Ning, S. Zhu, and S. Jajodia. Practical broadcast authentication in sensor networks. In *Proceedings of Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous)*, pages 118–129, July 2005.

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

[16] 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.

[17] A. Mahimkar and T. Rappaport. SecureDAV: A secure data aggregation and verification protocol for sensor networks. In *Proceedings of the IEEE Global Telecommunications Conference*, 2004.

[18] 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 (SECON)*, Oct. 2004.

[19] 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 (ICCSA)*, Aug. 2007.

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

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

[22] 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.

[23] 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*, 2003.

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

[25] 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 ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2006.

[26] 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 IEEE Symposium on Security and Privacy*, pages 259–271, May 2004.