Secure Code Distribution in Dynamically Programmable Wireless Sensor Networks

Jing Deng, Richard Han and Shivakant Mishra

Department of Computer Science
University of Colorado at Boulder

Technical Report CU-CS-1000-05

December 2005

Secure Code Distribution in Dynamically Programmable Wireless Sensor Networks

Jing Deng, Richard Han and Shivakant Mishra
Department of Computer Science
University of Colorado, Campus Box 0430
Boulder, CO 80309-0430
Email: {jing,rhan,mishras}@cs.colorado.edu

Abstract

For deployed wireless sensor networks, reprogramming sensor nodes through the wireless channel is an important capability. To avoid reprogramming false or viral code images, it is important to make sure that each sensor node can securely receive its code image through the wireless channel. Public key schemes based on elliptic curve cryptography are feasible in wireless sensor networks, yet are still very expensive in terms of memory and CPU consumption. This paper explores hybrid mechanisms that combine the speedy verification of packetized code allowed by hash-based schemes with the strong authenticity provided by a public key scheme. Based on this idea, three schemes for secure code propagation are proposed. The chain based scheme works best when packets are received in the order they are sent with very few losses. The hash tree based scheme allows nodes to authenticate packets and verify their integrity quickly, even when the packets may arrive out of order. Finally, the hybrid scheme combines the advantages of the first two schemes, and further reduces memory consumption and number of public key operations that a node has to perform. Simulation shows that the proposed secure reprogramming schemes add only a modest amount of overhead to a conventional non-secure reprogramming scheme, namely Deluge, and are therefore feasible and practical in a wireless sensor network.

1 Introduction

Applications of wireless sensor networks (WSNs) are becoming increasingly diverse, ranging from habitat monitoring [16] to indoor monitoring of semiconductor fabrication processes [3], and from counter-sniper localization on battlefields [21] to search and rescue operations [11]. Many of these applications require remote reprogramming of sensor nodes through the wireless channel for efficient sensor network management, i.e. for patching buggy code, changing run-time parameters, or installing new applications and unanticipated features. Manual reprogramming of sensor nodes is impractical for large in situ deployments, because of the scale of such deployments and the physical inaccessibility of certain sensor nodes, which can be deployed in hostile and/or rugged terrain.

A typical example of dynamically reprogramming a WSN is shown in Figure 1. In this example, the base station contains a code image that needs to be propagated to the sensor nodes. This code image is typically split into multiple pages. In general, the pages are reliably disseminated outwards

in a hop-by-hop fashion from the base station. Depending on the protocol, there may be additional signalling across each hop to specify which pages have been received and which pages are still desired.

For example, the Deluge code propagation protocol [12] implements a three phase Advertise-Request-Data polite gossip protocol, in which data is only pushed by a sender upon receiving an explicit request for data from an immediate neighbor. This pull-based approach has the benefit that no more data than is necessary is transmitted. Deluge benefits from a soft state design in that, as data is reliably flooded hop by hop throughout the network, the failure and loss of state at a single node does not affect more than the minimum number of nodes downstream. If there are other paths for code to propagate to downstream nodes, then the code will reach downstream, bypassing the failed node.

For certain WSN applications, securing the process of dynamic reprogramming is essential. For example, code updates in military applications must be authenticated to avoid the download of malicious code into deployed sensor nodes. Code updates in commercial applications such as monitoring of semiconductor fabrication labs and oil tankers must be verified to ensure that malicious code does not halt profit-making production or otherwise cause catastrophic damage [3]. In addition, applications that require privacy and anonymity to be preserved should not admit code updates that can reprogram the WSN to snoop on targets without permission. For all of these cases, it is important that the sensor nodes be able to efficiently verify that code originates from a trusted source, namely the base station.

The goal and challenge for this paper is to build a secure mechanism for dynamic reprogramming of a WSN that is also robust and efficient, namely frugal in terms of memory footprint, energy consumption, overhead, and processor usage. Prior work on this important emerging problem is relatively scant. Existing code propagation protocols developed for WSNs, e.g. Deluge [12], MNP [22], MOAP [23], and Aqueduct [20], focus on realizing reprogramming features and optimizing performance and are not designed to be secure. The first work that we are aware of on secure code distribution is a two-page poster [14] that describes a scheme based on a public key signing of a chain of cryptographic hashes. As we will describe later, this chain-based scheme is highly brittle and susceptible to wireless packet losses.

A simple solution for verifying code images at each node is to employ a single global secret key shared by a base station and all sensor nodes to protect the integrity and authenticity of disseminated code. However, if an adversary can compromise a sensor node and capture the key, he can inject malicious code. Compromising a sensor node mote has been shown to be relatively quick and easy [5], allowing all internal information such as TinySec keys to be revealed. Sensor nodes are at high risk of compromise due to their *in situ* deployment, placing them within proximity of an adversary. In addition, cost constraints for resource-poor nodes limit the hardware security protections that can be integrated into a node.

Another symmetric key-based approach is for the base station to share a distinct pairwise key with each sensor node, and use this pairwise key to compute a keyed message authentication code based on a given code image. Each sensor node would then be able to verify its own keyed MAC. Compromise of a sensor node would reveal only one pairwise key, and would not result in a global compromise of the entire network. Unfortunately, this pairwise approach does not scale well and results in significant overhead in a large WSN, as each MAC needs to be sent at least to its destination node. Moreover, depending on the propagation mechanism, MAC information will be forwarded as overhead

by intermediate nodes that cannot benefit from this authentication information.

A third approach is to leverage only local random pairwise keys [7, 15, 6] used to establish secure connections between immediate neighbors. In this approach, the base station propagates code to its first-level neighbors. These neighbors hand off the code to their (second-level) neighbors that are two hops from the base station, and so on. Communication at each hop is secured by local pairwise keys. The problem with this approach is that compromise of just one intermediate node allows an adversary to inject into the WSN any arbitrary number of fake packets masquerading as code updates.

An attractive approach for authenticating code images is to apply a public key scheme. Suppose a base station has a private key, and each sensor node has the base station's corresponding public key. The base station signs every packet with the private key, so a sensor node can verify every packet with its public key. This simple per-packet scheme is avoided on the wired Internet because of its computationally intensive cost. Per-packet public key authentication would be far worse in a resource-constrained WSN that has at least two orders of magnitude less RAM, CPU, and bandwidth per node. While recent work has shown that elliptic curve cryptography (ECC) is feasible on MICA2-class sensor motes [17, 10], ECC-class public key authentication is still only practical if used sparingly.

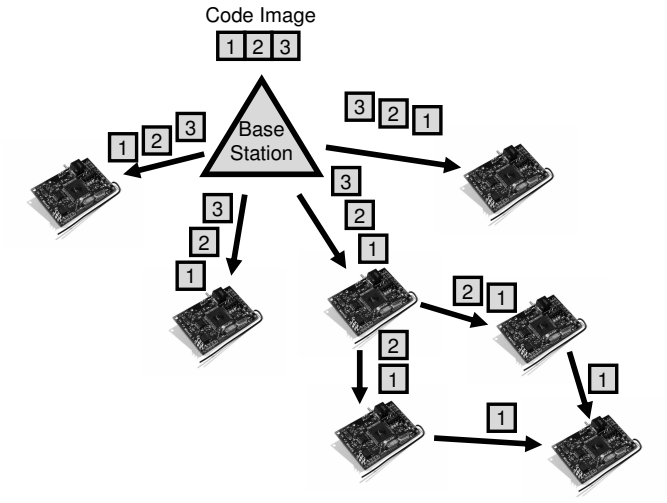

Figure 1: Code Propagation in WSNs. The code image is divided into multiple pages, which are reliably distributed hop-by-hop to nodes for reprogramming.

Our approach is to combine the best properties of both public key schemes and faster hash-based verification schemes to build secure and efficient dynamic programming of WSNs. Public key schemes have the advantage of simplifying key distribution while ensuring authentication even if a node is compromised, i.e. the public key does not allow a compromised node to spoof the base station. Hashed verification schemes have the advantage of fast execution time and small memory footprint. Our approach is akin in spirit to SSL on the Internet, which combines an initial public key scheme with a subsequent fast symmetric key approach.

We present two novel schemes for secure propagation of code in a dynamically programmable wireless sensor network, and compare them to an existing scheme. All these schemes are based on the idea of combining speedy hash-based verification of packetized code with the strong authenticity provided by a public key scheme. We first examine a version of the chain based scheme proposed earlier [14], which is a straightforward scheme that incurs very little overhead and requires nodes to perform only one public key operation. However, this approach results in significant overhead if packets arrive out of order. We propose a novel hash tree based scheme that allows nodes to authenticate packets out of order, thereby addressing the weakness introduced by chains. Hash trees also allow the integrity to be verified quickly. However, hash trees incur a relatively larger memory overhead and require nodes to perform more public key operations when the code image is large. Finally, we propose a novel hybrid scheme that judiciously combines the best aspects of hash trees and hash chains, overcoming the individual weaknesses of each approach to achieve robust, efficient, and secure code reprogramming in WSNs.

The paper is organized as follows. In section 2, we discuss the security requirements for dynamic reprogramming of WSNs. In section 3, we present the three schemes of secure dynamic programming of WSNs. Section 4 analyzes key properties of the proposed approaches, including overhead and security against DOS attacks. Section 5 presents the simulation results of our schemes, highlighting the modest overhead of our approach. Section 6 summarizes related work. Section 7 concludes the paper.

2 Security Requirements

2.1 Operating Assumptions

We assume our security scheme is targeted for today's standard sensor node platforms, such as the Berkeley mica2 node [1] or the Moteiv Telos Tmote [2]. A mica2 mote has 4K bytes of SRAM, 4KB internal EEPROM, and 128KB flash memory for program. The standard packet size provided by the TinyOS operating system is 29 bytes. The Tmote sky of moteiv company has an 8MHz CPU, which contains 10KB RAM and 48KB flash memory. It employs a 250 kbps Chipcon wireless transceiver which supports IEEE 802.15.4, and the maxmum packet size is 128 bytes.

Several groups of researchers have implemented RSA and Elliptic Curve (ECC) on mica2 motes [10, 17, 24, 9, 18, 8]. Up to now, the best result reported by N.Gura *et.al* shows that the public key encryption/decryption runs hundreds or thousands milliseconds, and consume hundreds of bytes of SRAM [10]. P. Ning *et.al* provide source code of ECC which runs 12 to 16 seconds to verify a signature on MICAz motes [18]. We assume that a sensor node can run public key cryptographic algorithms such as RSA and ECC. In addition, the base station has a private key K_s, and all sensor nodes are pre-configured with its corresponded public key K_p. For Elliptic curve with 168 bits of key, the size of a signature on a 4-byte hash is about 168*2 = 336 bits = 42 bytes, which can fit into a Tmote packet.

In this paper, we make the standard assumption of protocols like Deluge and MOAP that code images are propagated from a base station to every node in the network, as shown in Figure 1. The whole code image is segmented into a sequence of data packets from 1 to n. These packets

need to be reliably delivered to every node. Reliability is important because the code image is an executable binary that cannot contain bit errors or gaps due to lost packets. Because the data transmission in a wireless sensor network is unstable and the packet loss rate is high, reliable data transmission mechanisms (such as acknowledgement messages (ACK) or negative acknowledgement messages (NACK)) are only provided in one-hop communication.

We assume the base station is rich in computing resources and is securely protected. An adversary cannot compromise a base station. But the adversary can eavesdrop on any communication in the network, and is capable of compromising individual sensor nodes. An adversary can also inject any number of fake packets to sensor nodes nearby. If a sensor node cannot verify them, these fake packets will consume memory or computing time of the node and eventually exhaust its resources, e.g. battery life.

2.2 Security Goals

The goal of this paper is to efficiently protect the authenticity and integrity of propagated code images. Of course, a node cannot send or receive the whole code image at one time. In real implementations, the whole code image is divided into many data packets. These data packets should be reliably transmitted to each sensor node that is going to update its program image. So the goal of secure code propagation is to efficiently protect the authenticity and integrity of a large number of data packets during their reliable transmission process. In particular,

1. **Node-compromise resilience.** Every sensor node can authenticate and verify the integrity of the program code disseminated from a base station. An adversary cannot spoof the base station or change the contents of a code image without being detected by other nodes.
2. **DoS-attack resilience.** Every node can verify the code image as soon as it receives it. Otherwise an adversary can potentially launch denial of services attacks against the sensor network due to delayed authentication.
3. **Low cost.** The resource consumption of the proposed security mechanism must be light weight in terms of communication, computing and memory usage. To make sure our scheme is feasible and practical, it should consume low CPU workload, low memory consumption, and add as little excess communication overhead as possible.

3 Description of the Algorithms

3.1 Chain-based Scheme

Figure 2 illustrates the first method of secure code propagation proposed in [14], namely the chain based scheme. We examine this scheme more formally using the following notation. The base station divides the code image to be distributed into N fragments of data $data_i$, $i = 1..N$. A packet P_i is composed of both $data_i$ and a cryptographic hash H_{i+1} to verify the *next* packet. We use H_{i+1} to

denote the hash value of packet P_{i+1}. In general, $Hash(M)$ is the hash value of message M, and $E_k(M)$ is the encrypted value of message M with key k.

$$P_i = data_i||H_{i+1}, \qquad i = 1..N-1.$$
$$P_N = data_N.$$

Each cryptographic hash H_i is calculated over the full packet P_i, not just the data portion, thereby establishing a chain of hashes,

$$H_i = Hash(P_i) = Hash(data_i||H_{i+1}), \qquad i = 1..N-1.$$
$$H_N = Hash(P_N) = Hash(data_N).$$

Furthermore, the entire chain of hashes is verified by signing the first hash H_1 with the private key of the base station. That is, packet P_0 contains the signature S of H_1,

$$P_0 = signature(H_1)||H_1,$$

where $signature(H_1) = E_{K_s}(H_1)$, computed using the private key K_s of the base station.

The code image is transmitted as a sequence of packets (P_0 ... P_N). On receiving P_0, a node decrypts the signature S using the public key K_p of the base station and verifies H_1. On receiving P_1, this node verifies the authenticity and integrity of P_1 with the previously received and verified H_1. If verified, the node can safely extract both the code image fragment $data_1$ as well as the next hash H_2. On receiving P_2, this node verifies the integrity of P_2 with the previously received and verified H_2. This process continues until the final code fragment $data_N$ is received in packet P_N.

Because S is generated by encrypting H_1 using the private key of the base station, and only the base station knows this key, an adversary cannot generate a valid S containing H_1' (hash of a fake packet P_1'). As a result, a node can detect any tampering with packet P_1. In particular, if an adversary attempts to replace H_2 in P_1 with H_2' (hash of a fake packet P_2'), a node will detect this tampering. In turn, a node can detect any tampering with P_2, and so on.

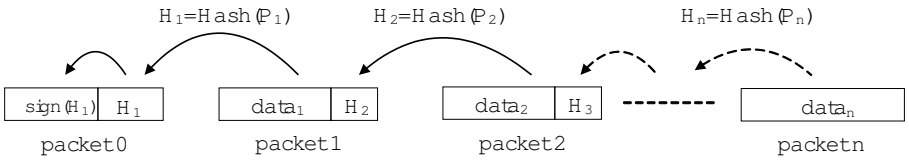

Figure 2: Signed Hash Chain Scheme.

In the absence of any packet losses and assuming that all packets arrive in the order they are sent, the chain based scheme performs very well. Public key encryption is used only once on a small-sized value (H_1) and a node needs to use public key decryption only once. Furthermore, a node only needs to save one hash value in SRAM and can verify the integrity of a packet as soon as it is received.

Referring back to Section 2.B, we see that this chain scheme satisfies security goals 1 and 3, but not adequately goal 2. In fact, the chain scheme meets the second goal only in a limited manner, as follows:

2a. Limited DoS-attack resilience. Suppose the data packets are disseminated as sequence from P_0 to P_n. Only after a node X has received all packets from P_0 to P_{k-1}, can it verify packet P_k.

In general, out-of-order packet arrivals are an important issue in WSNs, and arise from a variety of causes. Gaps in packet arrivals are commonly caused by packet losses due to wireless collisions at the medium access control layer, and/or by overflowed buffers at any intermediate nodes in the path. In addition, packets may arrive via multiple paths so that packets with higher sequence numbers arrive before those with lower sequence numbers. This disorder can occur because a protocol explicitly designs multi-path delivery into its operation, e.g. INSENS [4] and Deluge. For example, since Deluge employs reliable hop-by-hop flooding, code images can arrive by any path, so that a higher sequence number code update can arrive sooner via a shorter path than a lower sequence number code update taking a longer path. Disorder can also occur due to overhearing on the wireless broadcast medium, where newer packets destined for another node are overheard before older packets destined for the receiving node. This "feature" of wireless is actually exploited in the design of Deluge to lower the number of transmitted control messages (advertisements, requests, and NACKs) and data messages.

A more pertinent question is how well does a chain-based scheme map to the Deluge reliable code propagation mechanism.

At each hop, each Deluge receiver employs a negative acknowledgement-based (NACK) mechanism to reliably recover lost packets. When the sender is broadcasting packets, receivers delay transmitting ACK messages for a certain period of time. After the timeout, each receiver transmits a NACK message to inform the sender which packets it didn't receive. Then the sender re-broadcasts all missed packets. This process continues until every receiver obtains every packet. By allowing gaps in the packet sequence, the NACK approach innately tolerates disorder at a packet granularity, which is at odds with the requirement of chain schemes. This makes the chain scheme ill-suited for direct application to Deluge on a per-packet basis.

3.2 Hash Tree based Scheme

To achieve DoS-attack resilience and allow immediate verification of out-of-order packets, we propose a novel method for secure code propagation based on a signed hash tree. We assume an underlying code distribution mechanism like Deluge: a node sends a group of packets to its neighbor nodes, and after a certain time, each neighbor node sends back a NACK message to tell the sender which packets it missed. Then the sender retransmits missed packets. For example, if the sender transmits packet P_1, P_2, P_3, and P_4, and the receiver gets P_1 and P_4, but missed P_2 and P_3, then the receiver sends a NACK message to inform the sender. The sender then retransmits P_2 and P_3. This process saves traffic since the receiver doesn't have to acknowledge every packet. To simplify the algorithm description, we just assume that the sender transmits all packets from P_1 to P_n to its neighbor nodes.

To enable nodes to authenticate and verify the integrity of code image packets quickly, even when the packets arrive out of order, it is important to propagate the hash values of those packets *a priori*. This can of course be done by sending packets containing only the hash values of code image packets in advance. The key challenge here is to make sure that only a small number (preferably one) of

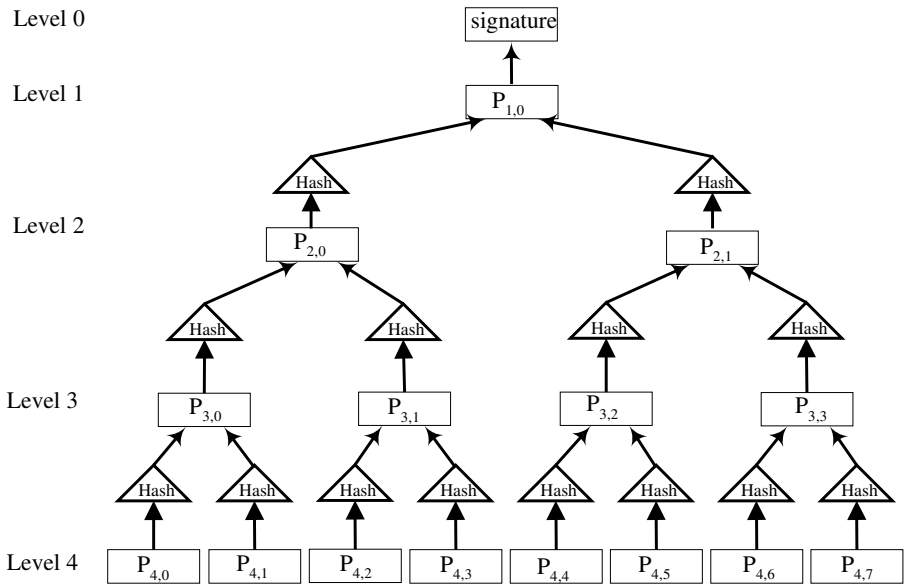

Figure 3: Signed Hash Tree Scheme. This hash tree structure has only 5 levels (m = 4), and every index packet contains 2 hash values (w = 2).

public key operations have to be performed by nodes.

Figure 3 illustrates our basic hash tree scheme. The code image is divided into packets at the base station, and a secure hash is computed on each packet. These hash values are themselves input to create a new level of hashes, and so on up the tree. A packet at level i contains hash values of w packets in level $i+1$. For example, a packet $P_{i,j}$ contains hash value $Hash(P_{i+1,j*w})$, $Hash(P_{i+1,j*w+1})$, ..., $Hash(P_{i+1,j*w+w-1})$. If the hash tree has $m+1$ levels (from 0 to m), the packets in level m are data packets that contain the code image. More precisely, for a packet in level i where $1 < i < m$,

$$P_{i,j} = Hash(P_{i+1,j*w})||...||Hash(P_{i+1,j*w+w-1})||other\_info$$

All packets at level m contain their respective data fragments of the code image. Thus the hash value of every data packet is included in one of the packets in level $m-1$, and the hash value of every packet in level $m-1$ is included in one of the packets in level $m-2$, and so on. This tree is built in such a way that there is exactly one packet at level 1. Note that the hash tree proposed here is different from a Merkle Hash tree.

The root value at the top of the tree, level 0, is a signature that is obtained by encrypting the hash value of the packet at level 1 using the private key (K_s) of the base station, i.e

$$P_{0,0} = E_{K_s}(Hash(P_{1,0}))||Hash(P_{1,0})||other\_info$$

This signature can be used to authenticate the source of P_1 as well as verify the integrity of P_1. Furthermore, since our hash tree links all packets at one level with the packets at the next level, this single signature in turn provides support for authentication and integrity verification of all packets in the hash tree. This is because if an adversary tampers with the contents of a packet, the hash

of the original (untampered) packet received in a packet in the previous level can be used to detect this tampering.

The signed root value is released *before* the data packets are released. More precisely, we define meta packets, called *index packets*, to contain the signature and hash tree values computed from the data packets. In the figure, all internal nodes are index packets, and all data packets are the leaf nodes of the hash tree. A sender transmits all index packets before sending any data packets. Because the total size of hash values is smaller than the size of the packetized code image, this pre-release of verification information creates little traffic. When a node sends a code image to its neighbor nodes, it sends packets from the low levels to the high levels. First, the root packet that contains the signature is sent, then the packet in level 1 is sent, then packets at level 2 are sent, and so on. For each level, the receiver sends back an acknowledgement message to inform the local sender whether it received all packets in this level, or which packets were missed. When a node receives the root packet, it saves the signature. It uses this signature to authenticate/verify the data in the level 1 packet, it uses the hash values in that level 1 packet to verify the integrity of packets in level 2, and so on. Eventually, this node can authenticate/verify every packet in level m (code image) using the hash values received in packets at level $m-1$.

Referring back to **2a** (limited DoS-attack resilience) of the chain-based scheme, we see that the hash tree based scheme satisfies a much stronger property:

2b. DoS-attack resilience. After a node has received all packets from levels 0 through $k-1$, it can verify any packet in level k, i.e. it can verify packets of level k as soon as they are received, without being constrained by the order in which they are received.

To explain this more precisely, assmue that there are w hash values in an index packet. This means there are w^{l-1} packets in level l ($0 < l \leq m$). If a node has received all packets in levels 0 through $l-1$, it can receive and verify the next w^{l-1} packets (i.e. the packets that belong to level l) in any order. Thus, the hash tree based scheme can be applied to current reliable data delivery schemes of sensor networks with little extra cost.

The hash tree scheme exhibits several advantages that address the security goals outlined in Section 2. First, signing the root of the hash tree with the private key prevents an adversary from spoofing the base station or tampering with code images without detection. This addresses the security goal of node-compromise resilience. Second, the receiver only needs to execute the public key verification operation once, upon receipt of the initial signature packet. All subsequent verification operations are performed quickly using hashes in the tree. Third, the hash tree enables every node to verify each packet immediately. When a data packet arrives, a quick hash of its contents can be compared to the previously saved hash to verify authenticity and integrity. These two points address the DoS-resilience goal. Fourth, even if a node receives data packets out of order, these packets can still be authenticated quickly with in some limited restrictions. This addresses the goal of low cost implementation by taking advantage of out-of-order arrival of packets, which cuts down on unnecessary retransmissions.

One cost of the hash tree scheme is the extra index packets that need to be transmitted. This overhead is relatively small compared to the total number of data packets. For example, suppose a code image to be downloaded is 32 KB in size. Assuming that each packet can hold 29 bytes of data, the code image amounts to about 1,070 data packets. Assuming a 4-byte hash, we can pack in

7 hashes in a packet. This implies that there will be 181 index packets (153 in level 4, 22 in level 3, 4 in level 2, 1 in level 1, and 1 in level 0). Furthermore, these index packets allow a node to receive and verify packets in any order, and in turn prevent transmission of many acknowledgment message in the underlying reliabilty mechanism.

Another cost of the tree-based scheme is its memory consumption. Roughly, if a node wants to verify each packet in level i, it should save the hash values of level $i-1$ packets in its SRAM. This means it may have to save all packets in level $m-1$ in its SRAM to verify all data packets. For a large code image, this could become a concern. Fortunately, sensor nodes have considerable storage in Flash or EEPROM to save the internal hash values. In addition, as tested in [5], reading data from EEPROM is very fast. It takes less than 1 millisecond to read 8 bytes of data from EEPROM, which introduces minimal delay. Finally, an RC5-based hash is light weight both in terms of memory and computational requirements for MICA2 motes [19].

The SRAM consumption problem exists in Deluge as well, since all packets received out of order need to be stored in SRAM. To address this problem and make the state management easier, Deluge segments the whole program code into pages, where each page contains a fixed number of packets. To receive packets in page i, a node has to reliably receive all packets in pages 0 through $i-1$. As described earlier, Deluge uses NACK messages to reliable receive all packets with in a page. This requires a node to temporally save packets only within a page into SRAM. So, for a 1 KB SRAM memory, the size of a page is also set to 1 KB. Given a packet size of 29 bytes, this implies that a page contains about 36 packets.

To adapt our hash tree scheme to Deluge, we can build a hash tree structure for each page, and the root node of each hash tree is a signature signed by the base station with its private key. We term this variation the *multiple public key based hash tree scheme*. The key drawback of this approach is that it introduces more public key operations (one per page of code image). To address this, we propose our hybrid scheme.

3.3 Hybrid Scheme

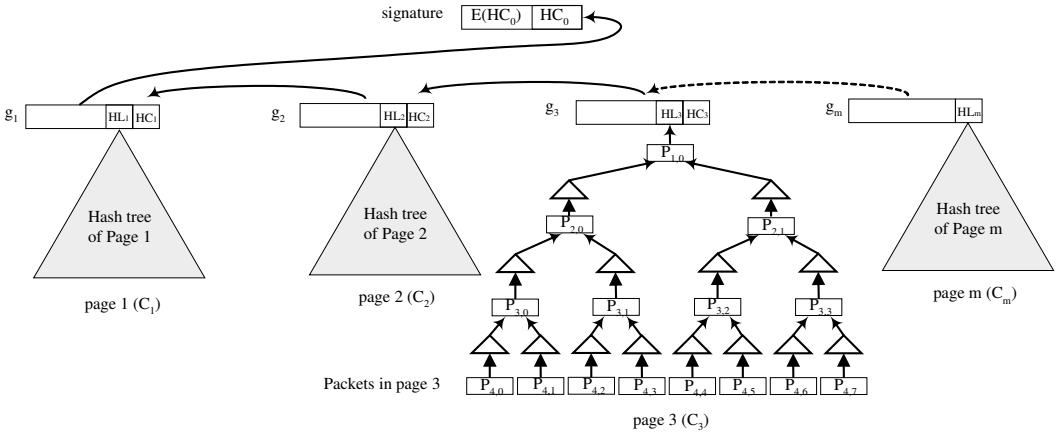

Figure 4: Hybrid Signed Scheme, combines hash chain structure for inter-page authentication and hash tree structure for internal page authentication.

To reduce memory consumption and the number of public key opertions that a node has to perform, we propose our third method, namely the hybrid scheme. As the name suggests, this scheme is a combination of the chain based and tree based scheme. This hybrid scheme has been designed to provide authentication and integrity support in code propagation, and also adapt to the Deluge protocol. In Deluge, a code image is first divided in a sequence of m pages, $C_1, C_2, ..., C_m$, and each page is further divided into a sequence of a fixed number of packet. A node must have received all packets in pages $C_1, ..., C_{k-1}$ before it can accept packets from page C_k. However, within page C_k, the NACK-based design allows out-of-order arrival. Thus, all pages of the code image must be received in a sequential order, while packets within a page may be received in any order. We preserve this requirement and provide authentication and integrity support.

Figure 4 illustrates our hybrid scheme. At the granularity of pages, our approach is to employ a chain-based scheme for authentication and integrity. At the granularity of packets, our hybrid approach employs our hash tree based design to allow rapid and out-of-order authentication. The key observation is that chain-based authentication of pages reduces the number of public key operations that a node has to perform to only one, in comparison with the *multiple public key based hash tree scheme* of the previous subsection. At the same time, the chain-based page authentication imposes no additional cost in terms of susceptibility to disorder. Deluge already requires order at the page granularity, so the chain-based approach of requiring order does not impose any additional penalty.

A hash tree is constructed for each page. This hash tree is similar to the one described in Section 3.2, except for the packet at level 0. For each page C_i, the packet g_i at level 0 is called the *page index* packet of C_i. This packet contains information about this page and the next page in the sequence:

$$g_i = HL_i||HC_i||other\_page\_info$$

Where $1 < i < m$. A page index packet g_i contains two hash values, HL_i, and HC_i $(1 < i \leq m)$. HL_i is the hash value of the single packet at level 1 in the hash tree of page C_i. HC_i is the hash value of g_{i+1}. For page C_0, the page index packet is a signature created by encrypting $(HL_1||HC_1)$ using the private key K_s of the base station, i.e.

$$g_0 = E_{K_s}(HC_0||other\_info)||HC_0||other\_info$$

Notice that the public key operation has been used in only one packet (g_0) in our hybrid scheme.

The hybrid scheme retains the principal advantage of the hash tree based scheme. It allows nodes to authenticate packets and verify their integrity quickly, even when the packets arrive out of order. Theoretically, the hybrid scheme satisfies a weaker DoS-attack resilience requirement than **2b**. It satisfies that:

2c. DoS-attack resilience. For all n packets in a page, when a node received all packets from 1 to $k-1$, it can verify any packets from k to $MIN(m, n-(k-1))$ with any order, where $m - k > k$.

Overall, the security goals of Section 2 are satisfied, and in a manner that is more efficient than the pure hash tree. Signing the first value of the chain satisfies the goal of node-compromise resilience, because an adversary cannot modify any index packets or data packets nor spoof the base station without being detected. DoS-attack resilience is preserved since packets can be verified immediately.

This is achieved at a low cost since the public key operation is performed but once, and disorder is tolerated. The hybrid solution scales well, owing to the tree structure, yet is more efficient than the pure hash tree thanks to the chain because public key operations don't have to be performed on each page.

4 Analysis

4.1 Number of Index packets

We would like to estimate the number of index packets needed for a hash tree. Let $I(n)$ denote the number of index packets (excluding packet $P_{0,0}$) in a hash tree with n data packets. The height of the hash tree is $\lceil \log_w n \rceil$ (excluding the root node). If $\log_w n$ is an integer k, then $I(n) = (w^k - 1)/(w-1) \approx n/(w-1)$. Otherwise, $(w^{\lfloor \log_w n \rfloor} - 1)/(w-1) \leq I(n) \leq (w^{\lceil \log_w n \rceil} - 1)/(w-1)$. So, we have

$$I(n) \approx n/(w-1)$$

We see that the number of index packets is within a factor of $\frac{1}{w-1}$ of the number of data packets.

4.2 Packet authentication in hash tree scheme

After a node has received k packets in the hash tree based scheme, it can authenticate and verify integrity of several new packets it receives next. We use function $f(x)$ to denote the number of new packets a node can verify immediately on receiving after it has already received the first x packets. Our goal is to estimate $f(x)$. We have $f(1) = 1$, since the first packet $P_{0,0}$ contains the signature of the hash value of next packet $P_{1,0}$. Suppose each index packet contains w hash values and a node has already receives k packets. To simplify the problem, let's assume that all k packets and next $f(k)$ packets are index packets. Since each packet contains w hash values, when this node receives $k+1^{th}$, it can verify an additional w upcoming packets. So we have $f(k+1) = f(k) - 1 + w$. Based on this, we get the following recursive equation

$$f(x) = \begin{cases} 1 & \text{if } x = 1 \\ f(x-1) - 1 + w & \text{if } x > 1 \end{cases}$$

Solve this equation, we get that

$$f(x) = (x-1)(w-1) + 1$$

Now, lets consider data packets. Suppose there are N packets in total. We have

$$f(x) = min((x-1)(w-1) + 1), N - x)$$

where $N \approx n \times \frac{w}{w-1}$.

However, sometimes an index packet may contain less than w hash values, so the actual value of $f(x)$ may be slightly less.

4.3 DoS attacks and countermeasures

An adversary can launch a DoS attack on a node by continually sending a fake signature. Although the scope of damage due to this attack is limited, we can defend against it if the size of the root packet is large enough. When the base station disseminates a root packet, it attaches a one-way hash chain (OHC) number H_i in the packet. Suppose every node is pre-configured with the inital number H_0 of the OHC. When a node receives a root packet, it verifies the OHC number by applying a one-way function with its saved OHC number. Only after this verification does the node verify the signature by running the public key algorithm. Because an attacker doesn't have the OHC, (s)he cannot make a node run public key algorithm by sending fake signatures to that node. In this way, the heavyweight public key verification is protected by a lightweight preceding step of OHC verification.

Another kind of DoS attack that is possible in our scheme is that an adversary continues to transmit ACK or NACK messages to the sender so that the sender will continue to retransmit packets and exhaust its battery and/or bandwidth. One countermeasure against such an exhaustion attack is to establish a shared secret key between a sender and all its receivers, called a cluster key. This cluster key is then used to generate a MAC for each acknowledgment packets. The sender can verify the authenticity of each packet. If an adversary can compromise a node, it can discover that node's cluster key. The adversary can then launch this DoS attack. However, notice that such an attack still has limited impact. The attacker cannot flood the entire network or program malicious code into other sensor nodes.

5 Experiments

The goal of our experiments is to understand the cost of our security mechanisms in code propagation, and see if these mechanisms are feasible in current sensor network applications. Since prior work has already tested memory and computational costs of public key algorithms and hash-based algorithms on sensor motes, our focus is on comparing the message overhead cost of our security schemes with the Deluge non-secure code propagation scheme. The intent is to discover if the hybrid, tree-structured, and chain-based approaches can achieve their security goals while introducing only modest overhead.

We simulated a network in which a base station can communicate directly with 20 sensor nodes. In our experiments, the base station disseminates a program code to its direct neighbor nodes, and we measured the number of packets sent during this process under different packet loss rates and code size. We simulated Deluge as well as our three secure code propagation schemes.

5.1 Message Overhead vs. Packet Loss Rate

Our first experiment was to measure the message overhead of our secure reprogramming schemes compared with Deluge. In this experiment, we varied packet loss rate from 0 to 15%. The size of program code is 32 KB, which is usual for real applications. Each hash value is 4 bytes, and a 29 bytes index packet contains 6 hash values (remaining space is used for some other information). We measured the total number of packets sent, and the total time to distribute code to all sensor nodes.

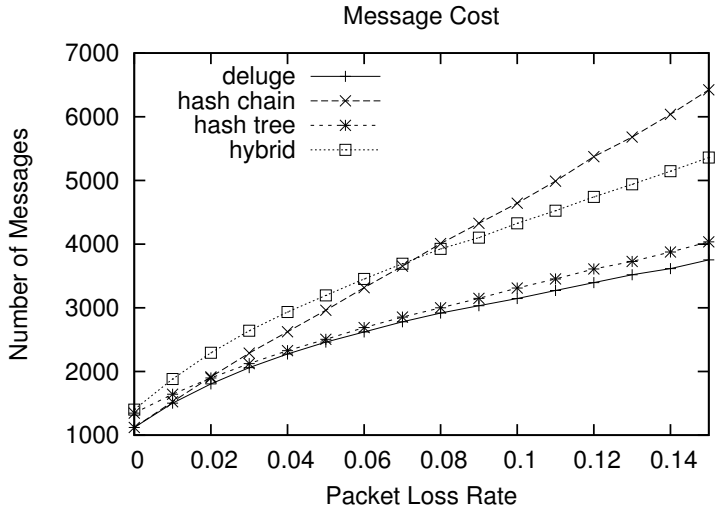

Figure 5: Message cost of original deluge scheme and secure schemes under different packet loss rate.

Figure 5 shows the message cost, measured by the number of packets sent in the network. The number of packets exchanged corresponds to the energy cost, since the power used in wireless transmission dominates energy consumption of a sensor node. From these experiments, we see that the extra cost of our security scheme is relatively small and is feasible in current sensor network platforms. For example, when packet loss rate is 2%, the number of packets sent in Deluge is about 1800, and the number of packets sent in hybrid scheme is about 2300. The extra cost is about 28%. Because reprogramming doesn't happen very often, we consider the 28% extra cost during the reprogramming phase to be relatively small for the entire life of a sensor node. Figure 6 shows the message cost under Tmote configuration. On Tmote, the packet size is 128 bytes, and a node use 4 KByes to save code image in RAM. We see that the hybrid scheme gets better performance than chain scheme, because hybrid scheme uses less amount of index packets when the length of packet increases, and because the number of packets in each page increases.

5.2 Delay vs. Packet Loss Rate

Figure 7 shows the time delay for reprogramming. The hash chain takes the most time to reprogram the network. This is as expected because the chain based scheme does not tolerate packet disorder. The hybrid scheme reprograms more quickly because it can take advantage of out-of-order packet arrival. Also, we see from figure 8 that hybrid scheme has better performance on Tmote. Deluge takes the least time to reprogram because it does not need to send out index packets a priori.

5.3 Message Overhead vs. Size of Code Image

Figure 9 displays the message cost when the size of the program code changes. In this experiment, we simulated propagation of program code with different sizes ranging from 1 KB to 128 KB. We fixed the packet loss rate as 10%. For all schemes, we see that the number of messages linearly

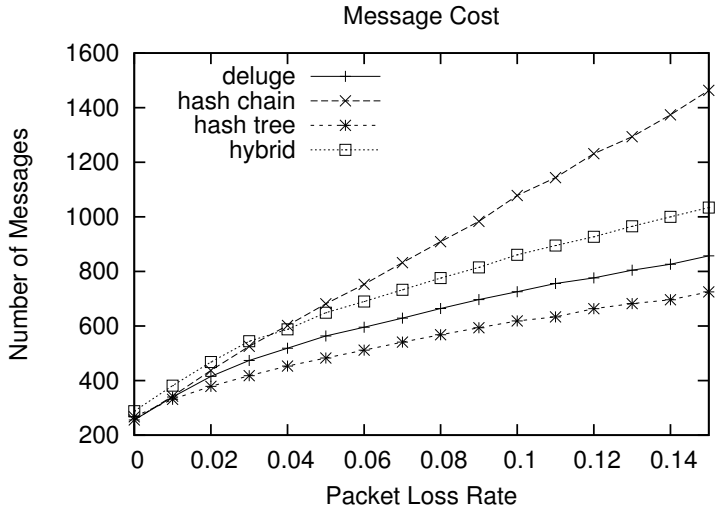

Figure 6: Message cost of original deluge scheme and secure schemes with Tmote configuration.

increases as the size of code increases, and the ratio of extra cost is about 39%. This linear increase as a function of code size conforms with the experimental results obtained in the Deluge paper.

5.4 Message Overhead vs. Network Density

In this experiment, we measured the number of exchanged messages when the density of network changes. For instance, we changed the number of neighbor nodes from 5 to 100. Notice that the network will become extremely dense when the number of neighbor nodes reaches to around 60.

Figure 10 depicts the message cost as a function of network density, e.g. number of neighbor nodes. Notice that the tree structure scheme looks better than Deluge when network becomes more dense. That is because in the "pure" tree scheme, the whole packets are not divided into pages. The hash tree assumes a node can receive all packets in its memory, whereas in Deluge and hybrid scheme a node can only receive all packets within a page. In addition, for the tree scheme, if we want to send 32KB code, as calculated, we have to send 32000/5=6400 bytes index data, and all of them need to saved in RAM. The tree scheme appears to be feasible when a node has more RAM and when the size of the code image is small.

6 Related Work

Hash functions have been widely used to authenticate transmitted data, mainly because of being significantly more efficient than public key algorithms. For example, A. Perrig *et. al* proposed μTESLA [19] protocol for source authentication in data dissemination through lossy channel. However, μTESLA requires global loose time synchronization which is difficult to apply to reprogramming wireless sensor networks. Also it is vulnerable to denial of services attacks due to delayed authentication. C. Karlof *et. al* proposed a mechanism for secure multicast in lossy data transmission

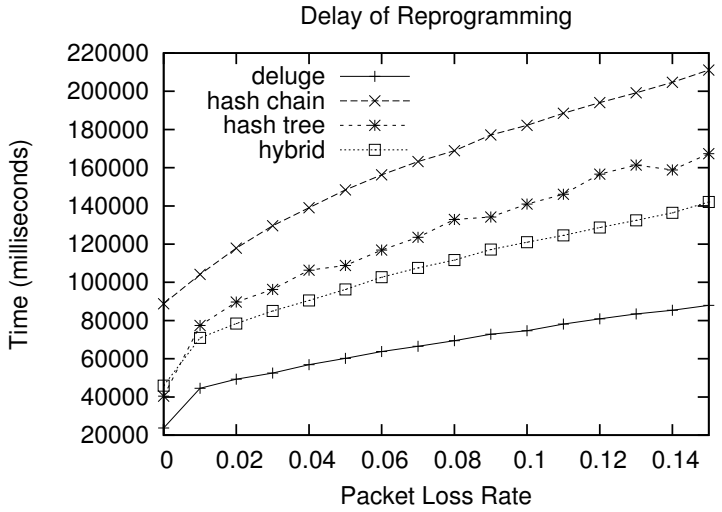

Figure 7: Time Delay of original deluge scheme and hash tree scheme under different packet loss rate.

environment [13]. A receiver can verify each packet it received and can recover original data even though some packets are lost. But this mechanism is too expensive for resource-poor sensor nodes.

V. Gupta *et. al* implemented the SSL protocol on sensor node with RSA and elliptic curve algorithms, called Sizzle [9]. In contrast with Sizzle, our scheme works for one-to-many data dissemination paradigms, in which a base station broadcasts a program code image to multiple sensor nodes one or more hops away. The base station doesn't use end-to-end data transmission scheme since that is too expensive.

Beyond the Deluge data dissemination protocol, MOAP (Multi-hop Over-the-Air Programming) [23] employs a ripplelike mechanism to propagate the new image to the entire network. At each ripple, a subset of nodes are the source nodes and all the other neighbor nodes are receivers. A publish-subscribe mechanism is used to prevent multiple nodes becoming the source nodes in the same neighborhood. Source nodes publish their newer version and all interested nodes subscribe. However, MOAP requires the entire image to be reliably received before the next ripple can begin and hence trades off latency. MOAP's code propagation approach of reliably flooding the same code image hop by hop to all nodes is similar to Deluge's approach.

7 Conclusions and Future Work

Authentication of code images received over a wireless channel is an important capability for many in situ WSN applications. This paper develops a novel *secure* code propagation protocol for wireless sensor networks. Our solution is to employ public key signing of a joint structure combining hash chains and hash trees. This solution has the following advantages: node-compromise resilience, since the public key and linked chain-tree hash structures protect all packets; DoS-attack resilience since the tree structure allows immediate and rapid verification of packets; and low cost realization in terms of 1) low delay due to public key authentication being performed only once initially, 2) low

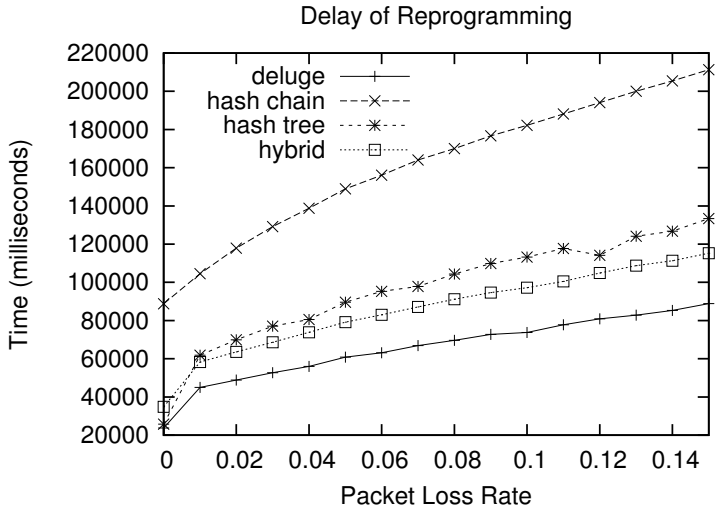

Figure 8: Time Delay of original deluge scheme and hash tree scheme with Tmote configuration.

delay due to fast hash-based verifications for all subsequent packets, and 3) low overhead since the hash tree tolerates disorder and reduces unnecessary retransmissions. Our experiments confirm many of these observations. Our schemes are light weight enough to be feasible for current sensor network platforms. This solution is practical in that is well-adapted to prevailing code propagation protocols such as Deluge. In the future, we plan to implement the hybrid scheme on a multi-node testbed comprised of the Tmote TELOS motes. This will enable us to test our realization in a large scale multi-hop configuration.

References

[1] Crossbow website. http://www.xbow.com.

[2] Tmote. http://www.moteiv.com.

[3] R. Adler, P. Buonadonna, J. Chabra, M. Flanigan, L. Krisnhamurthy, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and deployment of industrial sensor networks: Experiences from the north sea and a semiconductor plant. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, San Diego, California, USA, November 2005.

[4] J. Deng, R. Han, and S. Mishra. Insens: Intrusion-tolerant routing for wireless sensor networks. *Elsevier Journal on Computer Communications, Special Issue on Dependable Wireless Sensor Networks*, 2005. To appear.

[5] J. Deng, R. Han, and S. Mishra. Practical study of transitory master key establishment for wireless sensor networks. In *1st IEEE/CreateNet Conference on Security and Privacy in Communication Networks (SecureComm 2005)*, pages 289–299, Athens, Greece, September 2005.

[6] W. Du, J. Deng, Y. Han, and P. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *10th ACM Conference on Computer and Communications Security (CCS'03)*, Washington D.C, USA, October 2003.

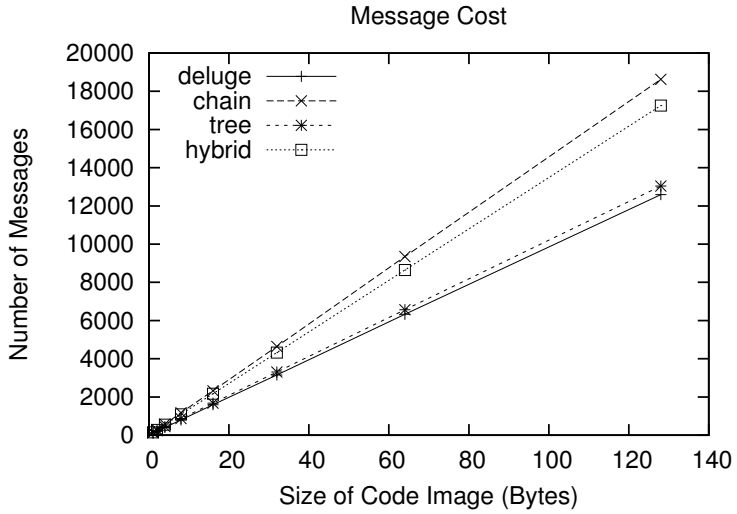

Figure 9: Message Cost of Deluge and hash tree scheme under different program cod e size.

[7] L. Eschenauer and V. Gligor. A key-management scheme for distributed sensor networks. In *Conference on Computer and Communications Security, (CCS'02)*, Washington DC, USA, November 2002.

[8] G. Gaubatz, J.-P. Kaps, E. Ozturk, and B. Sunar. State of the art in ultra-low power public key cryptography for wireless sensor networks. In *2nd IEEE International Workshop on Pervasive Computing and Communication Security*, Kauai Island, Hawaii, USA, March 2005.

[9] V. Gupta, M. Wurm, Y. Zhu, M. Millard, S. Fung, N. Gura, H. Eberle, and S. C. Shantz. Sizzle: A standards-based end-to-end security architecture for the embedded internet. In *3rd Annual IEEE International Conference on Pervarsive Computing and Communications*, Kauai Island, Hawaii, USA, March 2005.

[10] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In *6th International Workshop on Cryptographic Hardware and Embedded Systems(CHES'04)*, Cambridge, Boston, USA, August 2004.

[11] J.-H. Huang, S. Amjad, and S. Mishra. Cenwits: A sensor-based loosely-coupled search and rescue system using witnesses. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2005.

[12] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *2nd International Conference on Embedded Networked Sensor Systems(SenSys'04)*, Baltimore, Maryland, USA, November 2004.

[13] C. Karlof, N. Sastry, Y. Li, A. Perrig, , and J. Tygar. Distillation codes and applications to dos resistant multicast authentication. To appear in the 11th Annual Network and Distributed Systems Security Symposium (NDSS 2004), February 2004.

[14] P. E. Lanigan, R. Gandhi, and P. Narasimhan. Secure dissemination of code updates in sensor networks. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2005.

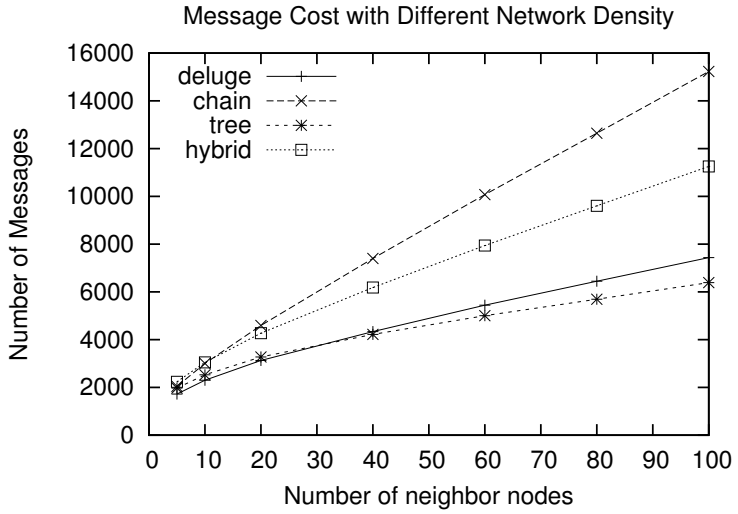

Figure 10: Message overhead as a function of network density.

[15] D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. In *CCS'03*, Washingon D.C, USA, October 2003.

[16] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *1st ACM Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, pages 88–97, 2002.

[17] D. Malan, M. Welsh, and M. Smith. A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography. In *1st IEEE International Conference on Sensor and Ad Hoc Communications and Networks*, 2004.

[18] P. Ning and A. Liu. Tinyecc: Elliptic curve cryptography for sensor networks. http://discovery.csc.ncsu.edu/ pning/software/TinyECC/index.html.

[19] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar. Spins: Security protocols for sensor networks. *Wireless Networks Journal(WINET)*, 8(5):521–534, September 2002.

[20] L. A. Phillips. Aqueduct: Robust and efficient code propagation in heterogeneous wireless sensor networks. Master's thesis, University of Colorado at Boulder, 2005.

[21] G. Simon, M. Maróti, Ákos Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton. Sensor network-based countersniper system. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 1–12, New York, NY, USA, 2004. ACM Press.

[22] S.S.Kulkarni and L. Wang. Mnp: Multihop network reprogramming service for sensor networks. In *25th International Conference on Distributed Computing Systems (ICDCS)*, Columbus, OH, June 2005.

[23] T. Stathopoulos, J. Heidemann, and D. Estrin. A remote code update mechanism for wireless sensor networks. Technical Report CENS-TR-30, University of California, Los Angeles, Center for Embedded Networked Computing, November 2003.

[24] R. Watro, D. Kong, S. fen Cuti, C. Gardiner, C. Lynn, and P. Kruus. Tinypk: Securing sensor networks with public key technology. In *2004 ACM Workshop on Security of Ad Hoc and Sensor Networks*, Washington, DC, USA, October 2004.