Secure Code Distribution in Dynamically Programmable Wireless Sensor Networks

Jing Deng Richard Han Shivakant Mishra
Department of Computer Science
University of Colorado
Boulder, Colorado, USA
Jing.Deng@Colorado.edu, Richard.Han@colorado.edu, mishras@cs.colorado.edu

ABSTRACT

Remote reprogramming of in situ wireless sensor networks (WSNs) via the wireless link is an important capability. Securing the process of reprogramming allows each sensor node to authenticate each received code image. Due to the resource constraints of WSNs, public key schemes must be used sparingly. This paper introduces a mechanism for secure and efficient code distribution that employs public key cryptography only to sign the root of a combined structure consisting of both hash chains and hash trees. The chain based scheme works best when packets are received in the order they are sent with very few losses. Our hash tree based scheme allows nodes to authenticate packets and verify their integrity quickly, even when the packets may arrive out of order, but can result in too many public key operations. Integrating hash chains and hash trees produces a mechanism that is both resilient to losses and lightweight in terms of reducing memory consumption and the 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.

Categories and Subject Descriptors

K.6.5 [**Computing Milieux**]: Management of Computing and Information Systems—*Security and Protection*; I.2.9 [**Sensors**]

General Terms

Security

Keywords

Security, Sensor Networks, Secure Reprogramming

IPSN'06, April 19–21, 2006, Nashville, Tennessee, USA.

1. INTRODUCTION

Many applications of wireless sensor networks (WSNs) can benefit from remote reprogramming of sensor nodes through the wireless channel. For example, deployed sensor nodes with buggy code can be patched with new code images or updated with new applications and/or unanticipated features. In these cases, remote wireless reprogramming of deployed sensor nodes that may be spread out over rugged terrain is far more practical than manual intervention.

In certain cases, securing the process of dynamic reprogramming is critical. In military deployments, distribution of false or viral code images by an adversary can cripple the WSN and/or deceive the deployer. In commercial applications such as monitoring of semiconductor fabrication labs or oil tankers [3], code updates must be verified to ensure that catastrophic damage is not caused to industrial processes. In certain applications, privacy and anonymity of communicating parties can be compromised by installing code updates that 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.

The goal of this paper is to build a secure and efficient mechanism for distributing code images in dynamically reprogrammable WSNs. This mechanism helps sensor nodes to securely verify the authenticity and integrity of code image. The challenge is to achieve this goal given the severe resource constraints of WSNs, namely the limited memory, energy, bandwidth, and processing. Prior work on this important emerging problem is relatively scant. Existing code propagation protocols developed for WSNs, e.g. Deluge [12], MNP [21], MOAP [22], 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 using public key cryptography for signing of a chain of cryptographic hashes. A similar scheme is proposed and implemented in Deluge by P. Dutta *et al.* [6]. As we will describe later, this chain-based scheme brings heavy traffic in lossy wireless communications.

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 still inject malicious code. Compromising a sensor node mote has been shown to be relatively quick and easy [4], allowing all internal information such as TinySec keys to be revealed. Sensor

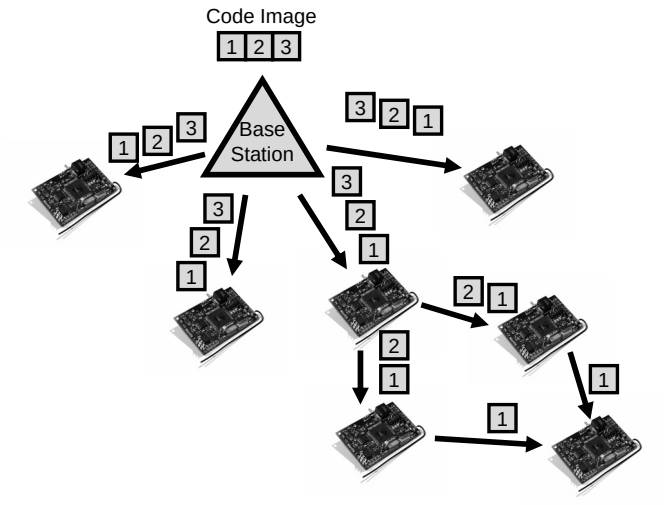


Figure 1: Code Propagation in WSNs. The code image is divided into multiple pages, which are reliably distributed hop-by-hop to nodes for reprogramming.

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. 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. Using randomly distributed pairwise keys [7, 15, 5] may improve the efficiency of the MAC based approach, but is still susceptible to code injection attacks from a compromised node.

Public key-based authentication of sensor code images represents another option. 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. However, this simple per-packet scheme is computationally expensive, and is avoided on the wired Internet. 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 [16, 11], ECC-class public key authentication is still only practical if used sparingly.

We present a novel scheme for secure and efficient propagation of sensor code images that combines the best features of public key authentication, hash-based verification, hash chains, and hash trees. 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. We first describe the signed hash chain scheme, which incurs very little overhead of computing and requires nodes to perform only one public key operation. However, this approach does not tolerate packets arriving out of order, as would be the case due to collisions in typical wireless networks. We further investigate a hash tree based scheme that allows nodes to authenticate packets out of order, thereby addressing the weakness introduced by chains. Hash trees allow the integrity to be verified quickly. However, hash trees incur a relatively larger memory overhead or require nodes to perform more public key operations when a large code image has to be fragmented into many pages. 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 three schemes for secure dynamic reprogramming 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. SYSTEM ASSUMPTIONS AND SECURITY REQUIREMENTS

2.1 System Model

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.

In this paper we make the standard assumption of protocols like Deluge protocol developed by J. Hui *et al.* [12], in which the code images are propagated from a base station to every node in the network, as shown in Figure 1. The whole code image is split into a sequence of pages, and each page contains a number of data packets. Each page is disseminated sequentially: a node must get all pages from 1 to $k-1$ before it begins to receive packets in page k. A sending node broadcasts all packets of a page to its neighboring nodes which haven't received that page yet. The sender sends packets round-robin; it broadcasts all packets one by one, and then waits for ACK or NACK messages back from receivers. Each feedback message tells the sender which packets are missing at a receiver. The sender resends all missed packets one by one, until every packet is received by every receiver. The goal of the above transmission scheme is to provide efficient and reliable program code image dissemination, since the data transmission in a wireless sensor network is unstable and the packet loss rate is high.

Several groups of researchers have implemented RSA and Elliptic Curve (ECC) on mica2 motes [11, 16, 23, 10, 17,

9]. 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 [11]. P. Ning *et.al* provide source code of ECC which runs 12 to 16 seconds to verify a signature on MICAz motes [17]. 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 ECC with 168 bits of key, the size of a signature on a 4-byte hash is about $168 \times 2 = 336$ bits $= 42$ bytes, which can be fit into a Tmote packet.

2.2 Threat Model and Security Goals

The goal of an adversary is to reprogram its own code into sensor nodes or launch denial of service attacks against the large number of sensor nodes in the network. He is able to eavesdrop on any communication in the network, to compromise individual sensor nodes and capture all information inside it, and to inject fake packets to sensor nodes nearby. However, we assume the base station is rich in computing resources and is securely projected. An adversary cannot compromise the base station. Although an adversary can attack the privacy of a code image by eavesdropping on or compromising a node, we don't protect the confidentiality of program code in this paper. If the adversary cannot compromise a node, confidentiality can easily be protected by a global key. However, if the adversary is capable of compromising a node, protecting confidentiality will be very difficult.

The goal of this paper is to efficiently protect the authenticity and integrity of propagated code images. Ideally, we hope that

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, For example, if an adversary injects fake packets to a node and that node cannot verify those packets immediately, then either those packets will consume memory or the computing time of this node and eventually exhaust its resources, or this node has to drop packets without verification, potentially dropping valid data packets.

3. **Low cost.** The resource consumption of the proposed security mechanism must be light weight in terms of communication, computing and memory usage.

3. DESCRIPTION OF THE ALGORITHMS

3.1 Chain-based Scheme

Figure 2 illustrates the first method of secure code propagation, 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

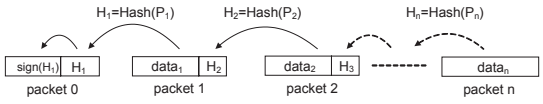


Figure 2: Signed Hash Chain Scheme.

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.

Thus, each packet has format of

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

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

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

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,

$$P_0 = signature(H_1) || H_1 = E_{K_s}(H_1) || H_1,$$

and the signature $S = E_{K_s}(H_1)$ is 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_k, this node verifies the authenticity and integrity of P_k with the previously received and verified H_k. 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.

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 2.2, 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 1 to n. Only when a node X has received all packets from 1 to $k-1$ can it verify packet k immediately when X receives it.

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

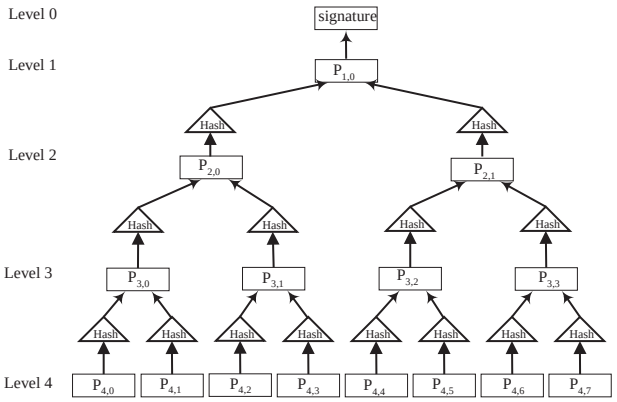


Figure 3: Signed Hash Tree Scheme. This hash tree structure has only 5 levels (m = 4), and every index packet contains 2 hash values (w = 2).

by overflowed buffers at any intermediate nodes in the path. A minor problem of hash chain scheme is it does not fit with efficient code dissemination protocol such as Deluge. Deluge sends a sequence of packets, then waits for ACK/NACK message back. However, chain scheme requires reliable delivery of every previous message, so the sender has to wait for feedback message from receiver and decides whether to re-send a packet or send next one. The hash chain scheme prevents the receiver from verifying a packet if its previous one is lost.

3.2 Hash Tree based Scheme

To achieve DoS-attack resilience and allow immediate verification of out-of-order packets, we propose another 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 1 to n to its neighbor nodes.

To enable nodes to authenticate and verify the integrity of code image packets quickly, even when the packets may 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 *index 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 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. Concretely, for a packet in level i where $1 < i < m$,

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

All packets at level m contain their respective data fragments of the code image (packets $P_{m,0}$ to $P_{m,n-1}$ correspond to the n data packets). 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 $P_{1,0}$ as well as verify the integrity of $P_{1,0}$. 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.

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 requirement **2a**, we see the hash tree scheme satisfies a weaker statement, i.e. applies more broadly:

2b. DoS-attack resilience. When a node received all packets from 1 to $k-1$, it can verify any packets from k to m with any order, where $m-k > k$.

Concretely, if there are w hash values in an index packet, there are w^l packets in level l. If a node receives all w^l packets in level l, it can receives the next w^{l+1} packets in level $l+1$ with any order. Thus, this hash tree 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 data 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. For example, if data packet $P_{m,2}$ is lost, all subsequent packets can still be authenticated, e.g. data packets $P_{m,3}$ and $P_{m,4}$.

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, there is a tradeoff because the underlying reliabilty mechanism saves many acknowledgment messages.

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 value of level $i-1$ packets in its SRAM, that means it should save all packets in level $m-1$ in its SRAM. For a large code image, this could become a concern. One way to solve this problem is to save these packets in EEPROM. However, as tested in [4], while reading data from EEPROM is fast, writing data into EEPROM is relatively slow.

Deluge segments the whole program code into pages, and each page contains a fixed number of packets. To receive packets in page k, a node has to reliably receive all packets in page 1 through $k-1$. Deluge uses the NACK message to transmit all packets in a page as we described previously. In this way, a node can temporally save all packets within a page into SRAM. For example, if it uses 1 KB SRAM memory, the size of a page is also 1 KB, and it 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*. However, this approach introduces more public key operations (one per page of code image). The later experiments show that these public key operations introduce significant delay.

3.3 Hybrid Scheme

To reduce the tree-based scheme's number of public key operations, we propose an improved 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.

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 mainly 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 Index packets in a hash tree

We would like to estimate the number of index packets needed for a hash tree. Let $I(n)$ to denote the number of index packets in a hash tree (excluding packet $P_{0,0}$).

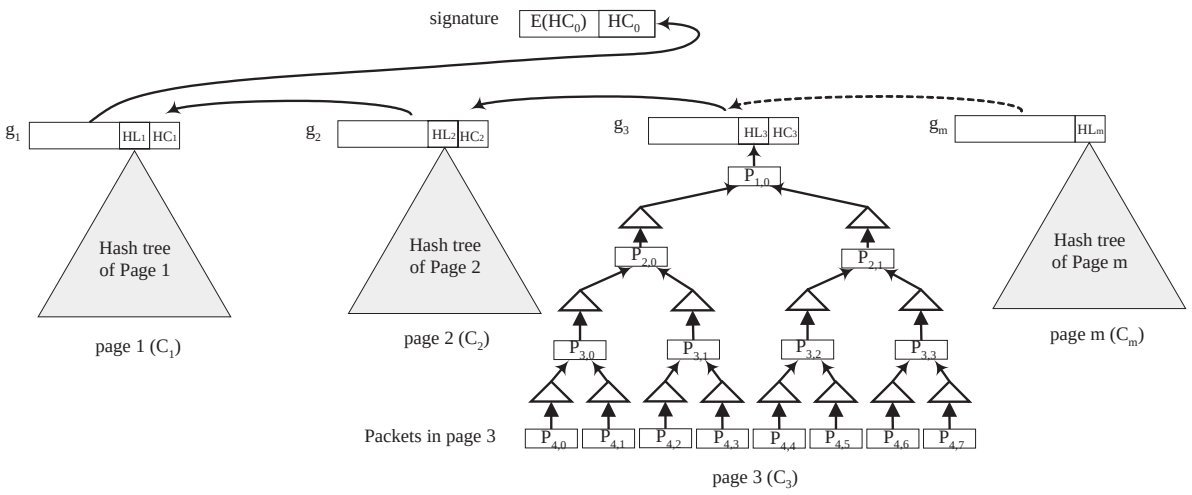


Figure 4: Hybrid Signed Scheme, combines hash chain structure for inter-page authentication and hash tree structure for internal page authentication.

Suppose there are n data packets. The height of the hash tree is $\lceil \log_w n \rceil$ (let's don't consider root node). If $\lceil \log_w n \rceil$ is an integer k, the number of index packets are $I(n) = (w^k - 1)/(w-1) \approx n/(w-1)$. Otherwise, the number of index packets are less than $I(n) \leq (w^{\lceil \log_w n \rceil} - 1)/(w-1)$.

Suppose $\lfloor \log_w n \rfloor = k$ where k is an integer. Let's denote n as that

$$n = \sum_{i=0}^{k} a_i \times w^{k-i}$$

Where $0 \leq a_i < w$ $(0 < i \leq k)$ and $1 \leq a_0 < w$.

For each group of packets w^{k-i}, they can form a hash tree with level of $k-i$. And, the a_i hash forests with level $k-i$ can form a hash tree with level $k-i-1$ by adding a new packet, and this packet can be used to form a tree with less $k-i-2$ with a_{i-1} forests. By this way, we can build one hash tree for all packets. The number of index packets of this tree is

$$\begin{aligned} I(n) &\leq \sum_{i=0}^{k} a_i \times \frac{w^{k-i}-1}{w-1} + k \\ &= \frac{\sum_{i=0}^{k} a_i \times w^k - \sum_{i=0}^{k-1} a_i + (w-1)k}{w-1} \\ &\approx \frac{n}{w-1} \end{aligned}$$

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

4.2 Estimation of f(k) 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 to 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 used to generate a MAC for each acknowledgment packets. The sender can verify the authenticity of each packet if the intruder cannot compromise a node. Notice that even though the intruder can compromise a node and launch an attack, it still has limited impact, as with the above DoS attack.

5. EXPERIMENTS

The goal of this experiment is to understand the costs of our security mechanism, and see if it is feasible for sensor network applications. Since prior work has already tested the memory and computational costs of public key algorithms and hash-based algorithms on sensor motes, we focus on comparing the communication overhead and delay of our hybrid scheme to the Deluge non-secure code propagation scheme as well as the hash tree and hash chain approaches. For the signed hash tree, we simulated the case that every page contains a single hard tree.

5.1 Overhead vs. Packet Loss Rate

Our first experiment was to measure the extra overhead cost of our secure reprogramming schemes compared with Deluge. In this experiment, we set the packet loss rate to range from 0% to 15%. The size of program code is 32 KB, which is usual for real applications, and the size of hash value is 4 bytes.

We run tests based on two types of sensor node configurations. The mica2 mote, which uses chipcon CC1000, has 29 bytes of packet. Each packet contains 6 hash values (remaining some space for other information). A node uses 1K bytes of RAM to save data in a page. The Tmote node, which uses chipcon CC2420, has 128 bytes of packet. Each packet contains 30 hash values. A node uses 4K bytes of RAM to save data in a page. We measured the total number of sent packets, and the total time to distribute code to all sensor nodes.

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. We simulated Deluge, hash chain, hash tree, and hybrid schemes. Looking at these figures, we see that the overhead of hybrid and tree schemes are

From these experiments we see that both hybrid and tree schemes cost modest message overhead coompared with Deluge protocol, however, the cost of chain scheme is very high. For instance, under Tmote configuration, when packet loss rate is 5%, hybrid scheme costs extra 43% messages compared with Deluge, which chain scheme costs extra 308% messages compared with Deluge. The reason is in the NACK mechanism used by Deluge, hybrid and tree schemes. They send all packets in a page, and wait for one ACK/NACK message back from each node. However, in chain scheme, the sender has to wait for the ACK message from each node, then it can decide whether to send next packet or repeat the current packet, as we know that a node cannot verify a packet if its previous packet was lost. The ACK message based on each packet creates a large amount of traffic. On the contrast, the extra message in the tree-structured index packets only create modest extra traffic. We think the chain

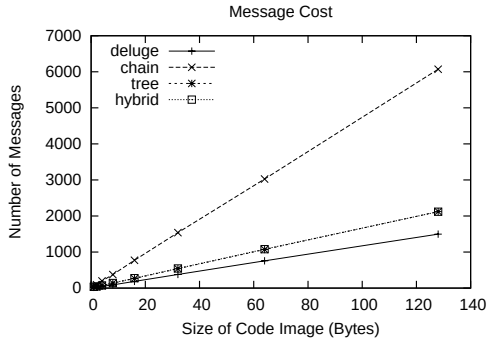


Figure 6: Message cost of reprogramming schemes under different program code size

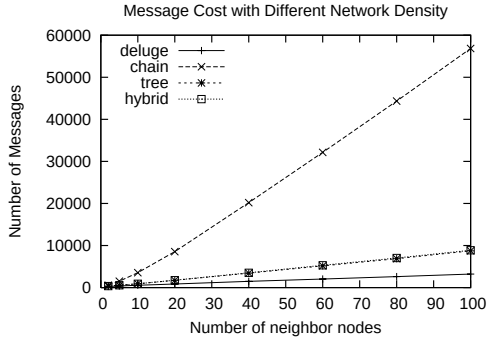


Figure 7: Message overhead as a function of network density.

scheme may work in the situation that the packet loss rate is extremely low, so the sender can send a number of (suppose m) packets and wait for a NACK message. If a packet k is lost, the sender re-sends packets from $k+1$ to m.

By comparing figure 5 (a) and (b), we see that when the size of packet increase, not only the total amount of messages decreases, the overhead of hybrid and tree scheme also decreases. This is because each index packet can contains more hash values.

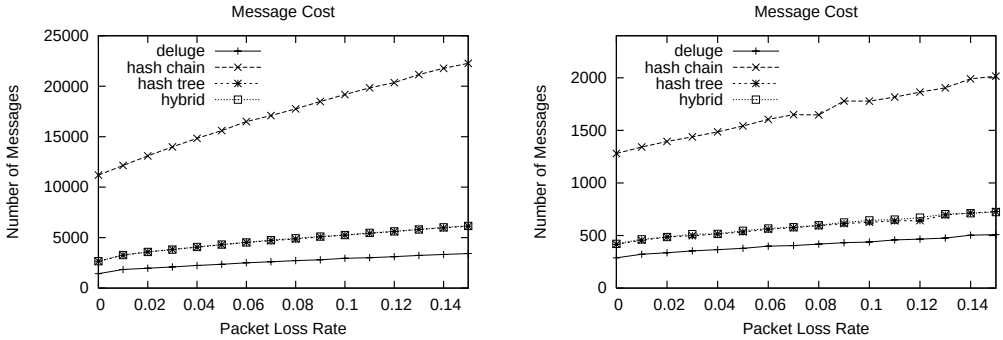
5.2 Overhead vs. Size of Code Image

Figure 6 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 5%. For all schemes, we see that the number of messages linearly increases as the size of code increases, and the ratio of extra cost is about 45% for hybrid scheme. This linear increase as a function of code size conforms with the experimental results obtained in the Deluge paper. Hybrid scheme and tree scheme has almost same result since the hybrid scheme only costs a few amount packets than tree scheme does.

5.3 Overhead vs. Network Density

In this experiment, we measured the number of exchanged messages when the density of network changes, while the packet loss rate is set as 5%. For instance, we changed the number of neighbor nodes from 2 to 100.

Figure 7 depicts the message cost as a function of net-



(a) Mica2 configuration (b) Tmote configuration

Figure 5: Message cost of reprogramming schemes under different packet loss rates.

work density, e.g. number of neighbor nodes. We see that when the density changes, the hybrid scheme still costs far less amount of messages compared with chain scheme. For example, when a base station broadcasts a code image to 19 nodes, the total messages of Deluge is about 855, the total messages of hybrid scheme is about 1756, and chain scheme is about 8547.

5.4 Delay vs. Packet Loss Rate

Figure 8 shows the time delay for reprogramming. We set the packet transmission delay at about 50 milliseconds, and the computing time of signature verification with ECC is about 13 seconds. We see that the hybrid scheme still keeps modest overhead compared with Deluge. The overhead of the chain scheme is much higher than hybrid, since it needs to send many more packets. The total delay of the tree scheme is much higher than chain scheme, because it has to run public key algorithms multiple times to verify pages of packets in one code image. When the packet loss rate is 5%. Deluge takes 2.2 minutes to send all data in mica2 motes, hybrid scheme uses 3.3 minutes, chain scheme uses 7.2 minutes, and tree schemes uses 10.6 minutes.

6. RELATED WORK

Because it is far more efficient than public key algorithms, hash functions have been widely used to authenticate transmitted data. For example, K. Fu *et al.* proposed a secure read-only file system with hash tree [8]. A. Perrig proposed the BiBa scheme which is based on one-way functions and the birthday paradox [18]. 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 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 resourc-poor sensor nodes.

V. Gupta *et. al* implemented the SSL protocol on sensor node with RSA and elliptic curve algorithms, called Sizzle [10]. 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.

7. CONCLUSIONS AND FUTURE WORK

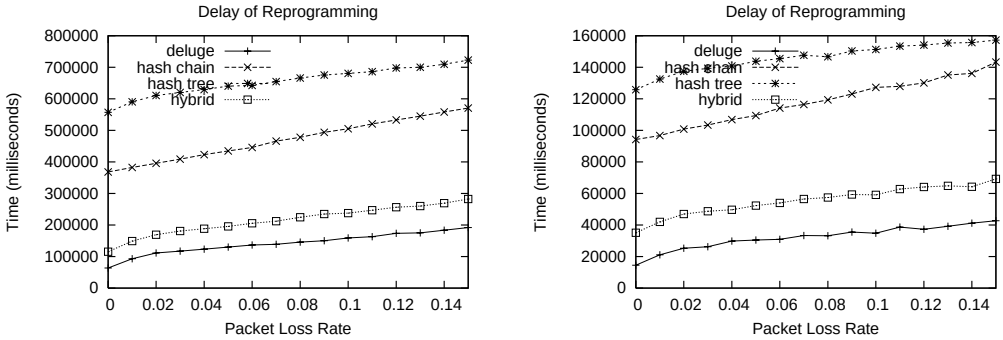
This paper develops a novel *secure* code propagation protocol for wireless sensor networks that employs private key signing of the root of a joint structure comprised of hash chains for inter-page authentication and hash trees for intra-page authentication. This solution has the following advantages: 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 delay due to fast hash-based verifications for all subsequent packets 3) low overhead since the hash tree tolerates disorder and reduces unnecessary retransmissions. Our simulations confirm that our solution incurs substantially less delay than the signed hash tree scheme, and substantially less overhead than the loss-sensitive signed hash chain. Our schemes are light weight enough to be feasible for current sensor network platforms, and are well-adapted to practical code propagation protocols such as Deluge. In the future, we plan to implement the hybrid scheme on a multi-hop testbed of the Tmote TELOS motes.

8. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable comments.

9. REFERENCES

[1] Crossbow website. http://www.xbow.com.
[2] Tmote. http://www.moteiv.com.
[3] R. Adler, P. Buonadonna, J. Chabra, and et al. Design and deployment of industrial sensor networks: Experiences from the north sea and a semiconductor plant. In *SenSys'05*, San Diego, California, USA, November 2005.
[4] 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*



(a) mica2 configuration (b) Tmote configuration

Figure 8: Delay of reprogramming under different packet loss rates.

(SecureComm 2005), pages 289–299, Athens, Greece, September 2005.

[5] 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.

[6] P. K. Dutta, J. W. Hui, D. C. Chu, and D. E. Culler. Securing the deluge network programming system. In *the Fifth International Conference on Information Processing in Sensor Networks (IPSN'06)*, April 2006.

[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] K. Fu, M. F. Kaashoek, and D. Mazières. Fast and secure distributed read-only file system. *Computer Systems*, 20(1):1–24, 2002.

[9] 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.

[10] 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.

[11] 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.

[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.

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

[16] 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.

[17] P. Ning and A. Liu. Tinyecc: Elliptic curve cryptography for sensor networks. http://discovery.csc.ncsu.edu/˜pning.

[18] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In *8th ACM Conference on Computer and Communications Security*, Philadelphia, PA, USA, November 2001.

[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] 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.

[22] 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.

[23] 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.