Efficiently Authenticating Code Images in Dynamically Reprogrammed Wireless Sensor Networks

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

Abstract

For large scale wireless sensor networks (WSNs), reprogramming sensor nodes through the wireless channel is an important capability. To avoid reprogramming false or viral code images, each sensor node needs to efficiently authenticate its received code image before propagating it. Public key schemes based on elliptic curve cryptography are feasible in WSNs, yet are still very expensive in terms of memory and CPU consumption. In this paper, we propose a hybrid mechanism that combines the speedy verification of hash schemes with the strong authenticity of public key schemes. A hash tree is computed from packetized code and its root is signed by the public key of the base station. Each sensor node can quickly authenticate the data packet as soon as it is received. Simulation shows that the proposed secure reprogramming scheme adds only a modest amount of overhead to a conventional non-secure reprogramming scheme, namely Deluge, and is therefore feasible and practical in a WSN.

1 Introduction

Applications of wireless sensor networks (WSNs) are becoming increasingly diverse, ranging from habitat monitoring to indoor sensor networks, and from battlefield surveillance to seismic monitoring of buildings. 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.

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. In addition, applications that require privacy and anonymity 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 *efficient*, namely frugal in terms of memory footprint, energy consumption, overhead, and processor usage. As far as we are aware, this is the first published full paper that builds a secure and efficient code propagation mechanism for WSNs. Existing work, such as the Deluge code propagation protocol [8] developed for WSNs, is not secure.

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 [3], allowing all internal information such as 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 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 computationally intensive and 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 [10, 7], ECC-class public key authentication is still only practical if used sparingly.

Our approach is to combine the best properties of both public key schemes and faster hashed verification schemes to build secure and efficient dynamic programming. 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.

The paper is organized as follows. In section 2, we discuss the security requirements for dynamic reprogramming of WSNs. In section 3, we present our scheme of secure dynamic programming of WSNs. Section 4 presents the simulation results of our scheme, highlighting the modest overhead of our approach. Section 7 concludes the paper.

2 Security Requirements

2.1 Assumptions

We assume our security scheme is built on standard sensor nodes like crossbow motes [1]. 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.

Several groups of researchers have implemented RSA and Elliptic Curve (ECC) on mica motes [7, 10, 14, 6, 11, 5]. Up to now, the best result reported by N.Gura *et.al* shows that public key encryption/decryption runs hundreds or thousands milliseconds, and consumes hundreds of bytes of SRAM [7]. P. Ning *et.al* provide source code of ECC which runs 12 to 16 seconds to verify a signature on micaz motes [11]. On the contrary, the block cipher based hash functions run about hundreds or thousands of times faster than public key schemes.

In this paper we assume that the code images are propagated from a base station to every node in the network. The whole code image is segmented into a sequence of data packets from 1 to n, and these packets need to be reliably delivered to every node. Because the data transmission in a wireless sensor network is unstable and the packet loss rate is high, reliable data transmission mechanisms must be provided, such as acknowledgement messages (ACK) or negative acknowledgement messages (NACK). We assume these are provided hop by hop as in the Deluge protocol.

2.2 Threat Model

The goal of an adversary is to reprogram its own code into sensor nodes or launch denial of services attacks to 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 them, and to inject fake packets to sensor nodes nearby. However, we assume the base station is rich in computing resources and it is securely protected. An adversary cannot compromise a base station.

2.3 Security Goals

The goal of this paper is to efficiently protect the authenticity and integrity of propagated code images. In particular,

- 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.
- Every node can verify the code image as soon as it receives it, even though some packets may arrive out-of-order due to packet losses. Otherwise an adversary can potentially launch denial of services attacks against the sensor network due to delayed authentication.
- The resource consumption of the proposed security mechanism must be light weight in terms of communication, computing and memory usage.

3 Hash-tree Based Code Image Authentication Scheme

3.1 Underlying Reliable Data Transmission Scheme

To build our security scheme, we assume that the underlying reliable data transmission mechanism is similar to Deluge: the whole code image is divided into n packets. 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.

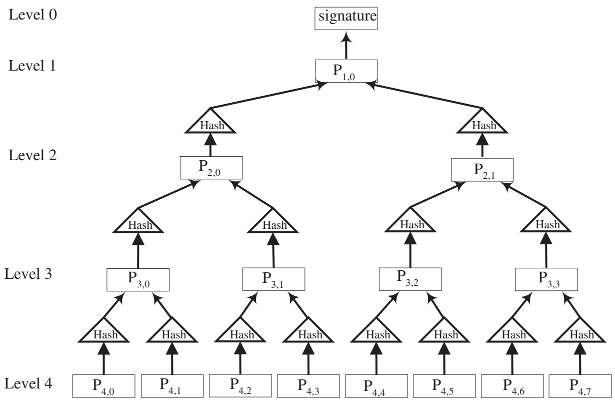

Figure 1. Hash Tree Scheme. This hash tree structure has only 5 levels (m = 4), and every index packet contains 2 hash values (w = 2).

3.2 Description of Algorithm

The principle of our secure dynamic reprogramming scheme is to apply public key operations in a sparing manner, yet allow the sensor node to quickly verify the authenticity and integrity of each data packet. Figure 1 illustrates our basic tree-structured approach. 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 values $Hash(P_{i+1,j*w})$, $Hash(P_{i+1,j*w+1})$, $\ldots$, $Hash(P_{i+1,j*w+w-1})$. 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, (from $P_{m,1}$ to $P_{m,n}$), and the packets internal to the tree are called *index packets*. By this way, every data packet has a hash value, and that value is linked to a hash tree structure. The root value at the top of the tree, level 0, is the level 1 hash signed with the private key of the base station.

A sender transmits packets from the low levels to the high levels, i.e. from level 0 to level m. When a node sends a code image to its neighbor nodes, it also sends packets from the low levels to the high levels. For each level, the receiver sends back an acknowledgement message to inform the 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, and it will use the signature to authenticate the data in the level 1 packet, and save all hash values in that level 1 packet. When a packet in level 2 is received, it can use the saved hash values to authenticate level 2 packets, and saves the hash values in this level 2 packet, and so on. Eventually, it can authenticate every packet in level m with saved hash values that were received from level $m-1$ 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.

This approach has several advantages. First, 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. Second, the hash tree enables every node to verify each packet immediately, even if it didn't receive some packets at the same level. When a data packet arrives, a quick hash of its contents can be compared to the previously saved hash to verify authenticity.

One cost of the hash tree scheme is the extra index packets that need to be transmitted. However, this 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 all packets at level $i-1$ in its SRAM. 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 [3], 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.

This hash-tree mechanism securely protects and verifies the authenticity of the tree of hashes. An adversary will be unable to modify hashes in the tree without being detected. Similarly, an adversary will be unable to modify the code packets without being detected by using the hashes.To prevent replay attacks of old yet valid code images, a version number and/or time stamp can be included in the signed root hash. Each node only accepts code that is at or higher than the current version number.

4 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, our focus was on comparing the message overhead cost of our security protection scheme to the Deluge non-secure code propagation scheme. The intent was to discover if the tree-structured approach could achieve its security goals while introducing only modest overhead.

We simulated the situation that a source node disseminates a program code to its direct neighbor nodes, and measured the number of packets sent during the process under different packet loss rates and code sizes. There is one

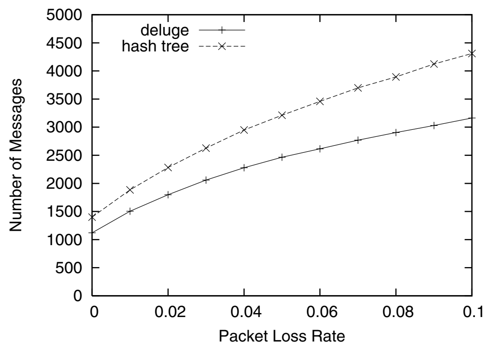

Figure 2. Message cost of original deluge scheme and hash tree scheme under different packet loss rate.

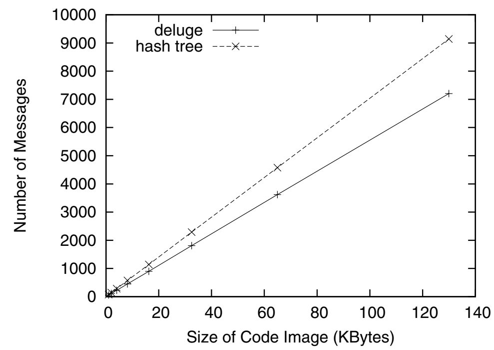

Figure 3. Message cost of Deluge and our hash tree scheme under different program code sizes.

source and twenty sensor nodes, and the source can communicate with every node. This simple test gives us a rough estimate of the per-hop costs of code dissemination, i.e. at each hop the same behavior of reprogramming would be initiated by a source to reprogram its direct neighbors. At the first hop, the base station would program its one-hop or level-one neighbors; at level one, each level one neighbor acts as a source to reprogram its level two neighbors (two hops from the base station); and so on. In future work, we intend to confirm these initial results by evaluating over multi-hop topologies, e.g. WSN routing trees.

4.1 Packets Loss Rate

Our first experiment was to measure the extra overhead cost of our secure reprogramming scheme compared with Deluge. In this experiment, we set the packet loss rate to range from 0 to 10%. 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. We measured the total number of sent packets. There is a tradeoff between security and efficiency: the larger the size of the hash value, the greater the security and the higher the energy cost. The new Telos sensor node [2] supports 128 bytes of packet size, which can contain larger sized hash values.

Figure 2 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 for current sensor network platforms. For example, when the packet loss rate is 2%, the packets sent in Deluge equal about 1800, and the packets sent in the hash tree scheme equal 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 when amortized over the entire life of a sensor node.

4.2 Size of Code Image

Our second experiment tested 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 at 2%. For both Deluge and the hash tree scheme, we see that the number of messages linearly increases as the size of the code increases, and the ratio of extra cost is about 28%. For Deluge, as the size of the code linearly increases, the time for sending them also linearly increases. For the hash tree scheme, since the ratio between index packets and data packets is constant, the total data size also linearly increases.

5 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 [4]. A. Perrig proposed the BiBa scheme which is based on one-way functions and the birthday paradox [12]. However, BiBa is still too heavy to be applied to WSN reprogramming. First, the base station needs to generate and maintain a large number of SEALs since a program is composed of many data packets. Second, each data packet needs to contains k SEALs if BiBa uses k-way collisions. A. Perrig *et. al* also proposed μTESLA [13] protocol for source authentication in data dissemination through lossy channel. However, μTESLA 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 [9]. 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 [6]. 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 employ end-to-end data transmission schemes.

6 Discussion and Future Work

In the Deluge protocol, the whole program code is segmented into pages, and each page contains a fixed number of packets. To receive packets in page i, a node has to reliably receive all packets in page 1 through $i - 1$. To adapt our 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. This approach introduces more public key operations. Thus, one of the weaknesses of applying this scheme directly to Deluge on a per-page basis is the increased number of public key operations, which can become a concern for large code images. As part of our future work, we plan to address this weakness while integrating our hash tree within the Deluge reprogramming protocol, and running it on a real sensor network testbed.

7 Conclusion

Authentication of code images received over a wireless channel is an important capability for many WSN applications. This paper is a first attempt at developing an efficient secure code propagation protocol for wireless sensor networks. In this paper, we proposed a hybrid security scheme that employs a public key algorithm and a hash tree to quickly and efficiently authenticate program code images. Our experiments show that the proposed mechanism tolerates disorder caused by packet loss in wireless communication, and introduces little overhead. Our scheme is light weight enough to be feasible for current sensor network platforms.

8 Acknowledgments

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

References

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

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] C. Karlof, N. Sastry, Y. Li, A. Perrig, , and J. Tygar. Distillation codes and applications to dos resistant multicast authentication. In *the 11th Annual Network and Distributed Systems Security Symposium (NDSS 2004)*, San Diego, CA, USA, February 2004.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.