Distributed Authentication of Program Integrity Verification in Wireless Sensor Networks

Katharine Chang and Kang G. Shin
Department of Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor, MI 48109-2122
{katchang, kgshin}@eecs.umich.edu

Abstract

Security in wireless sensor networks has become important as sensor networks are being used for an increasing number of applications. The severe resource constraints in each sensor make it very challenging to secure sensor networks. Moreover, sensors are usually deployed in hostile and unattended environments, and hence, are susceptible to various attacks, including node capture, physical tampering, and manipulation of the sensor program. The authors of [15] proposed a soft tamper-proofing scheme that verifies the integrity of the program in each sensor device, called the Program Integrity Verification *(PIV). This paper addresses how to authenticate PIV Servers (PIVSes) in a fully-distributed manner. Our distributed authentication protocol of PIVSes (DAPP) uses the Blundo scheme [5] and allows sensors to authenticate PIVSes without requiring commonly-used trusted third parties, such as authentication servers (ASes), in the network. We implement and evaluate both the DAPP and the PIV on Mica2 Motes and laptops. We also analyze the security of DAPP under different attack models, demonstrating its capability to deal with various types of attacks.*

1. Introduction

Wireless sensor networks are becoming important for many emerging applications such as military surveillance, alerts on terrorist attacks, burglar threats, fire, earthquake, and volcanic emergency systems. Therefore, security in sensor networks becomes very important. However, due to the limitations of each sensor device's battery energy, memory, computation, and communication capacities, achieving security in a sensor network has become a great challenge. Moreover, sensor networks are often composed of a large number of small low-capacity devices and deployed in hostile and unattended environments, thereby making them susceptible to physical capture and compromise, which, in turn, makes it difficult to keep the integrity of the original sensor program. Even just one compromised sensor can make the entire network insecure. Thus, making sensor devices *tamper-resistant* is a must. Another important issue is the *authentication* of the communication between sensor nodes and the servers in the network. A sensor node has to verify that the messages had really been sent by the genuine sender, and protect itself from communicating with a malicious server that pretends to be a legitimate one.

For the purpose of securing a network, there are usually two lines of defense. The first line is intrusion prevention. Typical intrusion prevention measures, such as authentication and encryption, can be used to prevent external nodes from disrupting the network. However, intrusion prevention can only counter outsider attacks, and cannot handle insider attacks. For example, if a sensor node is physically captured and compromised, then the attacker can obtain the cryptographic keys held by the sensor node. Thus, intrusion prevention measures that often require sharing secrets between nodes will not help defend against inside attacks.

The second line of defense is an intrusion detection system (IDS) that can detect insider attacks of compromised nodes in the network. Once an intrusion is detected, a response measure can be made to minimize damages to the network. Given the new vulnerabilities that continue to be discovered, the IDS has to be effective and efficient in identifying attacks, and then successfully responding to them.

We implement and demonstrate a soft tamper-proofing scheme based on the Program-Integrity Verification (PIV) [15] on Mica2 sensor devices [3]. We also propose a distributed authentication protocol called DAPP, for sensors to communicate with PIVSes securely without the AS infrastructure assumed in [15]. DAPP acts as the first line of defense for the network by enabling a senor node to prove the identity of a server before communicating with it. The sensors will then use the PIV protocol to verify their program with the authenticated PIVSes, and will be allowed to join the network only after passing the verification test. The

1-4244-0423-1/06/$20.00 ©2006 IEEE

authentication here means the degree to which one communicating party can ensure the valid identity of another party in the network. The DAPP we propose is used to enable sensors to validate a PIVS to be safe to use.

The remainder of this paper is organized as follows. For completeness we first give an overview of the PIV protocol [15] in Section 2. We present the motivation of this work as well as the assumptions we made, and give an overview of DAPP design in Section 3. Section 4 describes the details of DAPP. Section 5 analyzes the security of DAPP and PIV, which is followed by our DAPP and PIV implementations in Section 6. Section 7 presents the performance evaluation of DAPP. Finally, we discuss the related work in Section 8 and conclude the paper along with future work in Section 9.

2. Background: Overview of PIV

The PIV protocol [15] verifies the integrity of the program and data stored in a sensor device. It is purely software-based protection from physical attacks in sensor networks. It is also lightweight in that it is only triggered infrequently, and the verification of each program incurs a very small overhead, so it will not degrade normal sensor functions and services.

The PIV Server (PIVS) verifies the integrity of the program of each sensor device using the digests of the sensor programs maintained in a database. Conventional authentication servers (ASes) are used by sensors to ensure the PIVSes are authentic, and are safe to communicate with and execute the codes received from them.

The PIV protocol performs: (1) authentication of each PIVS via the AS; (2) transmission and execution of PIV code (PIVC); (3) program verification by PIVC and PIVS. The sensor that wants to join the network will first ask an AS for authentication of a PIVS. If authentication succeeds, the sensor will then ask the PIVS for verification of its program. During verification, the PIVS will send a mobile agent, PIVC, to the sensor, and then computes a hash value from the digest of the sensor program maintained in its database. After the sensor receives the PIVC from the PIVS, it executes the PIVC on its program to compute a hash value. The hash value will then be sent back to the PIVS for verification. The PIVS finally checks if the two hash values match to determine the integrity of the sensor's program. If the sensor passes the verification, then the PIVS registers it in its database PIV_DB which contains all successfully-verified sensor IDs. Otherwise, the sensor will be locked, with its ID deleted from PIV_DB if it had passed PIV before, becoming unable to join the network. Fig. 1 depicts the interactions among AS, PIVS, and the sensor during PIV.

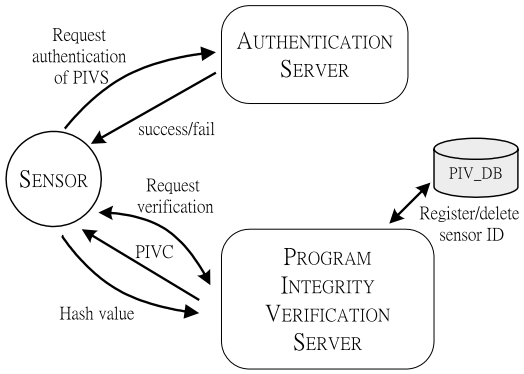

Figure 1. Interactions among AS, PIVS, and the sensor during PIV.

3. Motivation, Assumptions, and an Overview

We now state our motivation and the assumptions we made, and give an overview of our DAPP design.

3.1. Motivation

Our main goal is to eliminate the need for ASes in the PIV infrastructure. As the AS is needed for sensors to authenticate PIVSes, it will easily become a bottleneck for reliability, security, and communication. Also, requiring a centralized service is inconsistent with the distributed structure of sensor networks. Moreover, sensors that are deployed near the AS will consume more energy to route messages for other sensors, and will die of exhausted batteries first. So, having a centralized AS will not scale well to a large network.

The communication traffic to/from a single AS can, of course, be reduced if the AS is replicated in the network. However, since ASes act as trusted third parties, they are presumed to be trusted and secure. Therefore, ASes will need secure computing platforms to protect the servers from attacks. Also, as ASes require more memory, more energy, and stronger computation power than sensor devices, deploying more ASes in the network will increase the cost for deployment. For these reasons, we would like to remove the need for ASes from the PIV infrastructure, and distribute the authentication function to PIVSes themselves.

3.2. Assumptions

Listed below are the assumptions we make in this paper.

A1. Sensors and PIVSes are deployed randomly in their coverage area. Therefore, we have no prior knowledge about the neighbors or the location of each sensor and PIVS before deployment.

A2. Sensors and PIVSes in the network have unique node IDs.

A3. PIVSes are equipped with more energy, larger memory, and more computation and transmission power than sensor devices.

A4. PIVSes have all of the sensor programs stored in their memory before deployment. The sensor programs stored at the PIVSes are used for verification of the integrity of the sensors' programs.

A5. PIVSes have dual radio interfaces that the radios for communication with sensors and other PIVSes will not interfere with each other.

A6. After deployment, each PIVS has at least t neighboring PIVSes.

3.3. Design Overview

We exploit the PIV [15] protocol summarized in Section 2 to protect the integrity of sensor programs. In order to remove ASes from the PIV infrastructure, we use PIVSes to perform the AS functions and to authenticate one another to achieve the distributed authentication of PIVSes.

For PIVSes to authenticate one another before a sensor starts to use the PIV protocol with one PIVS to verify its program, PIVSes need to share some secrets with one another. We use the idea of Blundo scheme [5] for establishing pairwise keys in the network. Before deployment, each PIVS and sensor node will be loaded with a function to generate pairwise keys. After deployment, sensors and PIVSes can use the loaded function to establish pairwise keys with any node in the network.

Besides the distribution of the AS function to PIVSes, we also want to reduce the total number of messages exchanged in the network and the energy consumed by each sensor with the use of distributed authentication.

4. DAPP

In the original PIV design [15], protection of a sensor from a malicious server/code disguised as a PIVS/PIVC is achieved by using the AS that acts as a trusted third party. The AS can help a sensor ensure that the PIVS is authentic, and the PIVC it receives from the PIVS is safe to execute. Here, we propose a protocol, DAPP, for the sensors to authenticate PIVSes without using such an AS.

For a sensor network that has a maximum of n sensor nodes and s PIVSes, all sensors and PIVSes in the network have unique node IDs. All PIVSes store all the sensor programs for verification of each sensor.

Listed below are notations used in the rest of the paper.

- $A, B, \ldots$ are principals, such as PIVSes or sensor nodes.
- N_A is a nonce generated by A, which is also a randomly-generated number that is unpredictable and is used to achieve freshness.
- K_{AB} is the shared pairwise key between A and B.
- MAC(K, M) is the message authentication code (MAC) of message M generated with a symmetric key K.
- f is a symmetric bivariate k-degree polynomial for establishing pairwise keys in the network.

We apply the Blundo scheme [5] for setting up PIVSes' and sensors' pairwise keys. We can also use the pairwise key establishment scheme in LEAP [22]. Next, we describe the protocol using the Blundo scheme.

Recall that the PIVSes are computationally more powerful than sensors, and their memory, energy, and transmission capabilities are also greater than sensors'. Suppose, after the deployment of PIVSes, each PIVS has at least t neighbor PIVSes.

4.1. Initialization and PIVS Discovery Phase

We now describe the initialization and PIVS discovery phase of DAPP. Note that nodes can be added to the network later, and apply the same phase to join the network.

A. Pre-deployment Initialization Phase

Before the deployment of sensors and PIVSes, all the nodes are secured. In this phase, we establish the identity of each PIVS/sensor and its security basis. There are a maximum of n sensor nodes and s PIVSes. Not all sensor nodes and PIVSes need to be deployed at once; some can join later to the network. All s PIVSes have the n sensor programs stored in their memory.

A key server randomly generates a symmetric bivariate k-degree polynomial $f(x, y) = \sum_{i,j=0}^{k} a_{ij} x^i y^j$ over a finite field F_q, where q is a prime number that is large enough to accommodate a cryptographic key. For each PIVS and sensor node A, the key server computes $f(A, y)$, and loads the $k + 1$ coefficients as a function of y to node A. The function is used for nodes to establish pairwise keys with other nodes later in the network.

B. Post-deployment PIVS Discovery Phase

The sensors and PIVSes are then randomly deployed to the field. After the deployment of sensors and PIVSes, they can discover neighbor PIVSes within their communication range in this phase.

After deployment, each PIVS will periodically broadcast a PIVS Beacon Message containing its ID. The neighbor PIVSes that receive this PIVS Beacon Message can establish shared pairwise keys with the PIVS. When PIVS B wants to establish a pairwise key with PIVS A, it computes $f(B, A)$ by substituting y in $f(B, y)$, the function preloaded by the key server in the pre-deployment initialization phase, with A. Likewise, A can compute $f(A, B)$. Since $f(x, y)$ is a symmetric function, $f(A, B) = f(B, A)$, therefore A and B can establish a pairwise key $K_{AB} = f(A, B)$ between them. The two PIVSes can now verify one another's authenticity using the shared pairwise key K_{AB}.

$$\begin{aligned} A \rightarrow * &: \text{PIVS Beacon Message}(A) \\ B \rightarrow A &: B, N_B, \text{MAC}(K_{AB}, B|N_B) \\ A \rightarrow B &: A, N_A, \text{MAC}(K_{AB}, A|N_A|N_B) \\ B \rightarrow A &: B, \text{MAC}(K_{AB}, B|N_A) \end{aligned}$$

Note that here we use MAC for message authenticity and integrity. MAC can be viewed as a secure cryptographical checksum for the message. The sender and the receiver must both have a secret shared pairwise key to compute the MAC. The computed MAC value helps the receiver detect any change to the message content.

We optimize the protocol by including the nonce implicitly in the MAC computation. The receiver PIVS recomputes the MAC and compares it with the MAC value it received. If the values match, then the receiver PIVS can be sure of the sender PIVS's identity. Otherwise, it will reject the sender PIVS's messages. After two PIVSes verified the authenticity of each other with their shared pairwise key, they will add the ID of each other to their PIVS Reference List.

Moreover, the PIVS Beacon Messages allow the newly-deployed sensors to receive the information about the PIVSes within its communication range. If a newly-deployed sensor doesn't receive any PIVS Beacon Message for a certain period of time, it will broadcast a PIVS LookUp Message to search for a PIVS in its transmission range. Any PIVS in the network that receives the PIVS LookUp Message will then broadcast the PIVS Beacon Message again, or just unicast it directly to the sensor.

4.2. PIVS Authentication Phase

After the initialization and PIVS discovery phase, the sensors that want to join the network need to authenticate the PIVSes before starting PIV and then have their programs verified. In this phase, the sensors will authenticate PIVSes which will then authenticate each other for the sensors.

Based on the strength of signals (PIVS Beacon Messages) received from all the PIVSes, a sensor node can choose the one that is closest to it to verify its program. Sensors will first choose PIVSes that are one-hop away from it to authenticate. If no such PIVSes exist, then sensors can communicate to PIVSes that are multiple hops away via secure routing [11]. The sensor will first authenticate the PIVS, and if the PIVS is trustable, the sensor will start the PIV protocol [15] with it. However, if the PIVS fails to authenticate itself, the sensor will choose another PIVS to verify its program with, and restart the authentication phase with the other PIVS.

For sensor E to authenticate PIVS A, it will first compute the shared pairwise key $K_{AE} = f(E, A)$ with A. Sensor E will then send a PIVS Auth Message to A, which includes a randomly-generated nonce N_E and the MAC value of N_E computed using K_{AE}. Upon receiving the message, PIVS A will generate the shared key $K_{AE} = f(A, E)$, and verify the PIVS Auth Message sent by E. If the message is authentic, then PIVS A will send PIVS Ref Messages including sensor E's ID, nonce N_E, and the MAC value computed using the shared pairwise keys to the PIVSes on its PIVS Reference List. The PIVSes that receive the PIVS Ref Messages will check the authenticity of the message it received. To check the authenticity of a PIVS Ref Message, the PIVS simply recomputes the MAC value using the shared pairwise key, and compare the MAC value included in the message. If the two values match, then the message is really sent from PIVS A since it has the correct pairwise key. However, if the two values don't match, then the message is sent from a malicious PIVS faking to be A, and the message will be discarded.

After a reference PIVS authenticates A, it will grant an authentication ticket to A which can then provide the authentication ticket to E, proving its authenticity. Each authentication ticket includes the reference PIVS's ID, and the MAC value of N_E computed using the reference PIVS and sensor E's pairwise key. One PIVS needs to show N_{auth} authentication tickets to the sensor in order to pass the authentication. The complete protocol of the PIVS Authentication Phase is shown below.

$$\begin{aligned} &E \rightarrow A : \text{PIVS Auth Message}(N_E, \text{MAC}(K_{AE}, N_E)) \\ &A \rightarrow B : \text{PIVS Ref Message}(E, N_E, \text{MAC}(K_{AB}, E|N_E)) \\ &B \rightarrow A : E, \text{AuthTicket}(B, \text{MAC}(K_{BE}, N_E)), \\ &\qquad \text{MAC}(K_{AB}, E|\text{AuthTicket}(B, \text{MAC}(K_{BE}, N_E))) \\ &..... \\ &A \rightarrow E : \text{AuthTicket}(B, \text{MAC}(K_{BE}, N_E))| \\ &\quad \text{AuthTicket}(C, \text{MAC}(K_{CE}, N_E))|..., \text{MAC}(K_{AE}, N_E) \end{aligned}$$

For PIVS A to authenticate another PIVS B, A needs to hold a pairwise key with B. Therefore, if B has received PIVS Ref Message from A, it is on A's PIVS Reference List, and should share a pairwise key with A. When B grants A the authentication ticket, it will also include the

MAC of sensor E's ID and the authentication ticket computed using A and B's pairwise key K_{AB}. If the MAC value of B's message matches, then A is sure that B is also trustable, and will use the authentication ticket issued by B. After A receives N_{auth} authentication tickets issued by its reference PIVSes, it will forward the authentication tickets along with the MAC value of N_E computed using the pairwise key K_{AE} to sensor E for authentication.

When sensor E receives the response from PIVS A, it will first check if A has replied with the correct MAC value of N_E. If the value is correct, then E will continue to check the N_{auth} authentication tickets. If the value is incorrect, E will conclude that A failed the authentication. From the N_{auth} authentication tickets issued by A's referenced PIVSes, the authentication result must mask the effects of d or less malicious PIVSes. If we consider Byzantine failures in the network, then $t \geq N_{auth} \geq 3d + 1$, where t is the minimum number of neighbor PIVSes each PIVS will have after deployment. If we don't consider Byzantine failures, then $t \geq N_{auth} \geq 2d+1$. Thus, the design parameter N_{auth} needs to be determined by the PIVSes' failure model.

If A provides a correct MAC value, then sensor E will check the correctness of the authentication tickets and use a simple majority rule to determine A's authenticity. If more than $N_{auth}/2$ of the authentication tickets have the correct MAC value, then E will conclude that A is trustable. Otherwise, E will try to verify its program with another PIVS, if any, and restart the authentication procedure with that PIVS.

5. Security Analysis

We first discuss the network survivability in the event of sensor and PIVS compromises, and then analyze the security of DAAP and PIV against various attacks.

5.1. Survivability

After compromising a sensor or a PIVS, the adversary can discover the node's keying materials, such as the preloaded key functions. If a PIVS is found to have been compromised, it can be evicted by using a revocation scheme based on cooperation of its neighbor PIVSes. On the other hand, if the compromise of a sensor is detected/suspected, the sensor will be required to re-verify its program using PIV. The sensor will be excluded from the network if it fails to pass PIV.

For the case when the compromise of the sensors and PIVSes go undetected, we need to analyze the survivability of the network. For this, we will consider the general attacks an adversary can mount after it compromises a node.

Since each node in the network is only preloaded with the keying function for it to establish pairwise keys with other nodes in the network, revealing such a function will only allow the attackers to pretend to be that node. The Blundo scheme [5] is proven to be unconditionally secure and k-collusion resistant. That is, when no more than k nodes are compromised, the adversary will know nothing about the pairwise key between any two uncompromised nodes in the network. However, if more than k nodes have been compromised, then the symmetric bivariate k-degree polynomial will be revealed, and the adversary will know all the pairwise keys in the network. Therefore, it is important to choose a large enough k for the polynomial to generate the pairwise keys. As mentioned in [21], for the current generation of sensor nodes, k can be around 200.

5.2. Defense Against Various Attacks in Sensor Networks

We now describe how DAPP and PIV can defend against various attacks in sensor networks.

A. Defense Against Passive Attacks

In DAPP, all messages are sent out in plain text, along with the MAC value of the message. Even though an attacker can eavesdrop on the messages, the only content that is revealed is the nonces that the sensors and PIVSes exchange. Therefore, the attackers will not gain any insight by eavesdropping on the network.

After a sensor authenticates a PIVS, all the messages transmitted between sensors and PIVSes in PIV are encrypted using shared pairwise keys. Therefore, the attackers cannot get the content of the messages by simply eavesdropping on the messages in the network.

B. Defense Against Active Attacks

Having every message include the MAC value of the message computed using a pairwise key between two nodes to achieve authenticity and integrity can prevent an adversary from spoofing or inserting false data. By including nonces in the messages prevents replay attacks. For sensors or PIVSes that keep dropping packets, the malicious nodes can be detected by an IDS [13] in the network.

Service disruption and Denial-of-Service (DoS) attacks are caused by malicious sensors or PIVSes. There is no way to prevent such nodes from launching attacks. Again, if there is an efficient and effective IDS [14, 13, 20] in the network, we can rely on the IDS to detect and identify the malicious sensors or PIVSes, and then design a PIVS revocation scheme or use the PIV protocol to exclude malicious nodes from the network.

Sybil attacks [8] are particularly harmful in sensor networks where a Sybil node illegitimately pretends to have multiple identities in the network. It is not possible for the attackers to launch Sybil attacks against DAPP since each

node will need to have a pairwise key with its communicating node to authenticate its identity. Since each node will have a pre-loaded function for establishing pairwise keys, no nodes can generate the pairwise keys and fake to be another node without knowing the function.

However, if any two compromised PIVSes in the network can collude together, then one compromised PIVS can issue authentication tickets for any compromised PIVS in the network. If one PIVS can find N_{auth} compromised PIVSes to collude with, it can pass the authentication with any sensor. DAPP can defend against the above-mentioned attack because PIVSes have stronger transmission power than sensors, so one sensor can also receive the PIVS Beacon Messages sent by some of a PIVS's neighbor PIVSes. Therefore, even though one sensor doesn't know about the entire topology of the network, it knows part of the network topology in its proximity. So, if one PIVS shows authentication tickets from all other PIVSes that the sensor doesn't know, then one must suspect the PIVS. Or, a sensor can even send a list of PIVSes that it wants to have authentication tickets from, and force the compromised PIVS to fail the authentication.

6. Implementation of DAPP and PIV

This section describes our implementation of the DAPP and the PIV protocol [15] on Mica2 Motes [3] and laptops. The Mica2 sensor nodes have 128 Kbytes in-system reprogrammable flash, and 4 Kbytes internal SRAM. We use Java to write the PIVS, nesC [9] to write the Boot code with assembly embedded inside, C embedded with assembly to write the mobile agent PIV Code (PIVC), and assembly to write the flash downloader.

Fig. 2 describes DAPP and the modified PIV we implemented between PIVSes and a sensor. The sensor starts the DAPP with one PIVS, and other reference PIVSes will authenticate the PIVS for the sensor. The sensor will determine the success or failure of the authentication of the PIVS based on the response of the authenticating PIVS and the authentication tickets from the reference PIVSes. If the PIVS passes the authentication, the sensor will then start the PIV protocol with the PIVS to verify its program.

6.1. Message Authenticity and Integrity

In DAPP, we use MAC to achieve message authenticity and integrity, and allow sensors to authenticate PIVSes. The security of the MAC depends on the length of the MAC. Conventional security protocols use MACs of 16 bytes. We choose to use HMAC-MD5 [12, 16] for generating MACs in our implementation, and to truncate the output of the MAC to use a 10 byte MAC.

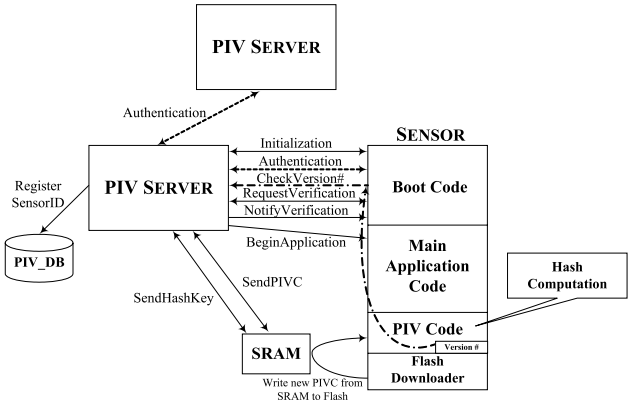

Figure 2. Overview of the implementation of DAPP and PIV between PIVSes and a sensor.

The most well-known and widely-used MAC algorithms are CBC-MAC and HMAC [12]. CBC-MAC (Cipher Block Chaining Message Authentication Code) utilizes block ciphers in CBC mode to create a message authentication code. HMAC is a keyed-hash message authentication code and is calculated using a cryptographic hash function in combination with a secret key. Crypto++ 5.2.1 benchmarks [2] are speed benchmarks for some of the most commonly-used cryptographic algorithms. In Crypto++ 5.2.1 benchmarks, HMAC-MD5 outperforms CBC-MAC-AES. Therefore, we decide to implement HMAC-MD5 for MAC computation.

6.2. PIVS, Boot Code, PIVC, and Flash Downloader Development

We now present some implementation details for each component of DAPP and PIV.

A. PIVS Development

The Mica2 Mote sensor used for our implementation runs under TinyOS. A laptop is used as a PIVS and a serial-line communication forms the primary channel for wired communication between a PIVS and the laptop. On the laptop, a simple Java application, SerialForwarder, provided a relay between the serial data over a TCP/IP socket connection. The PIVS sensor that connects to the laptop is for sending and receiving messages from the other sensors and PIVSes over the radio. The received messages will be relayed from the sensor to the laptop through the serial cable, and the send messages will also be relayed from the laptop to the sensor for broadcast or unicast.

The PIVS is written in Java, and we implemented HMAC-MD5 [12, 16] for generating MAC when PIVSes run DAPP. The shared pairwise key between two PIVSes

will be used for generating and verifying MACs. The key length is 16 bytes and the MAC length is 10 bytes. All of the sensor programs are stored on the laptop as files in binary formats, and are used for verification. PIVS calls a C program to compute HMAC_MD5 and hash over the stored sensor programs to verify the integrity of the programs on sensors. When sending PIVC to a sensor, the PIVS will read the PIVC from a PIVC binary file, and send it over the radio to the sensor.

The PIVS takes care of sensors' requests for authentication, requests for update of a mobile agent PIVC, requests for the hash key, requests for verification, and requests for checking the verified sensors in its PIV_DB. Upon updating the PIVC or sending the hash key to the sensor, a simple error checking is done by letting the sensors acknowledge to the PIVS the previous data it received. If the data is not the same, then there was data corruption during the previous transmission, and the PIVS will resend the previous data to the sensor.

PIVSes will randomly generate hash keys for each sensor during verification. If the previously-sent hash key bytes were corrupted, the PIVS will re-generate the corrupted bytes of the hash key, and retransmit the hash key bytes to the sensor. This is to prevent the sensors from reporting the wrong hash key bytes and try to gain time to generate the correct hash value for verification.

B. Boot Code Development

The Boot code is used for the sensor as a communication module between the sensor and the PIVS. The Boot code also allows the program pointer to jump back and forth between the Boot code, the PIVC, and the flash downloader.

We implemented the Boot code in nesC [9], along with inline assembly mixed in nesC code. After the communication between the sensor and the PIVS has built up, the Boot code will jump to the PIVC to get the version number of the PIVC, and then send it to the PIVS for checking if the version number is up-to-date. If not, then PIVS will send the new PIVC to the sensor, with 4 bytes of the PIVC per message. The bytes of the PIVC received by the sensor will first be stored in the SRAM. After one page, which is 128 words or 256 bytes, of the PIVC has been received, the Boot code will jump to the flash downloader and write the page from the SRAM to the Flash. After the entire PIVC has been written to the Flash, the Boot code will report its new PIVC version number to the PIVS again. If the version numbers match, then PIVS will send the hash key to the sensor for computing the hash value over its program. Finally, the Boot code will jump to the PIVC to start the hash computation over the entire Flash, and then send back the hash value for verification. Upon receiving the verification result from the PIVS, the Boot code will either activate the main application code on the sensor if the sensor passes the verification or otherwise lock the sensor to keep it from joining the network.

The message-loss problem is handled with timeouts and retransmissions. If a timer has expired and the sensor still has not received any message from the PIVS, then the sensor will resend the message to the PIVS.

C. PIVC Development

The PIV Code (PIVC), or the mobile agent, is written in C along with inline Assembly. The main function of the PIVC is to perform HMAC-MD5 on the sensor over the entire sensor Flash for verification.

The hash keys we use for HMAC-MD5 has a key length of 16 bytes, and the hash values are also 16 bytes. Our design for the PIVC has the flexibility to change the hash function, key length, and hash value length as needed. With the update of a new version of the PIVC, the changes can be made. When sending the hash key or the hash value over the network, the PIVS and the sensor will always specify the length of the data it is transmitting, and thus, the new version of the PIVC will work correctly. The version number of the PIVC is placed at the last part of the PIVC. This is to prevent the sensor from receiving only part of the PIVC but holds the up-to-date PIVC version number. The size of PIVC is about 10 Kbytes, and it takes about 5 minutes to transmit the entire PIVC from the PIVS to the sensor. When performing HMAC-MD5 over the entire 128 Kbytes of the Flash, we hash 64 bytes at a time, and XOR the 2048 hash values to get the final hash value for verification.

Since the PIVC needs to coexist with the Boot code in the Flash, and the variables will all be stored in the SRAM, we need to assign locations for the PIVC to be placed in the Flash, and the PIVC variables in the SRAM without overwriting the Boot code and its variables. We use `avr-objdump` command in TinyOS to create a dump file to analyze the Boot code memory information. The Flash location will be set inside the PIVC C program. We place the PIVC below the Boot code and the main application code in the Flash. Then, when compiling and linking the PIVC C program, we explicitly assign the SRAM location for the PIVC. The PIVC and the Boot code will also use SRAM space for variable value passings, such as the passing of the PIVC version number from the PIVC to the Boot code.

D. Flash Downloader

The flash downloader is written in assembly for writing one page of data to the sensor Flash with the one page of data in SRAM. The sensor Flash is divided into two constant sections, the Read-While-Write (RWW) section and the No Read-While-Write (NRWW) section [1]. Fig. 3

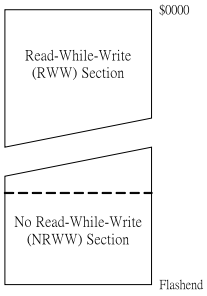

Figure 3. Read-While-Write section vs. No Read-While-Write section in the sensor Flash.

shows the limit between the two sections in the sensor Flash.

The main differences between the two sections are:

- While erasing or writing a page inside the RWW section, the NRWW section can be read.
- The CPU is halted during the entire operation of erasing or writing a page located inside the NRWW section.

Therefore, our flash downloader is placed in the NRWW section to allow read while writing a page of the PIVC to the RWW section in the Flash.

The program memory is updated in a page by page fashion. We use self-programming in the Flash to write the data to the Flash from SRAM. Before programming a page with the data stored in the SRAM, we first perform a page erase on the Flash. We then fill in the temporary page buffer one word at a time with the data in the SRAM. We finally perform a page write to write the data in the page buffer to the page in the Flash and complete the update.

7. Performance Evaluation

We first present the evaluation results using simulation. Then, we evaluate the computation and communication cost of DAPP, and the storage requirement for a sensor and PIVS to keep the pairwise keys.

7.1. Evaluation

We simulated DAPP with random networks consisting of 1000 sensors and 250 PIVSes in a 1000×1000 unit^2 area. We assume each sensor has a communication range of 150 units, and PIVS normally communicates within 200 units. Each sensor node initially is given 0.5 J of energy. Once a sensor node runs out of battery, it dies.

We compare the number of sensors alive when DAPP is used to when only one AS is available in the network

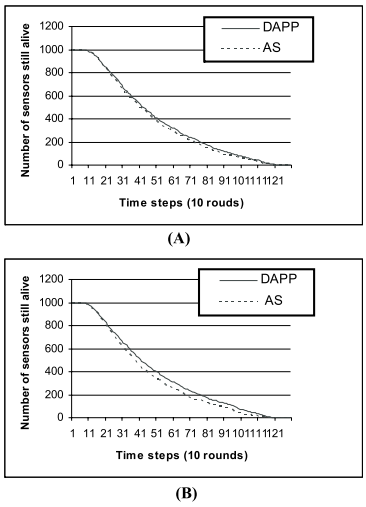

Figure 4. Number of sensors that remain alive using DAPP and AS for authentication, with (A) $ReAuth = 0.05$ **and, (B)** $ReAuth = 0.1$**.**

for authentication. In our simulation, the AS is placed at ($x = 1070$, $y = 1070$). When DAPP is used for authentication, PIVSes monitor and cooperate to authenticate one another, but the sensor only needs to communicate with the authenticating PIVS. We choose $N_{auth} = 5$, i.e., a PIVS needs to show 5 authentication tickets to one sensor to pass the authentication, for our simulation. When a dedicated AS is employed for authentication, since the AS is usually farther away from the network and placed in a safe area, the sensors might need to route through other sensors to reach the AS. After the authentication, the authenticated PIVS will act as a cluster head, and sensors will periodically transmit reports to the PIVS for data gathering.

Each sensor has a probability of $ReAuth$ that it needs to re-authenticate with its cluster head or PIVS. Our evaluation is done for different re-authentication probabilities $ReAuth$. Fig. 4 shows the number of sensors alive using DAPP and a dedicated AS for authentication, with $ReAuth = 0.05$ and 0.1.

Fig. 4 shows that using DAPP can have more sensors alive in the network than using a single AS in the network. Especially when the network is deployed in a highly hostile environment, and the sensors need to re-authenticate PIVSes more often, DAPP becomes more valuable.

For the same network, we also compare the messages exchanged and the average sensor energy consumption in the network when using DAPP and a dedicated AS. In the case of DAPP, since PIVS has dual radio interfaces, the communications between PIVSes will not interfere sensor communications. For each sensor to authenticate one PIVS, there are 2 messages exchanged between the sensor and the PIVS.

Since there are 1000 sensors in the network, assuming no transmission error has occurred, at least 2000 messages must be exchanged in the network for each sensor to authenticate one PIVS. And the average energy dissipated on each sensor for every sensor to authenticate one PIVS is 782 uJ. If a single AS is used in the network, then there will be an average of 21,873 messages exchanged in the network, and the average energy dissipated on each sensor for every sensor to authenticate one PIVS is 7,860 uJ. The increase of message traffic and energy consumption comes from sensors relaying authentication messages for other sensors. It is not difficult to see that the sensors deployed near the AS will die first due to relaying more messages for other sensors.

7.2. Computation Cost

The computation overhead of DAPP mostly comes from the setup of pairwise keys and the generation/verification of MAC value of the messages. We show below that both actions are very efficient.

A. Pairwise Keys Establishment

In DAPP, two nodes (either a sensor or PIVS) establish a pairwise key to authenticate their identities to each other. The Blundo scheme [5] is used to establish the pairwise keys between nodes. A node needs to compute k modular multiplications and k modular additions for a k-degree polynomial in order to generate a key. As stated in [21], if we choose $k = 100$, the size of the pairwise key to be 64 bits, and the size of node ID to be 16 bits, then the cost of computing a pairwise key is only about 1/10000 of that of creating a RSA signature, or the same order for computing an AES encryption.

B. MAC Generation and Verification

We use HMAC [12] for MAC generation and verification in DAPP. HMAC does not rely on encryption, but instead uses a cryptographic hash function in combination with a secret key. According to [17], HMAC consumes approximately 45.6 uJ if it runs on a Mote, which is less than the energy for transmitting 3 bytes.

7.3. Communication Cost

The communication overhead of DAPP is associated with a PIVS's authentication request to its neighbor PIVSes. For a PIVS to authenticate another PIVS, 2 messages need to be transmitted. A PIVS has to authenticate itself with N_{auth} neighbor PIVSes in order to pass the authentication. Therefore, there will be $2N_{auth}$ messages transmitted. However, DAPP will only be used before a sensor runs PIV and wants to authenticate a PIVS. Since a sensor will only run the PIV protocol infrequently, the communication overhead is not high. And since PIVSes have dual radio interfaces, the communications between PIVSes will not interfere sensor communications.

7.4. Storage Requirement

Each sensor and PIVS needs to store a k-degree polynomial for establishing pairwise keys, and will occupy $\frac{(k+1)\log q}{8}$ bytes. For a sensor to authenticate one PIVS, it needs to establish a pairwise key with it. The sensor will also need to decrypt the N_{auth} authentication tickets issued by the authenticating PIVS's reference PIVSes. Therefore, in DAPP, each sensor will need to establish at least $N_{auth}+1$ keys before it can authenticate a PIVS. Since each key is 128 bits (or 16 bytes) long, if we choose $N_{auth} = 5$, then a sensor will only need to store 6 keys, so a total of 96 bytes will suffice. Since a Mica2 Mote has 4 Kbytes SRAM, using 96 bytes to store keys requires only 2.3% of the sensor SRAM. PIVSes have more memory than sensors, therefore can store more keys than sensors.

Overall, DAPP is scalable and efficient in computation, communication, and storage in sensor networks.

8. Related Work

Weimerskirch and Thonet [19] present a security model for low-value transactions, especially focusing on authentication in ad-hoc networks. They use recommendation protocol to build trust relationships and extend it by requesting for references in ad-hoc networks. Each node maintains a local repository of trustworthy nodes in the network, and a path between any two nodes can be built by indirectly using the repositories of other nodes. The authors also introduced the idea of threshold cryptography [7] in which as long as the number of compromised nodes is below a threshold, the compromised nodes cannot harm the network operation.

Hubaux *et al.* [10] list the threats and possible solutions for basic mechanisms and security mechanisms in mobile ad-hoc networks. They develop a self-organizing public-key infrastructure. In their system, certificates are stored in local certificate repositories and are distributed by the users. Bauer and Lee [4] propose a distributed authentication scheme that is efficient and robust using the well-known concepts of "secrets sharing" cryptography and group "consensus". However, it still needs a centralized Processing Center (PC) which is responsible for coordinating the distribution of secret keys to each node in the network, thus lowering the value of its distributed nature.

Saxena *et al.* [18, 6] present a secure, efficient and a fully non-interactive admission control protocol and schemes that

allow a pair of nodes to compute a shared key without a centralized support in ad-hoc networks. Without the assistance of any centralized trusted authority, they also use secret sharing techniques based on bi-variate polynomials. As a comparison, our work focuses on authentication of servers, and [18, 6] feature admission control and pairwise key establishment.

9. Conclusions and Future Work

In this paper, we presented a distributed authentication protocol of PIVSes (DAPP) for sensors to authenticate PIVSes in sensor networks, and implemented DAPP and the PIV protocol [15] on Mica2 Motes. We also made modifications and improvements to the original PIV design. Our main contribution is the development of DAPP, to achieve the authentication of PIVSes without requiring a dedicated and trusted authentication server (AS), an important departure from [15]. DAPP maintains the distributed nature of sensor networks, and reduces the total traffic in the network and the energy consumption on each sensor, compared to the case of using a centralized trusted AS.

However, an intrusion detection system (IDS) for sensor networks is needed to initiate PIV on suspicious sensors after their initial admission, and is needed to detect malicious PIVSes in the network. Once the IDS identifies any malicious or malfunctioning sensors, it can collaborate with the PIV protocol to request for the sensor to re-verify with the PIVS. Once a malicious PIVS is detected in the network, other PIVSes can collaborate to revoke it. Unfortunately, there is not much work done on intrusion detection for sensor networks. This is a matter of our future inquiry.

References

[1] Atmel co., 8-bit AVR Microcontroller with 128 KBytes In-System Programmable Flash - ATmega128, ATmega128L http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf.

[2] Crypto++ 5.2.1 benchmarks, http://www.eskimo.com/~weidai/benchmarks.html.

[3] Crossbow co., MICA2 - Wireless Measurement System http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf.

[4] K. Bauer and H. Lee. A distributed authentication scheme for a wireless sensing system. In *Proceedings of the 2nd International Workshop on Networked Sensing Systems, INSS '05*, June 2005.

[5] C. Blundo, A. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung. Perfectly-secure key distribution for dynamic conferences. In *Advances in Cryptography - Crypto '92*, August 1992.

[6] C. Castelluccia, N. Saxena, and J. H. Yi. Self-configurable key pre-distribution in mobile ad-hoc networks,. In *The 4th International IFIP-TC6 Networking Conference*, May 2005.

[7] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Advances in Cryptography - Crypto '89*, August 1989.

[8] J. R. Douceur. The sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems, IPTPS '02*, March 2002.

[9] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation 2003*, June 2003.

[10] J.-P. Hubaux, L. Buttyàn, and S. Ĉapkun. The quest for security in mobile ad hoc networks. In *Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc '01*, October 2001.

[11] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. In *Proceedings of the 1st IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.

[12] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. IETF Network Working Group, RFC 2104, February 1997.

[13] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Network, MobiCom '00*, August 2000.

[14] A. Mishra, K. Nadkarni, and A. Patcha. Intrusion detection in wireless ad hoc networks. *IEEE Wireless Communications*, 11(1):48–60, February 2004.

[15] T. Park and K. G. Shin. Soft tamper-proofing via program integrity verification in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 4(3):297–309, May/June 2005.

[16] R. Rivest. *The MD5 Message-Digest Algorithm*. IETF Network Working Group, RFC 1321, April 1992.

[17] S. Sancak, E. Cayirci, V. Coskun, and A. Levi. Sensor wars: Detecting and defending against spam attacks in tactical ad-hoc sensor networks. In *2004 IEEE International Conference on Communications, ICC '04*, June 2004.

[18] N. Saxena, G. Tsudik, and J. H. Yi. Efficient node admission for short-lived mobile ad hoc networks. In *Proceedings of the 13th IEEE International Conference on Network Protocols, ICNP '05*, November 2005.

[19] A. Weimerskirch and G. Thonet. A distributed light-weight authentication model for ad-hoc networks. In *Proceedings of the 4th International Conference on Information Security and Cryptology, ICISC '01*, December 2001.

[20] Y. Zhang and W. Lee. Intrusion detection in wireless ad hoc networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, MobiCom'00*, August 2000.

[21] S. Zhu, S. Setia, S. Jajodia, and P. Ning. An interleaved hop-by-hop authentication scheme for filtering false data injection in sensor networks. In *IEEE Symposium on Security and Privacy*, May 2004.

[22] S. Zhu, S.Setia, and S. Jajodia. Leap: Efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS '03*, October 2003.