Distributed Authentication of Program Integrity Verification in Wireless Sensor Networks

KATHARINE CHANG and KANG G. SHIN
The University of Michigan

Security in wireless sensor networks has become important as they are being developed and deployed 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. Park and Shin [2005] proposed a soft tamper-proofing scheme that verifies the integrity of the program in each sensor device, called the *program integrity verification* (PIV), in which sensors authenticate PIV servers (PIVSs) using centralized and trusted third-party entities, such as authentication servers (ASs). This article presents a distributed authentication protocol of PIVSs (DAPP) without requiring the commonly used ASs. DAPP uses the Blundo scheme [Blundo et al. 1992] for sensors and PIVSs to establish pairwise keys and for PIVSs to authenticate one another. We also present a protocol for PIVSs to cooperatively detect and revoke malicious PIVSs in the network. We implement and evaluate both DAPP and PIV on Mica2 Motes and laptops, showing that DAPP reduces the sensors' communication traffic in the network by more than 90% and the energy consumption on each sensor by up to 85%, as compared to the case of using a centralized AS for authenticating PIVSs. We also analyze the security of DAPP under various attack models, demonstrating its capability in dealing with diverse types of attacks.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.4.6 [**Operating Systems**]: Security and Protection

General Terms: Algorithms, Design, Security

Based on "Distributed Authentication of Program Integrity Verification in Wireless Sensor Networks" by Katharine Chang, Kang G. Shin which appeared in *Proceedings of 2nd International Conference on Security and Privacy in Communication Networks (SecureComm)*, Baltimore, MD © 2006 IEEE.
The work reported in this paper was supported in part by the National Science Foundation under Grants CNS-0435023 and CNS-0523932, by the Office of Naval Research under Grant No. N00014-04-10726, and by a collaborative program between the US Air Force Office of Scientific Research (AFOSR) and Korean Ministry of Science and Technology (MoST).
Authors' address: The Real-Time Computing Laboratory, Computer Science & Engineering Division, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2121, U.S.A.; email: {katchang, kgshin}@eecs.umich.edu.

Additional Key Words and Phrases: Distributed authentication, program integrity verification, node revocation, wireless sensor networks

ACM Reference Format:
Chang, K. and Shin, K. G. 2008. Distributed authentication of program integrity verification in wireless sensor networks. ACM Trans. Inf. Syst. Secur. 11, 3, Article 14 (March 2008), 35 pages. DOI = 10.1145/1341731.1341735. http://doi.acm.org/10.1145/1341731.1341735.

1. INTRODUCTION

In recent years, security has become a primary concern to the communications between mobile nodes. Unlike wired networks, security in wireless networks is difficult to achieve due to the broadcast nature of internode communications. In sensor and ad hoc networks, it is even easier for attackers to circumvent the underlying intrusion detection system because a malicious user can join the network at one point, hide inside the network for a while, then mount attacks. An attacker who was detected and blocked from joining the network may just disconnect from the network, change his or her personal identification, and then rejoin from a completely different location in the same network.

Wireless sensor networks are becoming important for many emerging applications such as military surveillance, alerts on terrorists and burglars, and fire, earthquake, and volcano emergency systems. The security of sensor networks used for such applications is of utmost importance. However, the limitations on each sensor device's battery energy, memory, computation, and communication capacities make it very difficult to achieve security in sensor networks. Moreover, sensor networks are often composed of a large number of small low-cost 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.

To protect the sensors from physical attacks, including physical tampering and manipulation of the sensor programs, Park and Shin [2005] proposed a soft tamper-proofing scheme that verifies the integrity of the program in each sensor device, called the *program integrity verification* (PIV). Seshadri and colleagues also proposed a software-based memory attestation technique called SWATT [Seshadri et al. 2004] and Secure Code Update by Attestation (SCUBA) [Seshadri et al. 2006]. All of these deal with the problems of securing sensor devices and making sensor devices tamper resistant.

Another important issue is the authentication of the communication between the 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 also has to prevent itself from communicating with a malicious server that pretends to be a legitimate one.

For the purpose of securing the 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 or disabling the network. However, intrusion prevention can combat only 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 stored in the captured sensor node. Thus, the intrusion prevention measures that often require sharing secrets between nodes will not help defend against insider attacks.

The second line of defense is intrusion detection that can discover the insider attacks mounted by compromised nodes in the network. On detection of an intrusion, a countermeasure can be taken to minimize damages to the network. Given the new vulnerabilities that continue to be discovered, intrusion detection must be effective and efficient in identifying attacks and then successfully neutralizing them.

In the PIV protocol [Park and Shin 2005], the sensors rely on PIV servers (PIVSs) to verify the integrity of the sensor programs. The sensors authenticate PIVSs with centralized and trusted third-party entities, such as authentication servers (ASs), in the network. This article mainly addresses the first line of defense. Specifically, we focus on the authentication of PIVSs instead of the verification of sensors.

In this article, we propose a distributed authentication protocol of PIVSs (DAPP) for sensors to securely communicate with PIVSs without the authentication server (AS) infrastructure assumed in PIV [Park and Shin 2005]. DAPP is a solution to the problem of authenticating PIVSs in a fully distributed manner and acts as the first line of defense for the network by enabling each sensor node to prove the identity of a server before communicating with it. The sensors will then use the PIV protocol to have their program verified by the authenticated PIVSs and will be allowed to join the network only after passing the verification successfully. By *authentication*, we mean that one party ensures the valid identity of another party to communicate with. The proposed DAPP is to enable sensors to validate a PIVS before using it for their verification.

In addition to DAPP, we present a revocation mechanism for PIVSs to check with each other to detect and evict malicious PIVSs, if any. This mechanism is part of the second line of defense. Once a PIVS is determined to be malicious by a majority of its neighbor PIVSs, the revocation mechanism is used to evict it from the network and mitigate the possible damage it may cause to the network.

The remainder of this article is organized as follows. For completeness we first give an overview of the PIV protocol [Park and Shin 2005] in Section 2. We then present the overview of our design, including some of sensor device limitations and the motivation of this work as well as the system model we use, the attack model we consider, and the overview of DAPP in Section 3. Section 4 describes the details of DAPP, along with the PIVS revocation mechanism. Section 5 analyzes the security of DAPP and PIV and lists some security issues in PIV, which is followed by our DAPP and PIV implementations in Section 6. Section 7 evaluates the performance of DAPP. Finally, we discuss the related work in Section 8 and conclude the article by discussing future work in Section 9.

2. BACKGROUND: OVERVIEW OF PIV

The PIV protocol [Park and Shin 2005] 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. This protocol supports the tamper proofing of sensors and makes it difficult for attackers to modify the sensor programs without changing or adding the sensor hardware. It is triggered infrequently, only when a sensor tries to join the network or has left the network for a long time, and the verification of each program incurs a very small overhead. The PIV protocol will, therefore, not degrade normal sensor functions and services.

The network is composed of sensors and PIV servers (PIVSs). PIVSs verify the integrity of the sensors' programs and maintain a database of the digests of the original sensor programs. A randomized hash function (RHF) [Park and Shin 2005] is also employed in PIV. The RHF is used for computing hash on the program in the sensor device when the device needs to be verified. For each sensor verification, the PIVS creates a new RHF and sends it to the sensor in the PIV code (PIVC). The PIVS can verify the integrity of the program of each sensor device by comparing the hash value of the sensor program digests maintained in its local database with the hash value returned by the sensor after calculating it by executing the PIVC.

The security of sensors is accomplished by authenticating PIVSs before communicating with them, to protect sensors from malicious or compromised PIV servers. Sensors authenticate PIVSs with a conventional authentication server (AS), to ensure that the PIVSs are authentic and safe to communicate with and that the sensors can safely execute the codes received from the PIVSs.

The PIV protocol performs three tasks: (1) authentication of each PIVS via the centralized AS; (2) transmission and execution of the PIV code (PIVC); and (3) program verification by the PIVC and the PIVS. The sensor that wants to join the network will first ask the AS for authentication of a PIVS. If authentication succeeds, the sensor will then ask the authenticated PIVS for verification of its program. To verify a sensor's program, the PIVS will send a mobile agent, PIVC, containing a new RHF to the sensor and then use the RHF to compute a hash value from the digests of the sensor program stored 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, thus becoming unable to join the network. The PIV protocol offers three ways of actually locking a sensor: (1) The PIVS can ask the sensor's neighbor sensors not to replay packets from the sensor; (2) the key manager refreshes a new cluster key and excludes the sensor from the cluster; and (3) network services like routing may look up PIV_DB to ensure sensors are verified and thus genuine. Figure 1 depicts the interactions among the AS, the PIVS, and the sensor during PIV.

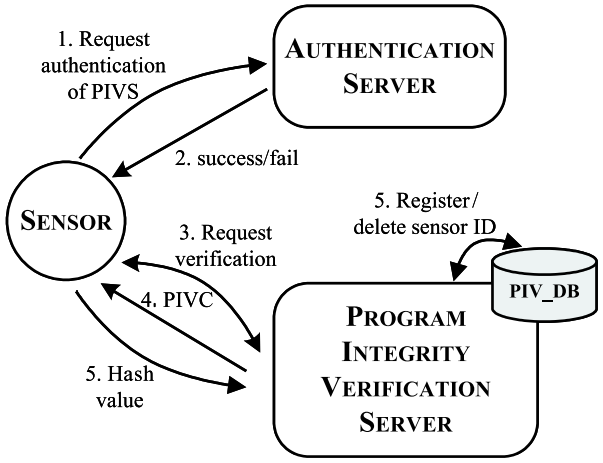

Fig. 1. Interactions among the AS, the PIVS, and the sensor during PIV.

The main objective of PIV is to counter most of the physical attacks, that is, to reprogram or manipulate a sensor without adding new sensor hardware, thus making it extremely difficult for the attackers to modify the sensor program without being caught. This protocol guarantees the integrity of the sensor program by requiring the sensor node to verify the integrity of its program before joining the network or after it has been disconnected from the network for a long period of time. However, PIV still cannot combat the attack of adding more memory to the sensor nodes.

3. DESIGN OVERVIEW

In this section, we first describe the architecture and limitations of a typical sensor network. We then state the motivation of our work, the system model we use, and the attack model we consider, and we give the overview of our DAPP design.

3.1 Sensor Network Architecture and Limitations

Sensor networks are often used for monitoring environments and information collection and aggregation. Most sensor networks have a base station that acts as their gateway to an external network. The base station is usually a more powerful node with larger computation, communication, memory, and energy capacities.

A sensor network may typically consist of from hundreds to several thousands of sensor nodes. However, sensor nodes are limited in their computation, communication, memory, and energy capacities due to their low cost and size requirements. Because of the large number of nodes and limited resources, and also due to the fact that sensor nodes are often deployed in hostile and unattended environments, they are susceptible to physical capture and compromise.

Sensor networks are also limited in other ways. Due to the limited memory, computation power, and energy capacity of each sensor device, the use of public-key algorithms such as the Diffie-Hellman (DH) algorithm [Diffie and Hellman 1976] is usually not practical in sensor networks. Public-key algorithms often require considerable memory, complex computation and processing, and large key length, which have limitations of their own and will quickly deplete the batteries on sensor devices.

Our scheme was implemented on Mica2 Motes [Crossbow]. Mica2 Motes feature an 8-bit 4MHz Atmel ATmega 128L microcontroller with 128K bytes in-system reprogrammable flash memory, 4K bytes internal SRAM, 4K bytes internal EEPROM, and 512K bytes external additional data flash memory. The microcontroller is based on an advanced RISC architecture. Mica2 Motes are powered by two AA batteries and communicates using a multichannel radio. The ISM band 868/916MHz radio transmitter communicates at a peak rate of approximately 40 Kbps within a range of up to 500 feet in an outdoor environment.

3.2 Motivation

Our main goal is to eliminate the requirement of the centralized authentication server (AS) in the PIV infrastructure to make PIV a fully distributed protocol. As the AS is needed for sensors to authenticate PIVSs, it may easily become a bottleneck for reliability, security, and communication. Also, requiring a centralized service such as AS 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 exhaust their batteries before others. Therefore, having a centralized AS will not scale well to a large sensor network.

The communication traffic to/from a single AS can, of course, be reduced if the AS is replicated in the network. However, because ASs act as trusted third parties, they are presumed to be trusted and secure. Therefore, ASs will need secure computing platforms to protect the servers from attacks. Also, as ASs require more memory, more energy, and stronger computation power than sensor devices, deploying more ASs in the network will increase the cost. Another problem with having multiple ASs is that of maintaining consistency among them. How to allow every AS in the network to maintain the same authentication information about PIVSs and be consistent with authenticating PIVSs in the network is a difficult issue to deal with. For these reasons, we would like to remove the need for ASs from the PIV infrastructure and distribute the authentication function to PIVSs themselves.

3.3 System Model

To generalize our design and analysis, we define the system model as follows:

—Sensors and PIVSs 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.

—There is a maximum of n sensor nodes and s PIVSs in the network, each of which has a unique node ID.
—PIVSs are equipped with more energy, larger memory, and more computation and transmission power than sensor devices.
—PIVSs maintain all of the sensor programs in their memory before deployment. The sensor programs stored at the PIVSs are used to verify the integrity of the sensors' programs.
—Each PIVS has dual radio interfaces so that the radios for its communication with sensors and other PIVSs will not interfere with each other. Note that the use of multiple radios and channels at each node is becoming commonplace.
—For most of sensor network applications, sensors are required to be time synchronized. Hence, PIVSs are assumed to be loosely time synchronized.
—Because PIVSs have a longer transmission range than sensors, each PIVS is assumed to have at least t neighbor PIVSs after deployment.

3.4 Attack Model

Because the sensor nodes are small and resource limited and are usually deployed in public or hostile locations, they are vulnerable to physical capture and compromise by attackers. PIVSs are much more powerful servers in the network and are more capable in defending the various attacks in the network. The goal of the PIV protocol [Park and Shin 2005] is to defend the sensors from physical compromise attacks. Our goal in this article is to allow the sensors to authenticate PIVSs in a fully distributed manner. Below we list the attack model we consider for PIVSs. We categorize the attacks into passive and active attacks.

—*Passive attacks*: *Eavesdropping.* This is an obvious threat because wireless communication is broadcast based. We assume the attackers can eavesdrop on all traffic in the network and data encryption is an effective countermeasure.
—*Active attacks*:
  —*Replay, spoof, drop, or insert false data.* An attacker might add a node to the network that simply attempts to interrupt message transmission. A malicious node could trick the system by simply dropping messages it is supposed to forward or by inserting false data into the network. A malicious node might also attempt to impersonate a legitimate sensor by replaying the messages it received or by spoofing the messages it is supposed to send.
  —*Denial-of-service (DoS) attack.* An attacker can maliciously insert messages or "flood" the network to cause resource consumption, such as battery depletion, thus making legitimate nodes unable to use service or provide service.
  —*Sybil attack.* A particularly harmful attack against sensor networks is known as the Sybil attack [Douceur 2002], where a Sybil node illegitimately fakes multiple identities in the network.

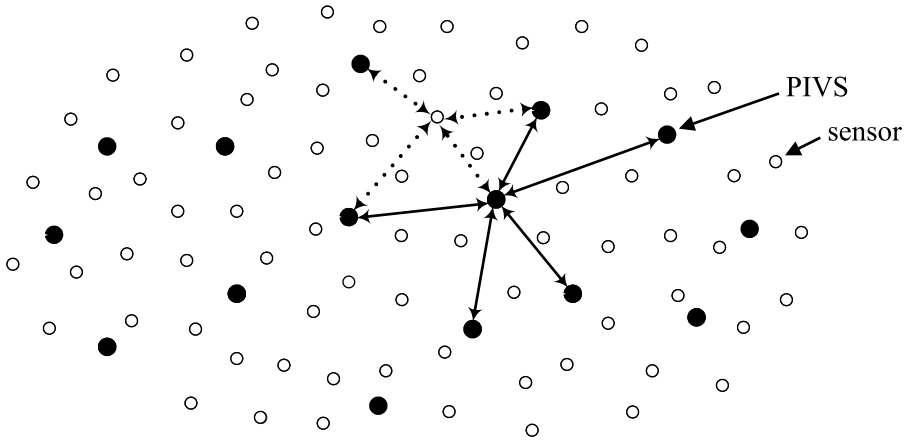

Fig. 2. The design overview of DAPP. PIVSs are represented by large circles and interact with one another for mutual authentication. The solid lines represent the interactions between the PIVSs. Small circles represent sensors in the network. The dotted lines represent the interactions between the PIVSs and the sensor.

—*Impersonation attack.* An attacker has the ability to impersonate or masquerade as a legitimate node. Other nodes will thus send confidential information to the attacker rather than to the real recipients. This is typically the hardest attack to mount and defend against.

3.5 Overview of DAPP

We exploit the PIV protocol [Park and Shin 2005] summarized in Section 2 to protect the integrity of sensor programs. To remove the need for a centralized AS from the PIV infrastructure, we use PIVSs to perform the AS functions and authenticate one another to achieve the distributed authentication of PIVSs. Interactions between PIVSs and sensors are depicted in Figure 2, where the large/small circles represent PIVSs/sensors in the network. PIVSs interact with one another to be authenticated without the need for a centralized AS in the network. The solid lines represent the interactions between the PIVSs, and the dotted lines represent the interactions between the PIVSs and the sensor.

PIVSs need to share some secrets with one another to authenticate one another. In our proposed distributed authentication protocol of PIVSs (DAPP), these secrets are pairwise keys shared between PIVSs that are established by using the Blundo scheme [Blundo et al. 1992]. Before deployment, PIVSs and sensors are loaded with different functions to establish pairwise keys. After deployment, PIVSs and sensors can use the loaded functions to establish pairwise keys with any node in the network.

DAPP has two more objectives in addition to the distribution of the AS function to PIVSs: Reduce (1) the total number of sensor communication messages exchanged in the network and (2) the energy consumed by each sensor by using DAPP for authentication.

To remove malicious PIVSs in the network, we also propose a PIVS revocation mechanism. The PIVS revocation mechanism will allow normal PIVSs to revoke a malicious PIVS after they detect its malicious behavior. Neighbor PIVSs will monitor each other's behavior, and once more than a certain number of its neighbor PIVSs have determined a PIVS to behave maliciously, the PIVS will be evicted from the network.

4. DAPP DETAILS

In the original PIV design [Park and Shin 2005], 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 thus safe to execute. Here, we propose a distributed protocol, DAPP, for sensors to authenticate PIVSs without using such a centralized AS.

DAPP is a protocol used in a sensor network that consists of a maximum of n sensor nodes and s PIVSs, in which all sensors and PIVSs in the network have unique node IDs. All PIVSs store all the sensor programs in the network for sensor verification. Listed here are the notations used in the rest of the article.

—$A, B, \ldots$ are principals, such as PIVSs or sensor nodes.
—N_A is a nonce generated by A, which is a randomly generated number that is unpredictable and 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.
—T_A is the time stamp sent by A.
—f is a symmetric bivariate k-degree polynomial for establishing pairwise keys in the network.
—F is a one-way function for generating revocation keys for PIVSs.

We apply the Blundo scheme [Blundo et al. 1992] for setting up PIVSs' and sensors' pairwise keys. The Blundo scheme was originally proposed to allow any group of m parties to compute a shared secrete key. Here we use this scheme to establish pairwise keys between two nodes in the network. In the Blundo scheme, a key server randomly generates a symmetric bivariate k-degree polynomial $f(x, y)$ that is a secret known only to the key server. Any two nodes in the network can generate a pairwise key by substituting x and y by their node IDs. Although we can also use the pairwise key establishment scheme in LEAP [Zhu et al. 2003], we describe below the protocol using the Blundo scheme for pairwise key generation.

The elements of the system model described earlier are necessary to understand the protocol described below. Recall that the PIVSs are computationally more powerful than sensors, and their memory, energy, and transmission capacities are also greater than sensors'. Moreover, after the deployment of

PIVSs, each PIVS has at least t neighbor PIVSs, where t depends on PIVSs' transmission range.

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 after deployment and use the same phase to join the network.

4.1.1 *Pre-Deployment Initialization Phase.* Before the deployment of sensors and PIVSs, all the nodes are secured. In this phase, we establish the identity of each PIVS/sensor and its security basis. The network consists of a maximum of n sensor nodes and s PIVSs. Each sensor and PIVS is assigned a node ID. Not all sensor nodes and PIVSs need to be deployed at once; some can later join the network. All s PIVSs have the n sensors' programs stored in their memory.

As stated above, the Blundo scheme [Blundo et al. 1992] is used to set up pairwise keys. 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 a pairwise key function, $f(A, y)$, and loads the $k + 1$ coefficients as a function of y to node A. The pairwise key function is used for nodes later to establish pairwise keys with other nodes in the network. Note that since the Blundo scheme is proven to be unconditionally secure and k-collusion resistant, compromising only one node A and discovering $f(A, y)$ does not enable the attacker to recover $f(x, y)$. The attacker needs to compromise more than k nodes to recover $f(x, y)$.

For each PIVS A, the key server randomly generates a base revocation key $K_{A,x}$ for A and then generates a sequence of v revocation keys from $K_{A,x}$ using a one-way function F. The remaining revocation keys are generated by applying F successively, that is, $K_{A,j} = F(K_{A,j+1})$. $K_{A,0}$ is called PIVS A's revocation verification key. Because revocation keys are generated by a one-way function, they are forward computable, but not backward computable, that is, one can compute $K_{A,0}, \ldots, K_{A,j}$ given $K_{A,j+1}$ but cannot compute $K_{A,j+1}$ from $K_{A,0}, \ldots, K_{A,j}$. The s PIVSs now all have a chain of $v + 1$ revocation keys and will store the revocation verification keys of the other $s - 1$ PIVSs. These keys are used for authenticating the PIVS revocation messages (to be described later in this section).

We require that no two PIVSs have the same revocation key in their key chains. This requirement can be met by having the key server first generate a long sequence of revocation keys by applying the one-way function F successively and then assign disjoint chains of keys to each PIVS.

4.1.2 *Post-Deployment PIVS Discovery Phase.* The sensors and PIVSs are then randomly deployed in the field. After deployment, they can discover neighbor PIVSs within their communication range in this phase.

In the postdeployment PIVS discovery phase, each PIVS will periodically broadcast a PIVS beacon message containing its ID. The neighbor PIVSs that

receive this message can establish shared pairwise keys with the PIVS using the pairwise key function. When PIVS B wants to establish a pairwise key with PIVS A, it computes $f(B, A)$ by substituting y with A in $f(B, y)$, the pairwise key function preloaded by the key server in the predeployment initialization phase. Likewise, A can compute $f(A, B)$. Because $f(x, y)$ is a symmetric function, $f(A, B) = f(B, A)$, so A and B can establish a pairwise key $K_{AB} = f(A, B)$ between them. The two PIVSs can now verify one another's authenticity using the shared pairwise key K_{AB}. Thus, the complete protocol of the PIVS postdeployment discovery phase is:

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

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 include nonces in the protocol to prevent replay attacks. The nonces are randomly chosen and are different each time to make it difficult, if not impossible, for the attackers to replay the messages. We further optimize the protocol by including the nonces implicitly in the MAC computation. This way, the PIVSs do not transmit the nonces again and save the communication traffic while meeting the verification goal.

After receiving a message, the receiver PIVS recomputes the MAC for the message and compares it with the MAC it received. If the two 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 PIVSs verify each other's authenticity with their shared pairwise key, they add each other's ID to their PIVS reference list.

Moreover, the PIVS beacon messages from a PIVS allow the newly deployed sensors to receive the information about the PIVSs within its communication range. If a newly deployed sensor does not receive any PIVS beacon message for a certain period of time, it will broadcast a PIVS lookup message to search for PIVSs 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 PIVSs before starting the PIV and having their programs verified. In this phase, the sensors will authenticate PIVSs, which will rely on their neighbor PIVSs to authenticate themselves for the sensors.

Based on the strength of PIVS beacon message signals received from all the PIVSs, a sensor node can choose the one that is closest to it to verify its

program. To authenticate, the sensors will first choose PIVSs within one-hop distance. If no such PIVSs exist, then sensors can communicate to PIVSs that are multiple hops away via secure routing [Karlof and Wagner 2003]. The sensor will first authenticate the PIVS; if the PIVS is trustworthy, the sensor will start the PIV protocol 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 new PIVS.

For example, 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 authentication message to A, which includes a randomly generated nonce N_E and the MAC value of N_E computed using K_{AE}. On receiving the message, PIVS A will generate the shared key $K_{AE} = f(A, E)$ and verify the PIVS authentication message sent by E. If the message is authentic, then PIVS A will send PIVS reference messages including sensor E's ID, nonce N_E, and the MAC value computed using the shared pairwise keys to the PIVSs on its PIVS reference list. The PIVSs that receive the PIVS reference messages will check the authenticity of the message it received. To check the authenticity of a PIVS reference messages, a PIVS simply recomputes the MAC value using the shared pairwise key, and compares the MAC value included in the message. If the two values match, then the message is actually sent from PIVS A because it has the correct pairwise key. However, if the two values do not match, then the message must have come from a malicious PIVS faking to be A; hence, 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 to pass the authentication. Note that the reason for sensor E to include a randomly generated nonce in the PIVS authentication message is to prevent external attackers from recording the authentication tickets in the network and then reusing them.

The complete protocol of the PIVS authentication phase is summarized as follows.

$$
\begin{aligned}
E \rightarrow A &: \text{PIVS authentication message}(N_E, \text{MAC}(K_{AE}, N_E)) \\
A \rightarrow B &: \text{PIVS reference message}(E, N_E, \text{MAC}(K_{AB}, E|N_E)) \\
B \rightarrow A &: E, \text{Authentication ticket}(B, \text{MAC}(K_{BE}, N_E)), \\
&\quad \text{MAC}(K_{AB}, E|\text{Authentication ticket}(B, \text{MAC}(K_{BE}, N_E))) \\
&\quad \ldots \\
A \rightarrow E &: \text{Authentication ticket}(B, \text{MAC}(K_{BE}, N_E))|\text{Authentication ticket} \\
&\quad (C, \text{MAC}(K_{CE}, N_E))|\ldots, \text{MAC}(K_{AE}, N_E + 1)
\end{aligned}
$$

Using this same example, for PIVS B to authenticate PIVS A, B needs to hold a pairwise key with A. Therefore, if B has received a PIVS reference 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 with the MAC value of B's message computed by A using their shared pairwise key K_{AB}, then A is sure that B is also trustworthy and will use the authentication ticket issued by B. After A receives N_{auth} authentication tickets issued by its reference PIVSs, it will forward the authentication tickets along with the MAC value of $N_E + 1$ computed using the pairwise key K_{AE} to sensor E for authentication.

When sensor E receives the response from PIVS A, it will check the authenticity of A. E will first check if A has replied with the correct MAC value of $N_E + 1$. 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 reference PIVSs, the authentication result must mask the effects of d or fewer malicious PIVSs. If we consider Byzantine failures in the network, then $t \geq N_{auth} \geq 3d+1$, where t is the minimum number of neighbor PIVSs that each PIVS has after deployment. If we do not consider Byzantine failures, then $t \geq N_{auth} \geq 2d + 1$. Thus, the design parameter N_{auth} needs to be determined by the PIVSs' failure model.

If we do not consider Byzantine failures in the network, then if A provides a correct MAC value, 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 trustworthy. Otherwise, E will try to verify its program with another PIVS, if any PIVS is available, and restart the authentication procedure with that PIVS. Figure 3 shows the interactions among PIVS A, PIVS A's reference PIVS B, and sensor E during DAPP authentication phase. Figure 4 is the pseudocode for sensor E performing DAPP to authenticate PIVS A. The pseudocode is to be executed on sensor E, PIVS A, and PIVS B, respectively. Sensor E performs the distributed protocol to authenticate PIVS A, while PIVS B authenticates A for E and is one of the PIVSs on PIVS A's PIVS reference list.

4.3 PIVS Revocation

As PIVSs may be compromised and then become malicious after their deployment, we need a way of detecting and evicting a maliciously or abnormally behaving PIVS without using a centralized entity, that is, a distributed PIVS revocation scheme. To meet this need, we design a PIVS revocation scheme to work harmoniously with DAPP to better authenticate PIVSs for sensors.

All PIVSs monitor their neighbor PIVSs and are able to issue PIVS revocation messages to other PIVSs when they detect abnormal/malicious behaviors from their neighbor PIVSs. Because each PIVS has at least t neighbor PIVSs, there are at least t neighbors who can detect the malicious behavior of a malicious PIVS and then cooperate to evict it. When a PIVS's abnormal or malicious behavior is detected, its neighbor PIVSs can send others PIVS

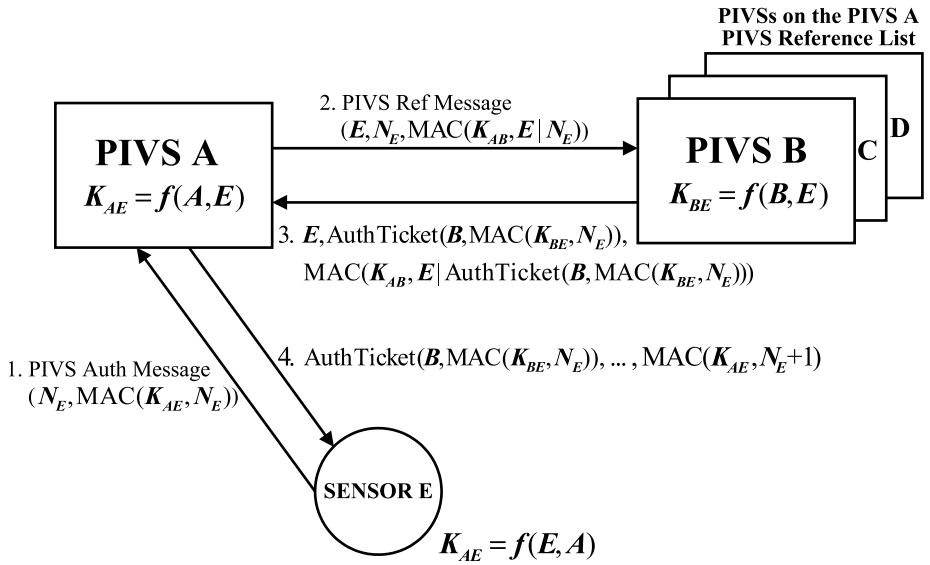

Fig. 3. Interactions among PIVS A, PIVS A's reference PIVS B, and sensor E under DAPP. (1) Sensor E sent PIVS authentication message to PIVS A, which (2) then sent PIVS reference messages to PIVSs on its PIVS reference list. (3) PIVS A's reference PIVSs then sent authentication tickets back to A, which (4) collected N_{auth} authentication tickets from its reference PIVSs and forwarded the tickets to E for authentication.

revocation messages on that PIVS. Examples of a PIVS's malicious behavior include authenticating other PIVSs when it is not expected to or continuing to fail authentications. How to detect PIVSs' abnormal output behaviors is the only concern to us in this article. As long as a PIVS behaves normally in interacting with the outside world, we treat it as normal.

Compromised PIVS are detected based on mutual monitoring among neighbor PIVSs in the network. The detection rules depend on the normal behavior of the PIVSs and use anomaly detection. Behavior-based anomaly detection compares the traffic being generated by a PIVS with its normal traffic-generation profile. Note that a PIVS's normal profile can be derived from its traffic-generation history and/or the underlying PIV protocol. Any PIVS that deviates from the normal behavior profile will be flagged as a potentially compromised PIVS. When a PIVS detects one of its neighbors, say N, to behave abnormally more than a prespecified number of times within a time interval of interest, then it will use μTESLA [Perrig et al. 2001] to broadcast a PIVS revocation message about N to other neighbor PIVSs. A receiver PIVS then uses μTESLA to authenticate the PIVS revocation messages it received. Note that μTESLA uses digital signature for packet authentication and uses only symmetric key encryption mechanisms. In μTESLA, time is divided into intervals, and key K_i is associated with the i-th time interval. Messages sent in interval i use K_i in MAC for authentication, and K_i will be disclosed after a delay δ for message authentication.

```
Sensor E (PIVS_AUTH msg)
{
    Choose PIVS A to authenticate;

    Randomly generate N_E;
    K_AE = f(E, A);
    PIVS_Authentication_Message = {N_E, MAC(K_AE, N_E)};
    Send PIVS A [PIVS_Authentication_Message];

    If msg.get_source == PIVS A,
        Receive [AuthTicketList, MAC(K, N +1)];

        If MAC(K, N +1) == MAC(K_AE, N_E +1),
            Check AuthTicketList;

            If CorrectAuthTicket >= IncorrectAuthTicket,
                PIVS A passes authentication;
                Start PIV with PIVS A;
            Else,
                PIVS A fails authentication;
                Choose another PIVS to authenticate;
        Else,
            PIVS A fails authentication;
            Choose another PIVS to authenticate;
}


Reference PIVS B (PIVS_AUTH msg)
{
    PIVS_Reference_List REF_LIST_B;

    If (PIVS A = msg.get_source) == PIVS on REF_LIST_B and
      msg == PIVS_Reference_Message,
        Receive [E, N, MAC(K, E | N)];
        K_AB = f(B, A);

        If MAC(K, E | N) == MAC(K_AB, E | N),
            K_BE = f(B, E);
            Send PIVS A [E, AuthTicket(B, MAC(K_BE, N)),
            MAC(K_AB, E | AuthTicket(B, MAC(K_BE, N)))];
}
```

```
PIVS A (PIVS_AUTH msg)
{
    PIVS_Reference_List REF_LIST_A;
    AuthTicket AuthTicketList[N_auth];

    N_ticket = 0;

    If msg == PIVS_Authentication_Message,
        sensor E = msg.get_source;
        Receive [N_e, MAC(K, N_e)];
        K_AE = f(A, E);

        If MAC(K, N_e) == MAC(K_AE, N_e),
            N_ticket = 0;
            For each PIVS B on REF_LIST_A
                K_AB = f(A, B);
                PIVS_Reference_Message = {E, N_e, MAC(K_AB, E | N_e);
                Send PIVS B [PIVS_Reference_Message];

    If (PIVS B = msg.get_source) == PIVS on REF_LIST_A and
        msg == [E, AuthTicket(B, MAC(K_BE, N)),
         MAC(K, E | AuthTicket(B, MAC(K_BE, N)))] and N_ticket <= N_auth,

        If MAC(K, E | AuthTicket(B, MAC(K_BE, N))) ==
                        MAC(K_AB, E | AuthTicket(B, MAC(K_BE, N)))],
            AuthTicketList[N_ticket] = AuthTicket(B, MAC(K_BE, N));
            N_ticket++;

        If N_ticket == N_auth,
            Send sensor E [ AuthTicketList, MAC(K_AE, N_e + 1)];
}
```

Fig. 4. Pseudocode to be executed on sensor E, PIVS A, and PIVS B. Sensor E performs DAPP to authenticate PIVS A, while PIVS B authenticates A for E and is one of the PIVSs on PIVS A's PIVS reference list.

To use μTESLA, all PIVSs in the network are loosely time-synchronized [Ganeriwal et al. 2005; Sun et al. 2006], and the time is divided into intervals. When PIVS A detects malicious behaviors from one of its neighbor PIVSs, it will broadcast a PIVS revocation message to the network and include the MAC of the message computed using the preloaded revocation keys in its key chain described earlier in this section.

If PIVS A detects the malicious behavior by its neighbor PIVS B, it will broadcast to its other neighbor PIVSs a PIVS revocation message about B that includes a timestamp T_A, the ID of the malicious PIVS B, the position of the revocation key in the key chain that is used to compute the MAC value, and the MAC of the message computed using the revocation key. A PIVS revocation message broadcast by A is in the form of:

$$A \rightarrow * : T_A, B, j, \text{MAC}(K_{A,j}, T_A|B|j)$$

The MAC value of a PIVS revocation message is generated by a PIVS using the revocation key in its key chain. After δ time intervals, the used revocation key will be disclosed. One PIVS can verify the authenticity of the disclosed revocation key by applying the one-way hash function F to the key a number of times and comparing the value with the origin PIVS's revocation verification key. For example, suppose the PIVS revocation message is from PIVS A, and A is using its fourth revocation key $K_{A,4}$ to generate the MAC value of the PIVS revocation message. Then, once the revocation key is disclosed, one

can apply the one-way function F four times and compare the value with A's revocation verification key $K_{A,0}$. If $F^4(K_{A,4}) = K_{A,0}$, then one can verify that the revocation key is really sent by A. One can then check the MAC value of the PIVS revocation message using the disclosed revocation key to see if the PIVS revocation message is authentic.

If A receives more than N_{revoke} PIVS revocation messages against B within a time interval of interest, A will suspect that B is compromised. A will thus stop authenticating B or ask B to authenticate itself again. Any malicious PIVS that cannot get authentication tickets from a majority of its neighbor PIVSs will not pass the authentication, and no sensor will trust such a PIVS. The malicious PIVS's neighbor PIVSs will also stop communicating with it. Therefore, the PIVS will be "evicted" from the network and will not be able to access any service in the network. N_{revoke} must be smaller than t, the minimum number of neighbor PIVSs one PIVS has, and must be large enough for no PIVS will be revoked by colluded neighbors.

5. SECURITY ANALYSIS

We first discuss the network survivability in the event of sensor and PIVS compromises and then analyze the security of DAPP and PIV against various attacks. We finally identify some possible attacks on the PIV protocol that we found along with their countermeasures.

5.1 Network Survivability

After compromising a sensor or a PIVS, the attacker can discover the node's keying materials, such as the preloaded pairwise key functions. If a PIVS is found to have been compromised, it can be evicted by using the PIVS revocation scheme with the cooperation of its neighbor PIVSs. On the other hand, if the compromise of a sensor is detected/suspected, the sensor will be required to reverify 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 PIVSs go undetected, we need to analyze the survivability of the network or the ability of the network to maintain an acceptable level of performance under node compromises. For this, we will consider the general attacks an adversary can mount after it compromises a node.

Because each node in the network is preloaded only with the pairwise key function for it to establish pairwise keys with other nodes in the network, revealing such a function will only allow the attacker to fake to be that node. We employ the Blundo scheme [Blundo et al. 1992] for the pairwise key function, which is proven to be unconditionally secure and k-collusion resistant. That is, when no more than k nodes are compromised, the attacker 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 pairwise key function or the symmetric bivariate k-degree polynomial will be revealed, and the attacker 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 by Zhu and colleagues [2004], for the current generation of sensor nodes, k can be around 200.

Each PIVS has a revocation key chain for it to authentically broadcast PIVS revocation messages. Once a PIVS is compromised, its revocation key chain will be revealed to the attacker, and the attacker can fake PIVS revocation messages to revoke benign PIVSs. However, because each PIVS needs to receive N_{revoke} PIVS revocation messages before revoking a PIVS, an attacker will need to compromise many PIVSs before it can revoke a PIVS. Or, if an attacker simply uses one compromised PIVS to continue broadcast PIVS revocation messages against other PIVSs, its abnormal behavior will be detected and then it will be revoked by other PIVSs. We can further modify the PIVS revocation scheme to limit a PIVS to issue only a certain number of PIVS revocation messages against other PIVSs, thus making it harder for an attacker to use compromised PIVSs to revoke benign PIVSs.

5.2 Defense Against Various Attacks in Sensor Networks

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

5.2.1 *Defense Against Passive Attacks.* In DAPP, each message is sent in plain text, along with its MAC value. Even though an attacker can eavesdrop on the messages, the only content that is revealed is the nonces that the sensors and PIVSs exchange. Therefore, the attacker will not gain any insight into the contents of messages by eavesdropping on the network.

After a sensor authenticates a PIVS, pairwise keys are used to encrypt all the messages transmitted between sensors and PIVSs in the PIV protocol. Therefore, the attacker cannot get the contents of the messages by simply eavesdropping on the messages in the network.

5.2.2 *Defense Against Active Attacks.* To prevent an attacker from spoofing or inserting false data, we equip every message with its MAC value computed using a pairwise key between two nodes to achieve authenticity and integrity. Replay attacks are prevented by including nonces in the messages. For PIVSs that keep dropping messages or data packets, the PIVSs compromised by the attacker can be detected and then revoked by its neighbor PIVSs.

Service disruption and denial-of-service (DoS) attacks are caused by malicious PIVSs. There is no way to prevent such nodes from launching attacks, but these nodes can be detected and then excluded from the network. Because we let PIVSs monitor their neighbor PIVSs, once a PIVS (with cooperation from its neighbor PIVSs) identifies malicious PIVSs, it can use the PIVS revocation scheme to evict the compromised PIVSs from the network.

Sybil attacks [Douceur 2002] are particularly harmful in sensor networks where a Sybil node illegitimately fakes having multiple identities in the network, but DAPP intrinsically withstands such attacks. It is not possible for the attacker to launch Sybil attacks against DAPP because each node will need to have a pairwise key with its communicating node to authenticate its identity. Because each node will have a preloaded pairwise key function for establishing

pairwise keys, no node can generate the pairwise keys and pretend to be another node without knowing the function.

However, if any two compromised PIVSs in the network collude together, then one compromised PIVS can issue authentication tickets for another compromised PIVS in the network. If one PIVS can find N_{auth} compromised PIVSs to collude with, then it can pass the authentication with sensors in the network. DAPP can defend against the above-mentioned attack because PIVSs have stronger transmission power than sensors; hence, one sensor can also receive the PIVS beacon messages sent by some of a PIVS's neighbor PIVSs. Therefore, even though one sensor does not know about the entire topology of the network, it knows part of the network topology in its proximity. Thus, if one PIVS shows authentication tickets from all other PIVSs that the sensor does not know, then one must suspect the PIVS. Or a sensor can even send a list of PIVSs that it wants to have authentication tickets from and force the compromised PIVS to fail the authentication. Note that the list of PIVSs that a sensor normally hears should contain fewer than N_{auth} PIVSs and should not use much of the sensor memory.

Last, PIV is designed to combat physical attacks to sensors in the network. However, if the sensors are captured and compromised after they passed PIV, then the network security may be breached. Therefore, to defend sensors against physical attacks after passing the PIV, we make them reverify their programs with PIVSs periodically. Likewise, in DAPP, for a PIVS to pass by a sensor's authentication, it needs to present N_{auth} authentication tickets to the sensor. If a PIVS is detected to have been compromised and then revoked by its neighbor PIVSs, then they will not issue authentication tickets for the compromised PIVS; therefore, it will not be able to pass authentication.

5.3 Security Issues and Possible Attacks to PIV

Listed below are some possible attacks on the PIV protocol that we found and the possible countermeasures against them.

5.3.1 *Flash Downloader Attack.* When a PIVS sends the mobile agent, PIVC, to a sensor, the received code will first be stored in the sensor's SRAM. We thus need to create a flash downloader to copy the received code from SRAM to the sensor flash memory. However, the attacker may try to use the flash downloader to write malicious code from SRAM to the flash.

We handle this attack by verifying the entire sensor flash memory, including (1) the boot code, (2) the main application code, (3) the mobile agent PIVC, and (4) the flash downloader. The boot code is the program to be executed before the sensor passes the verification and is used for the sensor to communicate with the PIVS and to execute the PIVC. Verification of the entire flash ensures no malicious code hidden in the flash that could exploit the flash downloader.

5.3.2 *Flash Free Space Compression Attack.* The free space in the flash can be used by the attacker to hide malicious code, and the original data in the free space can be compressed for later verification. The flash free space is shown in Figure 5, along with the flash memory layout of different program components

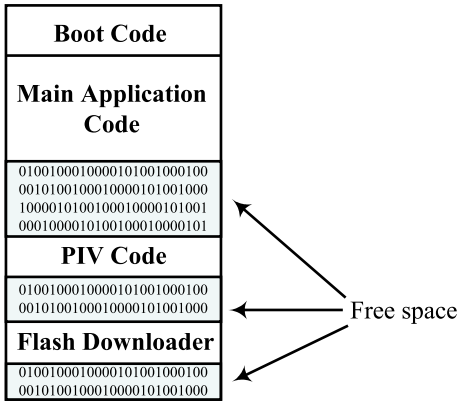

Fig. 5. The sensor flash memory layout in PIV.

in PIV. The attacker can use the compressed free space data for verification by uncompressing part of the data at a time for hashing and still keep the extra free space to hide the malicious code. Therefore, even hashing the entire flash for verification will not be able to counter this attack.

We counter this attack by filling the flash's free space with incompressible bit strings before deploying the sensor. By placing incompressible bit strings in the flash's free space, the attacker can neither compress the flash's free space nor gain more flash space to exploit.

5.3.3 *Malicious Mobile Agent PIVC Attack.* Another possibility is to have the attacker place malicious code in the flash and put the real code in SRAM, EEPROM, or the additional data flash memory. The attacker can have a malicious PIVC that performs verification normally, but, when validating the part of the flash that the real code is now stored in SRAM, EEPROM, or the additional data flash memory, use the real code instead. By using the space of SRAM, EEPROM, or the additional data flash memory, it is possible for the attacker to place other changes to the program in the flash.

Another attack exploiting the malicious PIVC is the application code compression attack. The attacker can compress part of the original application code and use the free space in the flash to store malicious code. When performing the verification, the attacker can then uncompress the compressed parts of the original code (or part thereof) and use them for verification and can still keep the extra free space to hide the malicious code.

One possible countermeasure is to set strict timing constraints on the hashing algorithm used for verification, and these attacks can be prevented using the tight upper and lower bounds for the hash-time interval. Because SRAM, EEPROM, and the additional data flash memory are much slower than the flash, they require more CPU clock cycles to read from. Therefore, if we set a tight hash-time interval, then the attacker will not be able to produce the correct hash values within the tight hash-time interval. The tight hash-time interval can also be used to counter application-compression attacks, because

uncompressing the compressed original code will take additional time, making it highly unlikely for the attacker to get the correct hash values within the hash-time interval.

Similar to the scheme by Shaneck and colleagues [2005], the hash-time interval should be set to the expected time for the PIVS to receive the hash value from a sensor, which is the sum of the time taken to compute the hash on the sensor program, the network roundtrip time, and the expected response delay that accounts for network delay. Any hash values returned from sensors that are within the hash-time interval will be accepted or will otherwise be rejected.

5.3.4 *Compromised PIVSs and Sensors Attack.* After compromising a PIVS, the attacker may use it as a back-end server to pass the verification of compromised sensors. For example, when a sensor receives the PIVC for verification of its program, it can then forward the PIVC to the compromised PIVS and ask it to compute the hash value for it. The compromised sensor will then be able to have a correct hash value to pass the verification. Another similar attack is possible if the attacker compromises a few sensors in the network and uses them to cooperate with each other. In this case, the attacker can store its original code in the other compromised sensors and let them use its code to compute the hash value and help it to pass verification.

The tight hash-time interval mentioned above can also be used to handle this attack. Because it must take a longer time for the sensors to communicate with the compromised PIVS/sensors and compute the hash value of its program, it is not possible for the attacker to generate and return the hash value to the PIVS within the hash-time interval.

6. IMPLEMENTATION OF DAPP AND PIV

This section describes our implementation of the DAPP and the PIV protocol [Park and Shin 2005] on Mica2 Motes [Crossbow] and laptops. Mica2 sensor nodes have 128K bytes of in-system reprogrammable flash and 4K bytes of internal SRAM and run under TinyOS. For laptops, we used Java to write the PIVS, and for sensors, we used nesC [Gay et al. 2003] 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.

6.1 Changes to the Original PIV Design

We made some modifications to the original PIV designed by Park and Shin [2005], which are listed with their justification.

6.1.1 *Hash Functions: RHF vs. HMAC-MD5.* The authors of PIV [Park and Shin 2005] proposed a special class of cryptographic hash functions, called randomized hash functions (RHFs). In addition to random hash computation, RHFs provide two ways of computing the hash value, that is, one from the program in the sensor and the other from the digest of the sensor program stored in the PIVS. However, because the free space in the flash of a sensor can be used by an attacker to hide malicious codes, as shown in Figure 6 and described in Section 5.3.2 as the flash free space compression attack, we need to fill up

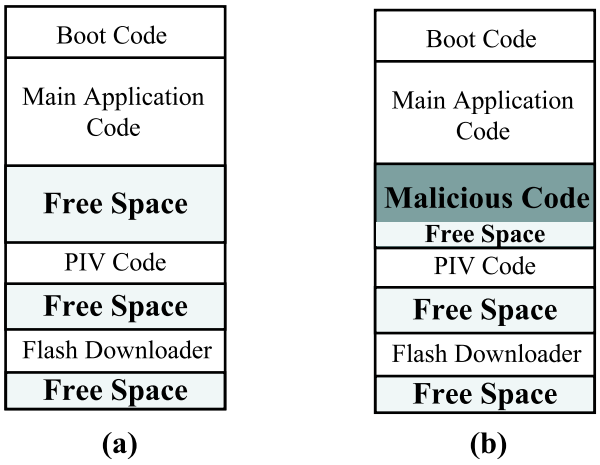

Fig. 6. (a) Normal sensor flash memory layout; (b) sensor flash memory layout with malicious code hidden in the free space.

the free space in the flash with some random incompressible bit strings before deploying the sensor. Unfortunately, the use of incompressible bit strings in flash may remove the advantage of storing digests in the PIVS memory instead of storing the entire program code of the sensor flash.

The digests of sensor programs stored in a PIVS as proposed in PIV [Park and Shin 2005] were created as follows. B program blocks, $\mathbf{x}_1, \ldots, \mathbf{x}_B$, were built from the original program $\mathbf{x}$, where $\mathbf{x}_l = [x_{l,1}, \ldots, x_{l,m}]^T$ was an $m \times 1$ vector and $x_{l,i} \in \mathbf{F}$.[1] A digest for $\mathbf{x}_l$ was defined as an $m \times m$ matrix X_l, which consists of all quadratic terms, $x_{l,i}\, x_{l,j}$. That is, $X_l = \mathbf{x}_l \mathbf{x}_l^T = \left(x_{l,i}\, x_{l,j}\right)$. One might think that a digest X_l is actually m times larger than the original program block $\mathbf{x}_l$. However, the size of the total digests will be smaller than the size required for just storing all sensor programs because there exist common program blocks for all sensors due to their similar purpose of service. Therefore, multiple digests were combined into just one digest, thus requiring a smaller memory size.

By storing incompressible and unique bit strings for each sensor, the common digests will decrease, and storing digests may require more memory than simply storing the sensor programs. For the purpose of defending the flash free space compression attack, we had to store incompressible strings in the flash free space for each sensor; thus, there was little advantage of using RHFs for hash computation. Therefore, we decided to store sensor programs instead of digests in PIVSs, not using RHFs for hash computation.

Instead, we decided to use HMAC [Krawczyk et al. 1997], a mechanism for message authentication using cryptographic hash functions, together with an iterative cryptographic hash function MD5 [Rivest 1992], in combination with a secret shared key. The reason for using HMAC for verification of sensor

[1] $\mathbf{x}^T$ (A^T) is the transpose of a vector $\mathbf{x}$ (a matrix A).

programs is that HMAC can be used in combination with any iterated cryptographic hash functions, such as MD5 or SHA1. Therefore, we can switch the hash function when needed. HMAC also uses a secret key for the calculation and verification of the message authentication values, which meets our need for using different secret keys to verify the sensor programs for each verification. MD5 is a widely used cryptographic hash function, and even though it has been shown to be vulnerable to hash collisions [Wang et al. 2004], because of the way hash functions are used in the HMAC construction, the techniques used in the MD5 hash collision attacks do not apply to HMAC-MD5.

6.1.2 *The Transmission of the PIVC.* In the original PIV design, every time a sensor asks for verification, the PIVS sends the entire PIVC to the sensor node to initiate the verification. However, transferring the whole PIVC to the sensor each time before verification incurs excessive network traffic. Moreover, because the flash in Mica2 Mote can allow only 10,000 erases or writes, allowing the PIVC to be written to the sensor flash before each verification is not a good approach. Therefore, we stored the PIVC in the flash before deployment to reduce network traffic and reduce flash erases and writes. Because the flash is the only place in memory where the PIVC can be executed, it is a good location to place the PIVC.

A version number for the PIVC was assigned and placed at the last part of the PIVC to avoid repeated transmissions. The PIVC version number is first checked by the sensor with the PIVS to see if the PIVC is up-to-date before the sensor executes the PIVC and begins computing the hash value of its program. If the PIVC version number differs from the current PIVC version number on the PIVS, then the PIVS will transmit the new PIVC to the sensor. This use of the PIVC version number allows the PIVC to be updated, if necessary. If the PIVC version number matches the current PIVC version number on the PIVS, then the sensor will need to request only the hash key from the PIVS and execute the PIVC already in the flash with the received hash key to perform the hash computation, which, in our implementation, is the HMAC-MD5 computation.

Because the PIVC will not change very often, the PIVS will transmit only the hash key instead of the entire PIVC to the sensor. By making this modification of keeping the PIVC in the sensor flash before deployment, we can save energy on the sensor, reduce network traffic, and extend the sensor flash life with less erases and writes on the flash.

6.2 Message Authenticity and Integrity

In DAPP, we used MAC to achieve message authenticity and integrity to allow sensors to authenticate PIVSs. The security of the MAC depends on the length of the MAC value. Conventional security protocols use 16-byte MACs. We chose to use HMAC-MD5 [Krawczyk et al. 1997; Rivest 1992] for generating MACs in our implementation and truncated the output of the MAC to use a 10-byte MAC.

Most well-known and widely used MAC algorithms are CBC-MAC and HMAC [Krawczyk et al. 1997]. CBC-MAC (cipher block chaining message

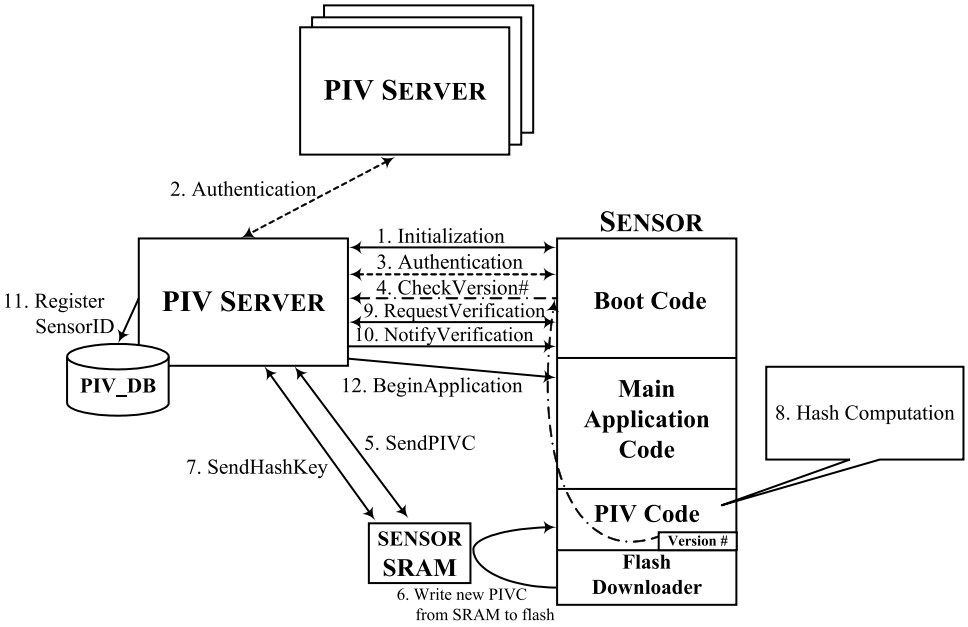

Fig. 7. Overview of our implementation of DAPP and PIV.

authentication code) uses block ciphers in CBC mode to create a MAC. HMAC is a keyed hash message authentication code and is calculated using a cryptographic hash function in combination with a secret key. These algorithms were evaluated using Crypto++ 5.2.1 benchmarks [Crypto++], which are speed benchmarks for some of the most commonly used cryptographic algorithms. In Crypto++ 5.2.1 benchmarks evaluation, HMAC-MD5 outperforms CBC-MAC-AES and is three times faster. Mills and colleagues [Burnside et al. 2002] analyzed the memory requirements for HMAC-MD5 on the Atmel processor. They showed the code size for HMAC-MD5 is 4.6K bytes and the data size is 386 bytes, which are fine for our implementation. Therefore, we decided to implement HMAC-MD5 for the MAC computation in our DAPP implementation.

6.3 Overview of the Implementation

Figure 7 describes our implementation of DAPP and the modified PIV. A sensor starts DAPP with one PIVS, and other reference PIVSs authenticate the PIVS for the sensor. The sensor determines the success or failure of the authentication of the PIVS based on the responses of the authenticating PIVS and the authentication tickets sent from the reference PIVSs. If the PIVS passes the authentication, then the sensor will start the PIV protocol with the PIVS to verify its program.

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

6.3.1 *PIVS Development.* A MICA2 Mote sensor and a laptop together are used as a PIVS. They are connected with a serial line that forms the primary channel for wired communication. On the laptop, a simple Java application, SerialForwarder, provides 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 PIVSs over the radio. The received messages are relayed from the sensor to the laptop through the serial cable, and the sent messages are also relayed from the laptop to the sensor for broadcast or unicast.

The PIVS is written in Java, and we implemented HMAC-MD5 [Krawczyk et al. 1997; Rivest 1992] to generate MAC when PIVSs run DAPP. PIVSs use the shared pairwise key between the two PIVSs to generate and verify MACs. The pairwise key length is 16 bytes, and the MAC length is 10 bytes. The 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. All of the sensor programs are stored on the laptop as files in binary formats and are used for sensor verification. When sending the PIVC to a sensor for hash computation, the PIVS reads the PIVC from a PIVC binary file and sends 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. The interactions between PIVSs and a sensor are shown in Figure 7 as well as the responses of PIVSs handling the sensor requests with the arrows between PIVSs and the sensor indicating the exchanged messages. Once a PIVS sends the verification result to the sensor, it activates the sensor's main application code if the sensor passes the verification or otherwise locks the sensor, thus blocking it from joining the network.

On updating the PIVC or sending the hash key to the sensor, the sensor performs a simple error check by acknowledging to the PIVS the previous data it has received. If the acknowledging data is not the same as the previous data, then there was data corruption during the previous transmission, and the PIVS will retransmit the data to the sensor.

PIVSs randomly generate hash keys for each sensor verification during PIV. If the previously sent hash key bytes have been corrupted, the PIVS regenerates the corrupted bytes of the hash key and retransmits the hash key bytes to the sensor. This is to prevent the sensors from reporting the wrong hash key bytes and trying to gain additional time to generate the correct hash value for verification.

A snapshot of the PIVS interface running on a laptop is given in Figure 8. It shows the current status of the sensors interacting with the PIVS as well as the connectivity of the sensors to the PIVS. The PIVS can choose to broadcast or unicast to a particular sensor and to reset sensors or to start the sensors' main applications.

6.3.2 *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

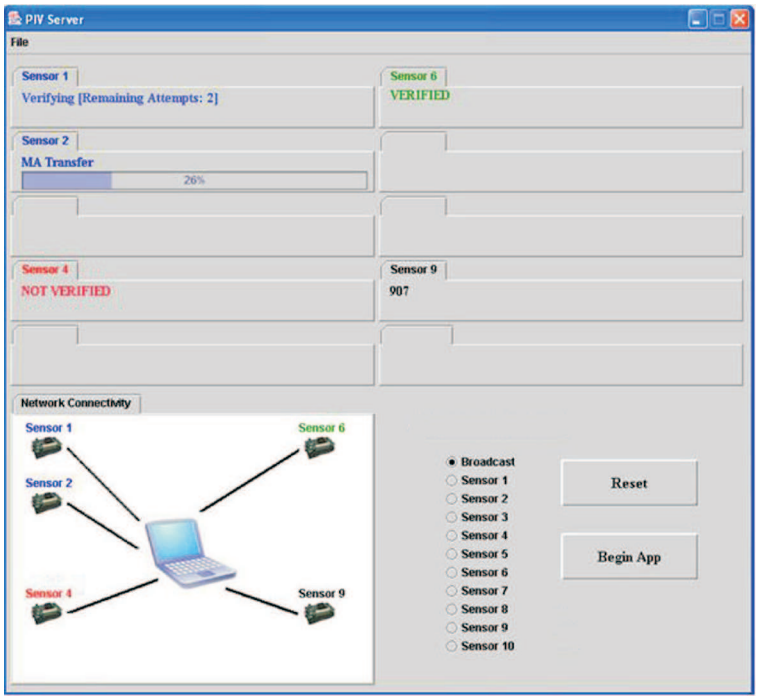

Fig. 8. A snapshot of the PIVS interface. It shows that Sensor 1 is verifying its program with the PIVS, Sensor 2 is in the middle of receiving the new version of the PIVC from the PIVS, Sensor 4 has failed the verification, Sensor 6 has just passed the verification, and Sensor 9 has already passed the verification and starts to run its main application.

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 [Gay et al. 2003], along with inline assembly mixed in nesC code.

After the communication between the sensor and the PIVS has built up, the Boot code jumps to the PIVC to get the version number of the PIVC and then sends it to the PIVS to check if the version number is up-to-date. If not, then the PIVS sends the new PIVC to the sensor, with four bytes of the PIVC per message. The bytes of the PIVC received by the sensor will first be stored in the sensor SRAM. After one page (128 words or 256 bytes) of the PIVC has been received, the Boot code jumps to the flash downloader and writes the page from the SRAM to the flash. We didn't use network programming for PIVC transmission and update because we are not reprogramming the entire sensor flash but only updating part of the flash with the PIVC.

After the entire PIVC has been written to the flash, the Boot code reports its new PIVC version number to the PIVS again. If the version numbers match, then the PIVS sends the hash key to the sensor for computing the hash value over its program. Finally, the Boot code jumps to the PIVC to start the hash computation over the entire flash and then sends back the hash value for verification. On receiving the verification result from the PIVS, either the Boot

code will activate the main application code on the sensor if the sensor passes the verification, or the sensor will otherwise be locked and unable to join the network.

Note that the Boot code is not trusted. If the Boot code does not work properly as it should, then the sensor will not be able to pass PIV. The PIV protocol offers three ways of actually locking a sensor: (1) The PIVS can ask the sensor's neighbor sensors not to replay packets from the sensor; (2) the key manager refreshes a new cluster key and excludes the sensor from the cluster; and (3) network services like routing may look up PIV_DB to ensure sensors are verified and thus genuine.

Because PIVSs and sensors communicate through the wireless network, message losses are common between nodes. The Boot code can also handle any message loss between PIVSs and sensors. Message losses are handled by timeouts and retransmissions. On its transmission of a message, the sensor starts a timer. If the timer has expired and the sensor still has not received any response from the PIVS, then the sensor will retransmit the message to the PIVS.

6.3.3 *PIVC Development.* The PIV Code (PIVC), or the mobile agent, is written in C along with inline assembly. The size of the PIVC is about 10K bytes, and it takes about five minutes to transmit the entire PIVC from the PIVS to the sensor. The main function of the PIVC is to perform HMAC-MD5 on the sensor over the entire sensor flash for verification. When performing HMAC-MD5 over the entire 128K bytes of the flash, the PIVC hashes the flash in 64-byte blocks and uses the intermediary hash value as the key for hashing the next 64-byte block. This way we enforced sequential hashing of the sensor flash. The hash keys the PIVC uses for HMAC-MD5 are 16 bytes long, and so are the hash values.

The reason for calculating the hash of the sensor flash in 64-byte blocks is related to our implementation of PIV and DAPP. Mica2 Mote has a SRAM of 4K bytes, which is too small to hold all the computational variables when the entire flash is hashed as a one shot. Therefore, we chose to implement the PIVC by hashing the flash in 64-byte blocks.

The security achieved by performing HMAC-MD5 on the entire 128K bytes of flash is not equivalent to that by sequentially performing HMAC-MD5 on 64-byte blocks. However, because MD5 operates on 64-byte blocks, if the hash message length is over 64 bytes, then MD5 will break up the message into blocks of 64 bytes and iterate over them with a compression function before doing the final hashing. By sequentially performing HMAC-MD5 on 64-byte blocks one-by-one, we perform hashing multiple times instead of just once after iterating the data over the compression function. The cryptographic strength of sequentially hashing in 64-byte blocks is not much worse than hashing the entire 128K bytes at once.

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; thus, the new version of the PIVC will work correctly. While updating the new PIVC, the PIVS sends the binary file of the new PIVC for the sensor to write the new PIVC to the flash for execution.

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 the sensor still holds the up-to-date PIVC version number. If the PIVC version number is to be updated first, then after a failure in the transmission of the PIVC, the sensor will have the up-to-date PIVC version number but not the correct PIVC. When the sensor tries to verify with the PIVS again, the PIVC is not updated because the sensor has the up-to-date PIVC version number. The sensor will then fail the verification by hashing the flash using the incorrect PIVC.

Because the PIVC needs to coexist with the Boot code, the main application code, and the flash downloader in the flash, and their 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 other part of the code and their variables. We use the `avr-objdump` command in TinyOS to create a dump file to analyze the Boot code memory information. The PIVC flash location is then computed so that the PIVC is placed below the Boot code and the main application code in the flash. The flash downloader and the PIVC can be placed at the very end of the flash, so the main application code can occupy the rest of the flash below the Boot code and above the PIVC. The PIVC flash location can be set once the size of the application code is decided, or it can be placed right above the flash downloader. The flash location is set inside the PIVC C program and the Boot code. When compiling and linking the PIVC C program, we manually assign the SRAM location for the PIVC. The PIVC and the Boot code also use SRAM for passing variable values, such as passing the PIVC version number from the PIVC to the Boot code.

6.3.4 *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. It is mainly used for the PIVC update, to write the new PIVC received from a PIVS from SRAM to the sensor flash for execution.

The sensor flash is divided into two constant sections, the read-while-write (RWW) section and the no read-while-write (NRWW) section [Atmel]. Figure 9 shows the limit between the two sections in the sensor flash.

The main difference between the two sections is that while erasing or writing a page inside the RWW section, the NRWW section can be read; however, the inverse is not true. 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 at the end of the flash to allow reading while writing a page of the PIVC to the RWW section in the flash.

We use self-programming on the sensor to write data from SRAM to the flash. The program memory updates itself page-by-page. Before writing a page to the flash with the data stored in the SRAM, the flash downloader first performs a page erase on the flash. It then fills in the temporary page buffer one

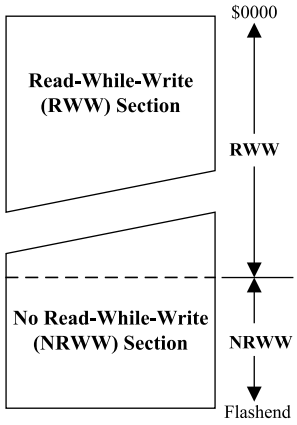

Fig. 9. Read-while-write section vs. no read-while-write section in the sensor flash.

word at a time with the data in the SRAM. It finally performs 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 evaluate the performance of DAPP 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, demonstrating that DAPP is scalable and efficient in computation, communication, and storage in sensor networks.

7.1 Evaluation with Simulation

We simulated DAPP with randomly generated networks consisting of 1,000 sensors and 250 PIVSs in a 1,000 $\times$ 1,000 unit area. We assume each sensor has a communication range of 150 units and that a PIVS normally communicates with others within 200 units from itself. Each sensor node is initialized with 0.5 J of energy. Once a sensor node exhausts its battery, it will stop working.

We simulated and compared the number of sensors surviving and continuing to work using DAPP with that using an AS in the network for authentication, to see how many sensors survive with our DAPP approach. When DAPP is used for authentication, PIVSs monitor and cooperate to authenticate one another, but the sensor needs to communicate only with the authenticating PIVS. In our DAPP simulation, we chose N_{auth} = 5, i.e., a PIVS needs to present five authentication tickets to a sensor to pass authentication. In our AS simulation, the AS is placed at (x = 1,070, y = 1,070). When a dedicated AS is employed, the sensors need to take multiple hops to reach the AS, making those sensors near the AS relay others' messages.

The simulation time is divided into rounds of actions. In each round, each sensor has the probability $ReVerify$ that it needs to reverify its program with

a PIVS and thus reauthenticate the PIVS. We performed simulation while changing $ReVerify$ and plotted the results in Figure 10, showing the numbers of sensors surviving using (1) DAPP and (2) a dedicated AS for authentication with $ReVerify$ = 0.05 and 0.1, respectively. More sensors are shown to survive or a longer life time with DAPP than for the case of using a single AS in the network. The advantage of DAPP becomes more pronounced, especially when the network is deployed in a highly hostile environment or sensors need to be reverified more often.

Figure 10 also shows that when using a dedicated AS for authentication, the curve of the number of sensors surviving in the network cuts off smoothly but not sharply. This is because all the sensors around the AS exhaust their batteries first, and then the sensors closer to the AS deplete theirs. Sensors exhaust their batteries gradually depending on their distance to the AS. In contrast, when using DAPP for authentication, the curve of the number of sensors surviving in the network cuts off more sharply because all the sensors, irrespective of their location, use almost the same amount of energy for authentication and will exhaust all their batteries approximately at the same time.

For the same network, we also compared the number of messages exchanged and the average sensor's energy consumption by using DAPP and a dedicated AS for authentication. In the case of using DAPP for authentication, inter-PIVS communications will not interfere with sensor communications because PIVSs have dual radio interfaces, one for communication between the sensors and one for communication between the PIVSs. For each sensor to authenticate one PIVS, there are two messages exchanged between the sensor and the PIVS. Because there are 1,000 sensors in the network, at least 2,000 messages must be exchanged in the network for each sensor to authenticate one PIVS, assuming that no transmission error occurred. When a single AS is used in the network for authentication, the sensors exchange an average of 22,766 messages in the network by counting the messages relayed by other sensors as separate messages. As a result, DAPP reduces the sensor communication traffic in the network by more than 90% as compared to the case of using a single AS for authentication.

Using DAPP for authentication also reduces the energy consumption on each sensor. With DAPP for authentication, one sensor dissipates, on average, 1,114 μJ to authenticate one PIVS. With a single AS for authentication, one sensor dissipates, on average, 7,624.5 μJ to authenticate one PIVS. The DAPP's energy consumption on each sensor improves up to 85% over the case of using an AS for authentication. Given an initial sensor energy of 0.5 J, a sensor can authenticate a PIVS 449 times with DAPP but only 66 times with a single AS.

The increase of sensor communication traffic and energy consumption under a single AS for authentication comes from sensors relaying authentication messages for other sensors. It is easy to see that the sensors deployed near the AS will exhaust their batteries faster than others due to their relaying of more messages for other sensors. However, sensors' lifetimes are extended by using DAPP because it authenticates PIVSs in a distributed manner and can

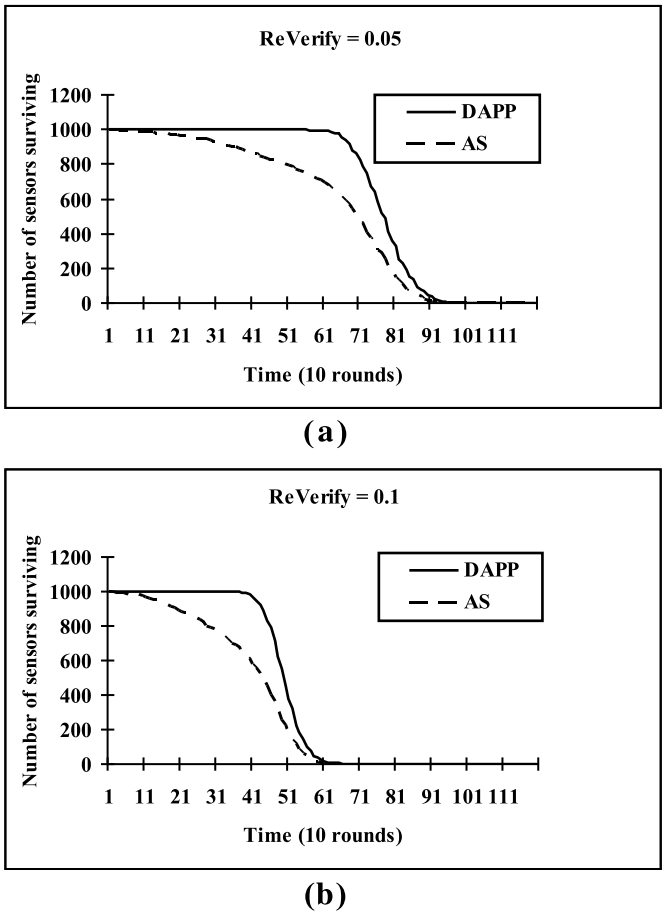

Fig. 10. Numbers of sensors that survive with DAPP and with an AS for authentication, with (a) *ReVerify* = 0.05 and (b) *ReVerify* = 0.1. Sensor life time is longer using DAPP than the case of using a single AS in the network.

thus reduce communication traffic and energy consumption by authenticating PIVSs locally.

7.2 Computation Cost

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

7.2.1 *Pairwise Keys Establishment.* In DAPP, two nodes (composed of two sensors, two PIVSs, or a sensor and a PIVS) establish a pairwise key to authenticate their identities to each other using the Blundo scheme [Blundo et al. 1992]. Each node needs to compute k modulo multiplications and k modulo additions for a k-degree polynomial to generate a pairwise key. As stated by Zhu

and colleagues [2004], if we choose k to be 100, the pairwise key size 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 1/10,000 of that of creating an RSA signature, or of the same order of the cost for computing an AES encryption. Also, Liu and colleagues [2005] showed that the computational cost of pairwise key establishment using a k-degree polynomial grows linearly with respect to k. Therefore, the computational cost of pairwise key establishment using a k-degree polynomial for $k = 200$ will be twice as much as the computational cost for $k = 100$, still much more efficient than the computational cost for creating an RSA signature.

Wander and colleagues [2005] compared the energy consumption of RSA and elliptic curve digital signature algorithm (ECDSA) on the low-power microcontroller Atmel ATmega128L. They showed the energy cost of creating an ECDSA signature is about 3/40 of that of creating an RSA signature. Therefore, the computational cost of pairwise key establishment using the Blundo scheme for $k = 200$ is about 1/375 of that of creating an ECDSA signature.

7.2.2 *MAC Generation and Verification.* We used HMAC [Krawczyk et al. 1997] 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 Sancak and colleagues [2004], HMAC consumes approximately 45.6 μJ if it runs on a Mote. Carman and colleagues [2000] analyzed the impact of security algorithms on energy consumption for sensor nodes, showing that the computation of HMAC-MD5 for a 1024-bit message is energy-cost-effective for numerous microprocessors.

7.3 Communication Cost

The communication overhead of DAPP is associated with a PIVS's authentication request to its neighbor PIVSs. For a PIVS to authenticate another PIVS, two messages need to be transmitted. First, a PIVS has to authenticate itself with N_{auth} neighbor PIVSs to pass the authentication; then the neighbors reply. Therefore, there will be $2N_{auth}$ messages transmitted to authenticate a PIVS. However, DAPP is used only before a sensor runs PIV and wants to authenticate a PIVS. Because a sensor runs the PIV protocol only infrequently, the communication overhead is not high. Also, because PIVSs have dual radio interfaces, the communications between PIVSs do not interfere with sensor communications.

7.4 Storage Requirement

Here we show that the memory requirement for sensors and PIVSs to store the pairwise keys is very small. Each sensor and PIVS needs to store a k-degree polynomial for establishing pairwise keys, and the polynomial occupies $\frac{(k+1)\log q}{8}$ bytes. For a sensor to authenticate one PIVS, it needs to establish a pairwise key with it. The sensor also needs to decrypt the N_{auth} authentication tickets issued by the authenticating PIVS's reference PIVSs. Therefore, in DAPP, each sensor needs to establish at least $N_{auth} + 1$ keys before it can authenticate a PIVS. Because each key is 128 bits (or 16 bytes) long, if we

choose $N_{auth} = 5$, then a sensor needs to store only six keys, and a total of 96 bytes suffices. Therefore, keys require only 2.3% of the sensor SRAM because a Mica2 Mote has 4K bytes of SRAM. Similarly, PIVSs use the same polynomial to establish pairwise keys, and because PIVSs are equipped with more memory than sensors, they can store more keys than sensors.

Because the PIVSs are storing only a window of history instead of logging the entire history of PIVSs' transmissions and they are just monitoring the neighboring PIVSs instead of all the PIVSs in the network, using the behavior-based anomaly detection should require an insignificant amount of memory. Also, because PIVSs are equipped with more memory than sensors, the storage cost for using the anomaly detection on PIVSs should not be a concern.

8. RELATED WORK

In this section, we review the related work that provides possible solutions for authentication and security mechanisms in ad hoc networks. We also discuss the related work in admission control in ad hoc networks. Last, we include and compare some work related to software verification in sensor networks and list some of them in which the Blundo scheme [Blundo et al. 1992] is used as their design basis.

Weimerskirch and Thonet [2001] presented a security model for low-value transactions, especially focusing on authentication in ad hoc networks. They used the recommendation protocol from the distributed trust model [Abdul-Rahman and Hailes 1997] 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. They also introduced the idea of threshold cryptography [Desmedt and Frankel 1989] in which, as long as the number of compromised nodes is below a given threshold, the compromised nodes cannot harm the network operation.

Hubaux and colleagues [2001] listed the threats and possible solutions for basic mechanisms and security mechanisms in mobile ad hoc networks. They developed a self-organizing public-key infrastructure. In their system, certificates are stored in local certificate repositories and distributed by the users. Bauer and Lee [2005] also proposed a distributed authentication scheme that is efficient and robust using the well-known concepts of "secrets sharing" cryptography and group "consensus." However, this scheme still needs a centralized processing center (PC) that is responsible for coordinating the distribution of secret keys to each node in the network, thus lowering the value of its distributed nature.

Saxena and colleagues [2005; Castelluccia et al. 2005] proposed secure, efficient and noninteractive admission control protocol and schemes that allow a pair of nodes to compute a shared key without centralized support in ad hoc networks. Without the assistance of any centralized trusted authority, they also use secret sharing techniques based on bivariate polynomials. In contrast, our work focuses on authentication of servers, while their work features admission control and pairwise key establishment.

Several researchers studied software verification in sensor networks. Our work is an extension to PIV [Park and Shin 2005], which verifies the integrity of the program and data stored in a sensor device. SWATT [Seshadri et al. 2004] is a software-based memory attestation technique that externally attests the code, static data, and the configuration settings of an embedded device. Secure code update by attestation (SCUBA) [Seshadri et al. 2006] enables secure detection and recovery from sensor node compromise. It is based on indisputable code execution (ICE) to guarantee unhampered execution of code even on a compromised node. Shaneck and colleagues [2005] proposed a software-based approach to verification of the integrity of a sensor's memory contents over the network without requiring any physical contact with the sensor.

Recently, many researchers in the area of sensor networks also use the Blundo scheme [Blundo et al. 1992]. Liu and colleagues [2005] used the Blundo scheme as a basis in their proposed scheme for establishing pairwise keys in distributed sensor networks. Zhu and colleagues [2004] presented an interleaved hop-by-hop authentication scheme that guarantees the base station to detect any injected false data packets. They used the Blundo scheme to establish multihop pairwise keys. Zhang and colleagues [2005] proposed several efficient schemes to restrict the privilege of a mobile sink without impeding its capability of performing any authorized operations for an assigned task. They also used the Blundo scheme for pairwise key establishment.

9. CONCLUSIONS AND FUTURE WORK

In this article, we presented a distributed authentication protocol of PIVSs (DAPP) for sensors to authenticate PIVSs in sensor networks, and implemented DAPP and the PIV protocol [Park and Shin 2005] on Mica2 Motes. Along with DAPP, we also developed a PIVS revocation mechanism for PIVSs to revoke malicious PIVSs detected in the network. Numerous modifications and improvements were also made to the original PIV design.

Our main contribution is the development of DAPP to achieve the authentication of PIVSs in a distributed manner without requiring a dedicated and trusted authentication server (AS), an important departure from PIV [Park and Shin 2005]. DAPP maintains the distributed nature of sensor networks and reduces the sensor communication traffic in the network by more than 90% and the energy consumption on each sensor by up to 85%, compared to using a centralized trusted AS for authentication. We also show that DAPP is robust and secure against various attacks in sensor networks.

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 PIVSs in the network. Once the IDS identifies any malicious or malfunctioning sensors, it can collaborate with the PIV protocol to request the sensor to reverify with the PIVS. Once a malicious PIVS is detected in the network, other PIVSs can collaborate to revoke it. Unfortunately, there is not much work done on intrusion detection for sensor networks. This is a matter for our future inquiry.

REFERENCES

ABDUL-RAHMAN, A. AND HAILES, S. 1997. A distributed trust model. In *Proceedings of the 1997 Workshop on New Security Paradigms*.

ATMEL. 8-bit AVR microcontroller with 128 KBytes in-system programmable flash — ATmega128, ATmega128L. www.atmel.com/dyn/resources/prod_documents/doc2467.pdf.

BAUER, K. AND LEE, H. 2005. A distributed authentication scheme for a wireless sensing system. In *Proceedings of the 2nd International Workshop on Networked Sensing Systems (INSS05)*.

BLUNDO, C., SANTIS, A. D., HERZBERG, A., KUTTEN, S., VACCARO, U., AND YUNG, M. 1992. Perfectly-secure key distribution for dynamic conferences. In *Proceedings on Advances in Cryptology (CRYPTO92)*.

BURNSIDE, M., CLARKE, D., MILLS, T., DEVADAS, S., AND RIVEST, R. 2002. Proxy-based security protocols in networked mobile devices. In *Proceedings of ACM Symposium on Applied Computing (SAC02)*.

CARMAN, D. W., KRUUS, P. S., AND MATT, B. J. 2000. Constraints and approaches for distributed sensor security. Tech. rep. 00-010, NAI Labs, Network Associates, Inc.

CASTELLUCCIA, C., SAXENA, N., AND YI, J. H. 2005. Self-configurable key pre-distribution in mobile ad hoc networks. In *The 4th International IFIP-TC6 Networking Conference*.

CROSSBOW. MICA2 - wireless measurement system. www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf.

CRYPTO++. Crypto++ 5.2.1 Benchmarks. www.eskimo.com/~weidai/benchmarks.html.

DESMEDT, Y. G. AND FRANKEL, Y. 1989. Threshold cryptosystems. In *Proceedings on Advances in Cryptology (CRYPTO89)*.

DIFFIE, W. AND HELLMAN, M. E. 1976. New directions in cryptography. *IEEE Trans. Inform. Theory 22,* 6, 644–654.

DOUCEUR, J. R. 2002. The sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02)*.

GANERIWAL, S., CAPKUN, S., HAN, C.-C., AND SRIVASTAVA, M. B. 2005. Secure time synchronization service for sensor networks. In *Proceedings of the 4th ACM Workshop on Wireless Security (WiSe05)*.

GAY, D., LEVIS, P., VON BEHREN, R., WELSH, M., BREWER, E., AND CULLER, D. 2003. The NESC language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*.

HUBAUX, J.-P., BUTTYÀN, L., AND CAPKUN, S. 2001. The quest for security in mobile ad hoc networks. In *Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc01)*.

KARLOF, C. AND WAGNER, D. 2003. Secure routing in wireless sensor networks: Attacks and countermeasures. In *Proceedings of the 1st IEEE International Workshop on Sensor Network Protocols and Applications*.

KRAWCZYK, H., BELLARE, M., AND CANETTI, R. 1997. HMAC: Keyed-hashing for message authentication. IETF Network Working Group, RFC 2104.

LIU, D., NING, P., AND LI, R. 2005. Establishing pairwise keys in distributed sensor networks. *ACM Trans. Inform. Syst. Secur. 8,* 1, 41–77.

PARK, T. AND SHIN, K. G. 2005. Soft tamper-proofing via program integrity verification in wireless sensor networks. *IEEE Trans. Mobile Comput. 4,* 3, 297–309.

PERRIG, A., SZEWCZYK, R., WEN, V., CULLER, D., AND TYGAR, J. D. 2001. Spins: Security protocols for sensor networks. In *Proceedings of the 7th International Conference on Mobile Computing and Networks (MobiCom01)*.

RIVEST, R. 1992. The MD5 message-digest algorithm. IETF Network Working Group, RFC 1321.

SANCAK, S., CAYIRCI, E., COSKUN, V., AND LEVI, A. 2004. Sensor wars: Detecting and defending against spam attacks in tactical adhoc sensor networks. In *2004 IEEE International Conference on Communications (ICC04)*.

SAXENA, N., TSUDIK, G., AND YI, J. H. 2005. Efficient node admission for short-lived mobile ad hoc networks. In *Proceedings of the 13th IEEE International Conference on Network Protocols (ICNP05)*.

SESHADRI, A., LUK, M., PERRIG, A., VAN DOORN, L., AND KHOSLA, P. 2006. Scuba: Secure code update by attestation in sensor networks. In *Proceedings of the 5th ACM Workshop on Wireless Security (WiSe06)*.

SESHADRI, A., PERRIG, A., VAN DOORN, L., AND KHOSLA, P. 2004. Swatt: Software-based attestation for embedded devices. In *Proceedings of the IEEE Symposium on Security and Privacy*.

SHANECK, M., MAHADEVAN, K., KHER, V., AND KIM, Y. 2005. Remote software-based attestation for wireless sensors. In *European Workshop on Security and Privacy in Ad-hoc and Sensor Networks (ESAS05)*.

SUN, K., NING, P., AND WANG, C. 2006. Secure and resilient clock synchronization in wireless sensor networks. *IEEE J. Select. Areas Comm. 24,* 2, 395–408.

WANDER, A. S., GURA, N., EBERLE, H., GUPTA, V., AND SHANTZ, S. C. 2005. Energy analysis of public-key cryptography for wireless sensor networks. In *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications (PERCOM05)*.

WANG, X., FENG, D., LAI, X., AND YU, H. 2004. Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199.

WEIMERSKIRCH, A. AND THONET, G. 2001. A distributed light-weight authentication model for ad hoc networks. In *Proceedings of the 4th International Conference on Information Security and Cryptology (ICISC01)*.

ZHANG, W., SONG, H., ZHU, S., AND CAO, G. 2005. Least privilege and privilege deprivation: Towards tolerating mobile sink compromises in wireless sensor networks. In *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc05)*.

ZHU, S., SETIA, S., AND JAJODIA, S. 2003. Leap: Efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS03)*.

ZHU, S., SETIA, S., JAJODIA, S., AND NING, P. 2004. An interleaved hop-by-hop authentication scheme for filtering false data injection in sensor networks. In *IEEE Symposium on Security and Privacy*.

Received February 2007; revised August 2007; accepted September 2007