Security and Collaboration Protocols for Mobile and Sensor Networks

by

Katharine Chang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2012

To my parents, Karen, and Wei-Ting,

with love and thanks.

ACKNOWLEDGEMENTS

My graduate school journey had been a long and rough ride for me. It was not possible for me to complete this journey without the help and encouragement from people around me. I am very grateful and would like to express my gratitude to all the people who had walked along with me and supported me through the years.

First, I would like to express my deepest gratitude to my advisor, Professor Kang G. Shin, for his continuous guidance, encouragement, support, and belief in me. He gave me much freedom and independence in exploring many interesting research topics. He encouraged me in life like a father and urged me for rigorous research like a mentor. I was very fortunate to work with him and I believe the training I received from him will prepare me well for my future career.

I would like to thank my committee members Professor Atul Prakash, Professor J. Alex Halderman, and Professor Qiaozhu Mei for their valuable and insightful comments and feedbacks to help me improve the quality and depth of my dissertation. I appreciate the opportunity to work with Professor Atul Prakash, and had learned many system software building and technical skills. I am also grateful and fortunate to receive the Bell Labs Graduate Fellowship that supported me for part of my graduate study. I would like to thank my mentors, Dr. Eric Grosse and Dr. Girija Narlikar, for guiding me during my summer internship at Bell Labs. I would also like to thank my mentor and collaborator, Dr. Xinwen Zhang, for guiding me during my internship at Huawei, teaching me how to explore new research ideas, and being a caring friend.

I am grateful to all the former and current Real-Time Computing Laboratory members for their friendship and technical assistance. I would like to especially thank the security group members for their helpful discussions and collaborations, especially Taejoon Park, Min-Gyu Cho, Matthew Knysz, Xin Hu, and Yuanyuan Zeng. Many thanks to Chun-Ting Chou, Chang-Hao (Howard) Tsai, Kai-Yuan Hou, Zhigang Chen, Kyu-Han Kim, Pradeep Padala, Xinyu Zhang, Xiaoen Ju, Caoxie (Michael) Zhang, and Antino Kim. I am thankful for what they shared with me professionally and personally over the years of my graduate study. My appreciation also goes to Stephen Reger for his administrative support.

I was very fortunate to have many wonderful friends to share special moments with me in my graduate study either in Ann Arbor or around the world. Many thanks to their friendship, encouragement, and support that made my long graduate study journey much more delightful. Special thanks to all my friends in Campus Crusade for Christ Bible studies for their friendship, prayers, and spiritual and physical support. I would like to thank many of my dear friends, but not limited to: Ya-Yunn Su, Po-Ju Lin, Nora Han, Yueh-Chuan Tzeng, Kai-Yuan Hou, Ching-Mei Lin, Juliette Kao, Hui-Yu Hsu, Howard Tsai, Po-Chun Hsu, Dianne and Chuck Roeper, Jane Liu, Galen Chen, Chun-Chi Hu, Li-Chu Lo, and Pei-Yeh Wu.

I would not be able to complete this dissertation without the unconditional love, tremendous support, and countless encouragement from my parents, sister Karen, and husband Wei-Ting. They were always there for me, believing in me, and giving me strength and comfort when I most needed them. I am very thankful and fortunate to have them on my side through all the ups and downs in my life. This dissertation is dedicated to my family.

Last, I would like to give my greatest thanks to God for His incomparable love and grace for me. I would like to honor Him for persevering through my graduate study. I pray God would continue to lead me for my life according to His plan.

TABLE OF CONTENTS

DEDICATION ii

ACKNOWLEDGEMENTS iii

LIST OF FIGURES viii

ABSTRACT xi

CHAPTER

I. Introduction 1

1.1 Background and Motivation 1
1.2 Research Goals 3
1.3 Overview and Contributions of the Dissertation 5
1.4 Organization of the Dissertation 7

II. Distributed Authentication of Program Integrity Verification in Wireless Sensor Networks 8

2.1 Introduction 8
2.1.1 Organization 11
2.2 Background: Overview of PIV 12
2.3 Design Overview 14
2.3.1 Sensor Network Architecture and Limitations 14
2.3.2 Motivation 15
2.3.3 System Model 16
2.3.4 Attack Model 17
2.3.5 Overview of DAPP 19
2.4 DAPP Details 20
2.4.1 Initialization and PIVS Discovery Phase 22
2.4.2 PIVS Authentication Phase 25
2.4.3 PIVS Revocation 28
2.5 Security Analysis 32

2.5.1 Network Survivability 32
2.5.2 Compromised PIVSs Collusion 33
2.5.3 Probability of PIVS True Authentication 35
2.5.4 Defense Against Various Attacks in Sensor Networks 38
2.5.5 Security Issues and Possible Attacks to PIV 40
2.6 Implementation of DAPP and PIV 43
2.6.1 Changes to the Original PIV Design 43
2.6.2 Message Authenticity and Integrity 46
2.6.3 Overview of the Implementation 47
2.7 Performance Evaluation 55
2.7.1 Evaluation with Simulation 55
2.7.2 Computation Cost 59
2.7.3 Communication Cost 60
2.7.4 Storage Requirement 61
2.8 Related Work 61
2.9 Conclusion 63

III. Application-Layer Intrusion Detection in MANETs 65

3.1 Introduction 65
3.1.1 Organization 67
3.2 Why are MAs Needed for Intrusion Detection? 67
3.3 Assumptions, Attack Model, and System Architecture 69
3.3.1 Assumptions 69
3.3.2 Attack Model 70
3.3.3 System Architecture 70
3.4 Design of Application-Layer IDS 72
3.4.1 Local IDS for Intrusion Detection 73
3.4.2 MA Functions 74
3.4.3 MA Authentication and Authorization 77
3.4.4 MA Dispatching 80
3.4.5 Intrusion Response 83
3.5 Security Analysis 83
3.5.1 Defense Against Compromised MAs and Nodes . . . 83
3.5.2 Defense Against Various Attacks in Ad Hoc Networks 85
3.6 Evaluation 87
3.6.1 Simulation Setup 88
3.6.2 Results 88
3.6.3 Non-uniformly Distributed Network Simulation and Results 93
3.7 Related Work 99
3.8 Conclusion 101

IV. TGIS: Booting Trust for Secure Information Sharing in Dynamic Group Collaborations 102

4.1 Introduction . . . 102
4.1.1 Organization . . . 105
4.2 Motivation . . . 105
4.3 Cryptographic Primitives . . . 107
4.3.1 Hierarchical Identity-Based Encryption . . . 107
4.3.2 Attribute-Based Encryption . . . 109
4.4 Overview of TGIS . . . 110
4.4.1 System Architecture and Assumptions . . . 111
4.4.2 Attack Model . . . 113
4.4.3 Design Overview . . . 114
4.5 TGIS Protocol . . . 114
4.5.1 Domain Setup . . . 116
4.5.2 Group Setup . . . 117
4.5.3 User Enrollment . . . 118
4.5.4 Intra-group Communication . . . 120
4.5.5 Inter-group Collaboration . . . 121
4.5.6 Message Authentication . . . 122
4.6 Decentralizing TGIS . . . 122
4.7 Security Analysis . . . 124
4.7.1 Network Survivability . . . 124
4.7.2 Defense Against the Attack Model . . . 125
4.8 Implementation and Evaluation . . . 127
4.8.1 Prototype Implementation . . . 127
4.8.2 Performance . . . 127
4.9 TGIS Application User Interface . . . 129
4.10 Related Work . . . 130
4.10.1 Trust Management . . . 130
4.10.2 Access Control . . . 130
4.11 Conclusion . . . 131

V. Conclusions . . . 132

BIBLIOGRAPHY . . . 135

LIST OF FIGURES

Figure

2.1 Interactions among the AS, the PIVS, and the sensor during PIV. . 14

2.2 A design overview of DAPP. PIVSs interact with one another for mutual authentication. The solid lines represent the interactions between the PIVSs and the doted lines represent the interactions between the PIVSs and the sensor. 19

2.3 Interactions among PIVS A, PIVS A's reference PIVS B, and sensor E under DAPP. 28

2.4 DAPP pseudocode to be executed on sensor E, PIVS A, and PIVS B. 29

2.5 Relationship between the probability of PIVS true authentication and the number of compromised PIVSs. 37

2.6 Relationship between the probability of PIVS true authentication and the number of required authentication tickets from PIVS. 37

2.7 The sensor flash memory layout in PIV. 41

2.8 (a) Normal sensor flash memory layout; (b) Sensor flash memory layout with malicious code hidden in the free space. 44

2.9 Overview of our implementation of DAPP and PIV. 48

2.10 A snapshot of the PIVS interface. 50

2.11 Read-While-Write section and No Read-While-Write section in the sensor flash. 54

2.12 Numbers of sensors that survive with DAPP and with an AS for authentication, with (A) $ReVerify = 0.05$ and, (B) $ReVerify = 0.1$. 57

3.1 The IDS system architecture for MANETs. 71

3.2 The local IDS architecture on a mobile node. 72

3.3 The simulation environment for MA-based IDS. Laptops represent the MA servers and black circles represent mobile nodes in the network. 89

3.4 The relationship between different MA TTL values and numbers of MAs dispatched (including the extra sent MAs) by one MA server using the simulation environment in Figure 3.3(A). The number of nodes that did not receive any MA is shown in (A) and the number of nodes that did receive multiple MAs is shown in (B). 91

3.5 The relationship between different MA TTL values and numbers of MAs dispatched (including the extra sent MAs) by four uniformly positioned MA servers using the simulation environment in Figure 3.3(B). The number of nodes that did not receive any MA is shown in (A) and the number of nodes that did receive multiple MAs is shown in (B). 92

3.6 The relationship between different MA TTL values and numbers of MAs dispatched (including the extra sent MAs) by randomly-deployed MA servers. With four MA servers deployed, the number of nodes that did not receive any MA is shown in (A) and the number of nodes that did receive multiple MAs is shown in (B). With five MA servers deployed the number of nodes that did not receive any MA is shown in (C) and the number of nodes that did receive multiple MAs is shown in (D). 94

3.7 The simulated network quadrant layout. 95

3.8 The relationship between different MA TTL values and numbers of MAs dispatched (including the extra sent MAs) by one MA server using the simulation environment in Figure 3.3(A). With network deployment (1), the number of nodes that did not receive any MA is shown in (A) and the number of nodes that did receive multiple MAs is shown in (B). Simulation results for network deployment (2) and (3) are shown in (C), (D), and (E), (F) respectively. 96

3.9 Comparison of MA distribution results between different network deployments. Empty circles represent nodes in the network that did not receive any MA, and solid circles represent nodes that has received one or more MAs. 97

3.10 The relationship between different MA TTL values and numbers of MAs dispatched (including the extra sent MAs) by four uniformly positioned MA servers using the simulation environment in Figure 3.3(B). With network deployment (1), the number of nodes that did not receive any MA is shown in (A) and the number of nodes that did receive multiple MAs is shown in (B). Simulation results for network deployment (2) and (3) are shown in (C), (D), and (E), (F) respectively. 98

3.11 The intrusion detection system for MANETs proposed in [57]. . . . 99

4.1 HIBE system architecture. 108

4.2 TGIS System architecture. 112

4.3 TGIS System architecture using The University of Michigan as an example of a hierarchical organization. 112

4.4 System architecture of users in different organizations forming a dynamic group. 113

4.5 Snapshots of the location-based application over TGIS on Android. 128

ABSTRACT

Security and Collaboration Protocols for Mobile and Sensor Networks

by

Katharine Chang

Chair: Kang G. Shin

Research in network and computer system architecture is evolving beyond its traditional focus as mobile devices become ubiquitous and mobile computing triggers dramatic change in the computing world. Mobile devices can form heterogeneous mobile networks that provide distributed services and information access in real time from anywhere in the world. Coincident with this change, the assurance of network and system security and availability becomes an important problem. This problem is challenging because it requires the system to be easy to manage and operate, but also requires reliability and security. For the purpose of securing a network, we usually require authentication, authorization, and accounting. Authentication requires users to prove their identity. Accounting requires intrusion detection or forensic analysis to find attacks, if any, in the system. Finally, authorization requires access control to ensure data privacy.

This dissertation aims to design security and collaboration protocols to create a comprehensive trust framework to protect mobile and sensor networks by applying cryptographic algorithms. It makes three primary contributions. First, we propose and implement a distributed authentication protocol called DAPP in wireless sensor networks to allow sensors to authenticate servers without requiring a commonly-used trusted authentication server. DAPP maintains the distributed nature of sensor networks, has low computation and communication overhead, and is resilient to node compromises. Second, to attain security for nodes in mobile ad hoc networks, we present an intrusion detection system (IDS) architecture at the application layer to help detect malicious nodes in the network. We describe the design of this architecture and the use of mobile agents to augment each node's IDS. Finally, we design a trusted group-based information sharing protocol called TGIS for mobile devices to establish a trust relationship with collaborators and enforce data access control between collaborators with different privileges. TGIS is built upon existing trust infrastructures in individual organizations to enable trust management for group collaboration.

The security and collaboration protocols presented in this dissertation together achieve secure distributed authentication, authorization, and accounting in mobile and sensor networks.

CHAPTER I

Introduction

1.1 Background and Motivation

In recent years, mobile computing has changed the world significantly. As mobile devices become ubiquitous, people can connect to the Internet and access data and information from various locations. Mobile devices such as smartphones and wireless sensors can form heterogeneous mobile networks that provide distributed services and information access in real time. Mobile computing and sensor networks have changed the world from wired networks and desktop computers to a world of ubiquitous computing.

As research in network and computer system architecture evolves beyond its traditional focus, the guarantee of network and system security and availability becomes an increasingly important problem. This problem is challenging because it requires the system to be easy to manage and operate, but also requires reliability and security. For the purpose of securing a network, there are usually two lines of defense. The first line is intrusion prevention. Typical intrusion prevention measures, such as authentication and encryption, can prevent external nodes from disrupting or disabling the network. The second line of defense is intrusion detection that can discover the insider attacks mounted by compromised nodes in the network. Upon detection of an intrusion, a countermeasure is triggered to minimize damages to the network.

Intrusion prevention usually includes authentication and encryption. Authentication is the process of letting one party ensure the valid identity of another party to communicate with. Encryption, however, is the process of making information unreadable to anyone without the required key. Traditional intrusion prevention measures often need a centralized and trusted third-party server to issue credentials or encryption and decryption keys. Given the distributed structure of sensor and mobile networks, many traditionally used authentication and encryption mechanisms are no longer effective.

Intrusion-detection mechanisms used for traditional wired networks are also difficult to be used for mobile networks because of their architectural differences. Without centralized audit points like routers, switches, and gateways, mobile networks can only collect audit data locally and thus require a distributed and cooperative intrusion detection system. Moreover, the devices used in mobile networks and sensors in sensor networks are all small, and therefore computation- and energy-constrained. Therefore, when designing security protocols for the mobile devices, we need to design protocols that are lightweight and energy-efficient.

Achieving trust for secure information sharing in mobile groups, especially between devices in mobile networks, opens up a new area of research. Establishing a trust relationship between devices and allowing users to collaborate and communicate with their collaborators in a distributed fashion is also a new challenge. Achieving trust for secure information sharing in mobile groups requires authentication to attain trust and information encryption to achieve access control. Moreover, the collaboration protocols must be energy-efficient to suit the computation and energy-constrained mobile devices.

Overall, to protect a network system, we usually require authentication, authorization, and accounting, commonly known as the AAA protocol in computer security, essential technology to create a trusted environment. Authentication requires users to show passwords, biometrics, or tokens to prove their identities. Accounting, or sometimes referred to as auditing, requires intrusion detection or forensic analysis on the system to find attacks. Finally, authorization requires access control to ensure data privacy in the system, and that also becomes a challenge in mobile networks. This dissertation is motivated by the above challenges and aims to design security and collaboration protocols to create a comprehensive trust framework to protect mobile and sensor networks, and to achieve authentication, authorization, and accounting.

1.2 Research Goals

The goal of this dissertation is to design security and collaboration protocols for distributed networks to allow secure distributed authentication, authorization, and accounting by applying cryptographic algorithms. The protocols are designed to reduce network traffic and resource consumption on the network hosts and nodes. Without these protocols, it would be difficult to achieve the research goal because the existing protocols usually require trusted and centralized services and are difficult and impractical to realize in the distributed network environment. The security and collaboration protocols proposed in this dissertation are developed with the following research goals.

- **Achieving Distributed Authentication:** Authentication is the process of letting one party ensure the valid identity of another party to communicate with. It is important for nodes in distributed networks to have mutual authentication, as this will guard against many security attacks, such as impersonation, Sybil, and man-in-the-middle attacks. Traditional network authentication protocols involve a centralized and trusted authentication server (AS). Since the AS is needed for sensors to authenticate servers in sensor networks, 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 deployed near the AS will consume more energy to route messages for other sensors, and will thus exhaust their batteries before others. Therefore, one of our research goals is to eliminate the requirement of a centralized AS in sensor networks to achieve distributed authentication and make the system a truly distributed network environment.

- **Developing an Intrusion Detection and Accounting Framework:** Accounting is the tracking of system and network resource consumption and activity, and the recording of system failures to allow the verification of correct procedures and hosts based on the accounting data. Intrusion detection systems (IDSs) rely on the accounting data to detect the compromised nodes in the network. The traditional IDSs developed for wired networks are difficult to use for mobile ad hoc networks (MANETs) because of their architectural differences. Without centralized audit points like routers, switches, and gateways, MANETs can only collect audit data locally and thus require a distributed and cooperative IDS. Also, nodes in MANETs can move freely through the network, and thus their dynamically-changing network topology makes MANETs very different from the traditional wired networks. Therefore, one of our research goals is to develop an application-layer intrusion detection and accounting framework for nodes in MANETs to detect and prevent viruses, worms, and malicious applications by using the mobile agent technology to complement the IDS.

- **Enabling Trust and Authorization in Group Collaboration:** Authorization is the function to define the resource access rights, and guarantee access control to ensure data privacy in the system. For group collaboration, it is important to verify the identity of group collaborators and achieve trust among group members, and allow authorization and access control of resources within the group. The fact that mobile devices are often required to be small, light, and easy to hold places strict limitation on the devices' resources. Moreover, mobile devices form dynamic collaborative relationships because of their mobility. Thus, traditional trust management and authorization mechanisms will not be effective on mobile devices. So, we would like to design a distributed method to allow users of mobile devices to establish a trust relationship to collaborate and communicate with their collaborators, and to have access control over their shared information among their collaborators.

1.3 Overview and Contributions of the Dissertation

The main focuses of this dissertation are to achieve security in mobile and sensor networks with security protocols, and realize information sharing in mobile networks with collaboration protocols. The designed protocols are to allow secure distributed authentication, authorization, and accounting in distributed networks by applying cryptographic algorithms.

To meet the research goals mentioned in Section 1.2, we design and implement protocols that achieve distributed authentication, authorization, and accounting. The major contributions of this dissertation are summarized as follows.

- **Distributed Authentication of Program Integrity Verification in Wireless Sensor Networks:** Wireless sensors are usually deployed in hostile and unattended environments, and hence, are susceptible to various attacks, including tampering and manipulation of the sensor program. Preventing the sensors from communicating with attackers masquerading as legitimate nodes requires authentication to verify that the messages had really been sent by genuine nodes. We propose and develop DAPP, a distributed authentication protocol in wireless sensor networks to achieve the authentication of servers in a distributed manner without requiring a dedicated and trusted authentication server (AS). DAPP maintains the distributed nature of sensor networks, and reduces the sensor communication traffic in the network and the energy consumption on each sensor as compared to the case of using a centralized trusted AS for authentication. We also show that DAPP is robust and secure against various attacks in sensor networks.

- **Application-Layer Intrusion Detection in MANETs:** In a network of mobile devices connected by wireless links, node mobility makes it very challenging to secure mobile ad hoc networks (MANETs). Moreover, their constantly-changing topology causes network node density and neighbor relationships to change dynamically. To detect intrusions in MANETs, we present an intrusion detection system (IDS) framework for MANETs at the application layer. The IDS framework can utilize (1) both anomaly and misuse detection schemes to identify attacks in MANETs and (2) mobile agents (MAs) to augment each node's intrusion-detection capability. In particular, each node is equipped with a local IDS, and MAs will be dispatched periodically or on-demand to augment each node's IDS. We present the design of this IDS framework and the overall network structure, as well as the methods for authenticating and dispatching MAs. We also evaluate the trade-offs between different design parameters of MAs.

- **TGIS: Booting Trust for Secure Information Sharing in Dynamic Group Collaborations:** The various types of mobile devices and services that are becoming available introduce many collaboration opportunities for people using their mobile devices. We explore dynamic group collaboration and information sharing with mobile devices, such as smartphones and tablets. In particular, for secure information sharing among mobile devices, we propose trusted group-based information sharing (TGIS). TGIS is a protocol for mobile devices to establish trust relationships in order to form group-based information sharing. We exploit existing (group or organizational) identity hierarchies of mobile users to establish trust between group members with hierarchical identity-based encryption (HIBE). In order to control information sharing within a group and among groups, we further leverage attribute-based encryption (ABE) for secure access control, where attribute secret keys are distributed with the trust relationship with HIBE. We have implemented and evaluated TGIS on Android phones, demonstrating its viability.

1.4 Organization of the Dissertation

The remainder of the dissertation is organized as follows. Chapter II describes DAPP, a distributed authentication protocol without requiring the commonly-used centralized Authentication Server (AS) for server authentication in sensor networks. Chapter III presents an intrusion detection system (IDS) framework for mobile ad hoc networks (MANETs) at the application layer utilizing both anomaly and misuse detection schemes. Chapter IV presents TGIS, a protocol for mobile devices to establish trust relationship in order to form group-based information sharing with secure access control. Chapter V concludes this dissertation.

CHAPTER II

Distributed Authentication of Program Integrity Verification in Wireless Sensor Networks

2.1 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 inter-node communications. In sensor and ad-hoc networks, it is even easier for attackers to circumvent the underlying intrusion detection system since a malicious user can join the network at one point, hide inside the network for a while, then mount attacks. If the attacker was detected and blocked from joining the network, he may just disconnect from the network, change his personal identification, and then re-join 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, earthquakes, 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.

In order to protect the sensors from physical attacks, including physical tampering and manipulation of the sensor programs, Park and Shin [39] proposed a soft tamper-proofing scheme that verifies the integrity of the program in each sensor device, called the *Program Integrity Verification* (PIV). Seshadri *et al.* also proposed a software-based memory attestation technique called SWATT [46] and Secure Code Update By Attestation (SCUBA) [45]. 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 only combat 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. Upon 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 [39], the sensors rely on PIV Servers (PIVSs) to verify the integrity of the sensor programs. The sensors authenticate PIV Servers (PIVSs) with centralized and trusted third-party entities, such as authentication servers (ASs), in the network. This chapter mainly addresses the first line of defense. Specifically, we focus on the authentication of PIVSs instead of the verification of sensors.

In this chapter, 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 [39]. 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.

DAPP defends against both passive attacks such as eavesdropping and active attacks such as replay, spoof, drop, or insert false data. It also defends against impersonation attacks by using encryption and requiring pairwise keys between communicating nodes. Furthermore, DAPP can also tolerate compromised PIVSs via a revocation mechanism for compromised PIVSs and using the majority rule to determine PIVS authenticity.

DAPP offers the following attractive properties:

- *Distributed authentication guarantee:* DAPP allows PIVSs be authenticated in a distributed manner by neighbor PIVSs and guarantees the authentication result when there are no compromised PIVSs.

- *Resilient to node compromise:* Even if some PIVSs are compromised, a high probability exists that DAPP provides a true authentication result as DAPP uses the majority rule to determine PIVS authenticity.

- *Low computation overhead:* The computation overhead of DAPP mostly comes from the setup of pairwise keys and the generation/verification of MAC values, which are both energy cost-effective.

- *Low communication overhead:* The communication overhead of DAPP is associated with a PIVS's authentication requests to its neighbor PIVSs. Only two messages need to be transmitted for a PIVS to authenticate for another PIVS and thus the communication overhead is low.

- *Low storage overhead:* In DAPP, each sensor and PIVS only needs to store a polynomial for establishing pairwise keys with other nodes in the network, which takes very little memory space.

2.1.1 Organization

The remainder of this chapter is organized as follows. For completeness we first give an overview of the PIV protocol [39] in Section 2.2. We then present an overview of our design, including some of sensor device limitations, the motivation of this work, as well as the system model we use, the attack model we consider, and an overview of

DAPP in Section 2.3. Section 2.4 describes the details of DAPP, along with the PIVS revocation mechanism. Section 2.5 analyzes the security of DAPP and PIV and list some security issues in PIV, which is followed by our DAPP and PIV implementations in Section 2.6. Section 2.7 evaluates the performance of DAPP. Finally, we discuss the related work in Section 2.8 and conclude this chapter in Section 2.9.

2.2 Background: Overview of PIV

The PIV protocol [39] 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) [39] 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 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 2.1 depicts the interactions among the AS, the PIVS, and the sensor during PIV.

The main objective of PIV is to counter most of the physical attacks, i.e., 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

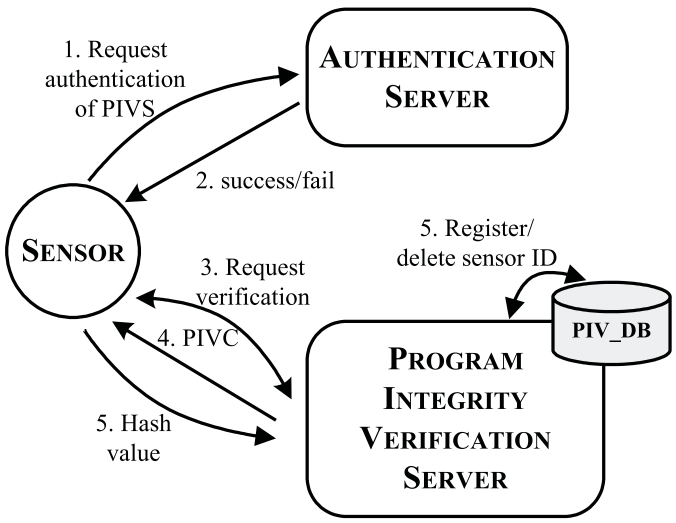

1. Request authentication of PIVS
AUTHENTICATION SERVER
2. success/fail
SENSOR
5. Register/ delete sensor ID
3. Request verification
PIV_DB
4. PIVC
PROGRAM INTEGRITY VERIFICATION SERVER
5. Hash value

Figure 2.1: Interactions among the AS, the PIVS, and the sensor during PIV.

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.

2.3 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, the attack model we consider, and give an overview of our DAPP design.

2.3.1 Sensor Network Architecture and Limitations

Sensor networks are often used for monitoring environments and information collection & 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 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 [19], 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 [16]. 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 communication is using a multi-channel 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.

2.3.2 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 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, since 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 the problem 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.

2.3.3 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 are 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 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.
- Since PIVSs have a longer transmission range than sensors, each PIVS is assumed to have at least t neighbor PIVSs after deployment.

2.3.4 Attack Model

Since the sensor nodes are small and resource-limited and usually deployed in public or hostile locations, they are vulnerable to physical capture and compromise. PIVSs are much more powerful servers in the network and can capably defend various attacks. The goal of the PIV protocol [39] is to defend the sensors from physical compromise. Our goal in this chapter is to allow the sensors to authenticate PIVSs in a fully-distributed manner. Below we list the attack models we consider, most of which are based on the lack of mutual authentication. We categorize the attacks into passive and active attacks.

- *Passive attacks*:
  - **Eavesdropping.** This is an obvious threat since wireless communication is broadcast-based. We assume the attackers can eavesdrop on all traffic in the network and that 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 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 node by replaying the messages it received, or by spoofing the messages it is supposed to send.

- **Sybil attack.** A particularly harmful attack against sensor networks is known as the Sybil attack [20], in which a Sybil node illegitimately fakes multiple identities in the network.
- **Impersonation attack.** This occurs when an attacker has the ability to impersonate a node. A node sends confidential information to the attacker rather than to the real recipients. This is typically the hardest attack to mount and defend against.
- **Man-in-the-middle attack.** An attacker is in-between the communication of two nodes and relays messages between them, making them believe that they are communicating directly. This attack resembles impersonation attack but the attacker controls the entire communication and must intercept all messages and inject new ones.
- **Collusion among compromised PIVSs.** When PIVSs are compromised after deployment, the attackers can uncover the encryption keys used and foster collusion among compromised PIVSs. The compromised PIVS can pass authentication with a sensor and send malicious code for the sensor to execute.

  Compromised PIVSs collusion can be categorized into the following scenarios:

  1. The compromised PIVSs need to have direct communication links in order to collude.
  2. The compromised PIVSs can collude through multiple hops and would require the knowledge of the location and ID of other compromised PIVSs.

In this chapter, we focus on countering the first collusion scenario above through collaboration among neighbor PIVSs. We discuss how DAPP

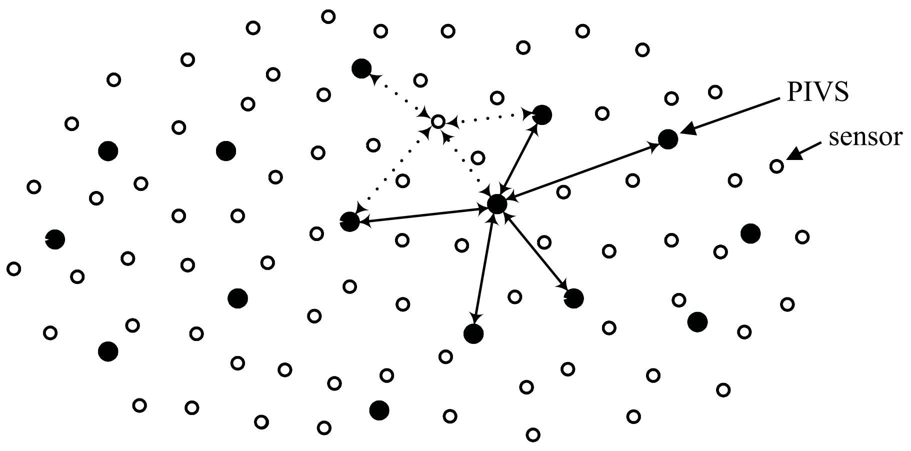


Figure 2.2: A design overview of DAPP. PIVSs interact with one another for mutual authentication. The solid lines represent the interactions between the PIVSs and the doted lines represent the interactions between the PIVSs and the sensor.

can be extended to counter against the second collusion scenario in Section 2.5.2.

2.3.5 Overview of DAPP

We exploit the PIV protocol [39] summarized in Section 2.2 to protect the integrity of sensor programs. In order 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 2.2, where the large circles represent PIVSs in the network, and the small ones represent sensors. 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 doted lines represent the interactions between the PIVSs and the sensor.

PIVSs need to share some secrets with one another in order for them to authenticate one another. In our proposed distributed authentication protocol of PIVSs (DAPP), these secrets are pairwise keys shared between PIVSs which are established by using the Blundo scheme [8]. 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 determined a PIVS to behave maliciously, the PIVS will be evicted from the network.

2.4 DAPP Details

In the original PIV design [39], 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 below are the notations used in the rest of the chapter.

- $A, B, \ldots$ 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 timestamp 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 [8] 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 $f(x, y)$ which is a secret only known 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 [59], 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.

2.4.1 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.

2.4.1.1 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 unique 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 [8] is used to set up pairwise keys. An offline 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 in order 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, i.e., $K_{A,j} = F(K_{A,j+1})$. $K_{A,0}$ is called PIVS A's revocation verification key. Since revocation keys are generated by a one-way function, they are forward-computable, but not backward-computable, i.e., 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 Section 2.4.3).

We require that no two PIVSs will 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.

2.4.1.2 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 post-deployment 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 pre-deployment initialization phase. Likewise, A can compute $f(A, B)$. Since $f(x, y)$ 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 discovery phase is:

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

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 the authenticity of each other with their shared pairwise key, they add the ID of each other 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.

2.4.2 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 [32]. The sensor will first authenticate the PIVS, and 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 Auth Message to A, which includes a randomly-generated nonce N_E and the MAC value of N_E computed using K_{AE}. Upon receiving the message, PIVS A will generate the shared key $K_{AE} = f(A, E)$, and verify the PIVS Auth Message sent by E. If the message is authentic, then PIVS A will send PIVS Ref Messages including sensor E's ID, nonce N_E, and the MAC value computed using the shared pairwise keys to the PIVSs on its PIVS Reference List. The PIVSs that receive the PIVS Ref Messages will check the authenticity of the message it received. To check the authenticity of a PIVS Ref Message, 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 since 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, and 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 in order to pass the authentication. Note that the reason for sensor E to include a randomly-generated nonce in the PIVS Auth Message is to prevent external attackers from recording the authentication tickets in the network and then reuse them.

The complete protocol of the PIVS authentication phase is summarized below.

$$\begin{aligned}
E \rightarrow A &: \text{PIVS Auth Message}(N_E, \text{MAC}(K_{AE}, N_E)) \\
A \rightarrow B &: \text{PIVS Ref Message}(E, N_E, \text{MAC}(K_{AB}, E|N_E)) \\
B \rightarrow A &: E, \text{AuthTicket}(B, \text{MAC}(K_{BE}, N_E)), \\
&\quad \text{MAC}(K_{AB}, E|\text{AuthTicket}(B, \text{MAC}(K_{BE}, N_E))) \\
&\quad \ldots\ldots \\
A \rightarrow E &: \text{AuthTicket}(B, \text{MAC}(K_{BE}, N_E))|\text{AuthTicket}(C, \text{MAC}(K_{CE}, N_E))|..., \\
&\quad \text{MAC}(K_{AE}, N_E + 1)
\end{aligned}$$

Using this same example, for PIVS B to authenticate another PIVS A, B needs to hold a pairwise key with A. Therefore, if B has received a PIVS Ref Message from A, it is on A's PIVS Reference List, and should share a pairwise key with A. When B grants A the authentication ticket, it will also include the MAC of sensor E's ID and the authentication ticket computed using A and B's pairwise key K_{AB}. If the MAC value of B's message matches with the MAC value of B's message computed by

A using their share 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 neighbor PIVSs. If we consider Byzantine failures in the network, then $t \geq N_{\text{auth}} \geq 3d + 1$, where t is the minimum number of neighbor PIVSs that each PIVS has after deployment. If we do not consider Byzantine failures, then $t \geq N_{\text{auth}} \geq 2d + 1$. 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, then sensor E will check the correctness of the authentication tickets and use a simple majority rule to determine A's authenticity. If more than $N_{\text{auth}}/2$ 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 2.3 shows the interactions among PIVS A, PIVS A's reference PIVS B, and sensor E during DAPP authentication phase. The messages exchanged are (1) Sensor E sent PIVS Auth Message to PIVS A, which (2) then sent PIVS Ref Messages to PIVSs on its PIVS Reference List. Next (3) PIVS A's reference PIVSs then sent AuthTickets back to A, which (4) collected N_{auth} AuthTickets from its reference PIVSs and forwarded the tickets to E for authentication. Figure 2.4 is the

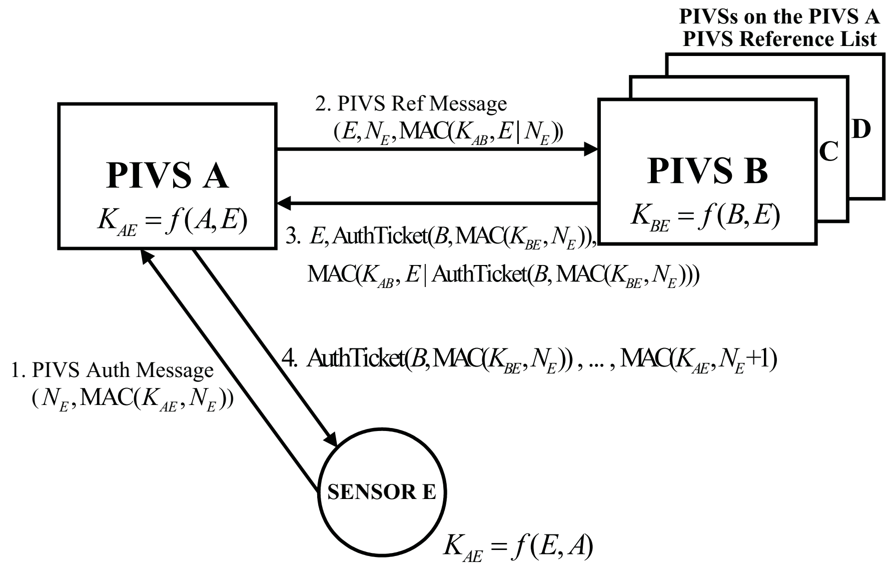

PIVSs on the PIVS A
PIVS Reference List
2. PIVS Ref Message
$(E, N_E, \mathrm{MAC}(K_{AB}, E|N_E))$
PIVS A
$K_{AE} = f(A,E)$
PIVS B
$K_{BE} = f(B,E)$
C
D
3. E, AuthTicket(B, MAC(K_{BE}, N_E)), MAC(K_{AB}, E| AuthTicket(B, MAC(K_{BE}, N_E)))
4. AuthTicket(B, MAC(K_{BE}, N_E)) , ... , MAC(K_{AE}, N_E+1)
1. PIVS Auth Message
$(N_E, \mathrm{MAC}(K_{AE}, N_E))$
SENSOR E
$K_{AE} = f(E,A)$

Figure 2.3: Interactions among PIVS A, PIVS A's reference PIVS B, and sensor E under DAPP.

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 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.

2.4.3 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, i.e., 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. Since 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

```
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;
}

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)];
}

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)))];
}
```

Figure 2.4: DAPP pseudocode to be executed on sensor E, PIVS A, and PIVS B.

neighbor PIVSs can send others PIVS 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 chapter. 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 utilize 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 pre-specified number of times within a time interval of interest, then it will use μTESLA [40] 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 [40], 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.

To use μTESLA, all PIVSs in the network are loosely time-synchronized [22, 51], 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 in Section 2.4.1.1. For PIVS A, after key $K_{A,j}$ is used in a PIVS Revocation Message, it will issue the next PIVS Revocation Message with $K_{A,j+1}$ to compute the MAC value.

If PIVS A detects the malicious behavior of 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:

$$A \to * : T_A, B, j, \mathrm{MAC}(K_{A,j}, T_A|B|j).$$

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 4-th 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 PIVSs to be revoked by colluded neighbors.

2.5 Security Analysis

We first discuss network survivability in the event of sensor and PIVS compromises, and next discuss how DAPP can be extended to counter multiple hops compromised PIVS collusion. We then mathematically analyze DAPP to show the probability of PIVS true authentication, after which we analyze the security of DAPP and PIV against various attacks. We finally identify some possible attacks on the PIV protocol, along with appropriate countermeasures.

2.5.1 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 re-verify its program using PIV. The sensor will be excluded from the network if it fails to pass PIV.

When the compromise of the sensors and PIVSs go undetected, we need to analyze the survivability of the network, or its ability to maintain an acceptable level of performance given compromised nodes. For this, we will consider the general attacks an adversary can mount after compromising a node.

Since each node in the network is only preloaded 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 impersonate that node. We employ the Blundo scheme [8] for the pairwise key function, which is proven 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 *et al.* [60], 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, since each PIVS needs to receive N_{revoke} PIVS Revocation Messages before revoking a PIVS, an attacker will need to compromise many PIVSs before revoking a PIVS is possible. 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 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.

2.5.2 Compromised PIVSs Collusion

As stated in Section 2.3.4, there are two scenarios of compromised PIVSs collusion: (1) for neighbor PIVSs with direct communication links to collude, and (2) for all PIVSs in the network with multiple hops to collude. In this chapter, we focus on defending against collusion scenario (1) since it is easier for PIVSs to collude with theirs neighbors as they know the locations and IDs of one another and they share direct communication links. It is harder for PIVSs to collude when they are multiple hops away since they need to deal with intermediate PIVSs in their communication path or require setting up separate channels for compromised PIVSs to communicate in order to evade intrusion detection.

If any two compromised PIVSs in the network collude, 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. Previously we only consider any two compromised neighbor PIVSs collude, but DAPP can also be extended to counter against the collusion of any compromised PIVSs in the network.

In Section 2.4.2, we state that from the N_{auth} authentication tickets issued by a PISV's reference PIVSs, the authentication result must mask the effects of d or fewer malicious neighbor PIVSs. To consider multiple hop compromised PIVSs collusion, the authentication result must mask the effects of d or fewer malicious PIVSs in the entire network. Nevertheless, the guideline for choosing N_{auth} is still the same as before. If we consider Byzantine failures in the network, then $t \geq N_{\text{auth}} \geq 3d + 1$, where t is the minimum number of neighbor PIVSs that each PIVS has after deployment. If we do not consider Byzantine failures, then $t \geq N_{\text{auth}} \geq 2d + 1$. However, considering compromised PIVSs in the entire network can collude, N_{auth} needs to be large enough to mask the effects of d or fewer malicious PIVSs. It is not practical if d is large since t, the minimum number of neighbor PIVSs that each PIVS has after deployment, will not be large enough for setting the required N_{auth} value.

DAPP can defend against the multiple hop compromised PIVSs collusion 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 the entire topology of the network, it knows the network topology in its proximity. DAPP can be extended to allow sensors to choose the referencing PIVSs for a PIVS to show authentication tickets from. Thus, one PIVS has to show authentication tickets from the sensor assigned reference PIVSs and it might not have control over the reference PIVSs the sensor chooses. By having a sensor sending a list of PIVSs that it wants to have authentication tickets from, this can force the compromised PIVS to fail the authentication.

Depending on the deployment strategy, on average, a sensor should overhear about half of a PIVS's neighbor PIVSs. Moreover, if sensors are equipped with directional radios, then sensors can choose PIVSs from different directions to avoid compromised PIVSs collude to launch Sybil attack and claim to be multiple identities.

Our analysis below focus on defending against compromised PIVSs colluding through direct communication links.

2.5.3 Probability of PIVS True Authentication

As stated in Section 2.3.3, we will now consider a network that consists of s PIVSs that are evenly distributed and in which each PIVS has at least t neighbor PIVSs after deployment. The pairwise key function used for the PIVSs and sensors to establish pairwise keys is a symmetric bivariate k-degree polynomial. We assume the attackers compromise N_c servers that are evenly distributed in the network and $N_c \leq k$.

For a PIVS to claim authenticity, it needs to collect N_{auth} authentication tickets from its neighbor PIVSs. The authentication result must mask the effect of d or fewer malicious neighbor PIVSs. If we consider Byzantine failures in the network, then we choose $t \geq N_{\text{auth}} \geq 3d + 1$. However, if we do not consider Byzantine failures, then $t \geq N_{\text{auth}} \geq 2d + 1$. In reality, we would not know how many neighbor PIVSs are malicious; therefore, it is not possible to determine N_{auth} based on d. Below we show how to estimate the probability of DAPP true authentication and how to determine N_{auth} from it.

The probability of a compromised PIVS is $p = \frac{N_c}{s}$. Among the N_{auth} neighbors that provides authentication tickets, the probability that a certain group of i neighbors being compromised is

$$\mathrm{P}[i \text{ compromised neighbors}] = \binom{N_{\text{auth}}}{i} p^i (1-p)^{N_{\text{auth}}-i}.$$

Note that this is a Binomial Distribution with success probability p for i out of N_{auth} successful events.

If we consider Byzantine failures in the network, then for the collaborating neighbor PIVSs to provide a correct authentication result, $i \leq h := \lfloor \frac{N_{\text{auth}}-1}{3} \rfloor$. However, if we do not consider Byzantine failures, then $i \leq h := \lfloor \frac{N_{\text{auth}}-1}{2} \rfloor$.

Consequently, the probability of PIVS true authentication, or of any PIVS to have a correct authentication result, can be estimated by

$$P_{\text{correct}} = \sum_{i=0}^{h} \mathrm{P}[i \text{ compromised neighbors}] = \sum_{i=0}^{h} \binom{N_{\text{auth}}}{i} p^i (1-p)^{N_{\text{auth}}-i}.$$

For the discussion below, we assume a network without Byzantine failures. Assume there are 250 PIVSs in the network, that is, $s = 250$, and PIVSs are required to provide 5 authentication tickets to sensors in the PIVS Authentication Phase, that is, $N_{\text{auth}} = 5$. We have shown in Figure 2.5 the relationship between the probability of PIVS true authentication P_{correct} and the number of compromised PIVSs N_c. As shown in Figure 2.5, as the number of compromised PIVSs increased in the network, the probability of PIVS true authentication decreased. However, for a network with 80 compromised PIVSs out of 250 total PIVSs, the probability of getting a PIVS true authentication with DAPP is still around 80%.

Now consider a network with 250 PIVSs and 90 compromised PIVSs, that is, $s = 250$ and $N_c = 90$. Then the probability of a compromised PIVS is $p = \frac{N_c}{s} = \frac{90}{250} = 0.36$. Figure 2.6 shows the relationship between the probability of PIVS true authentication P_{correct} and the number of required authentication tickets from a PIVS Nauth. As shown in Figure 2.6, if N_{auth} is chosen to be 5, then the probability of getting

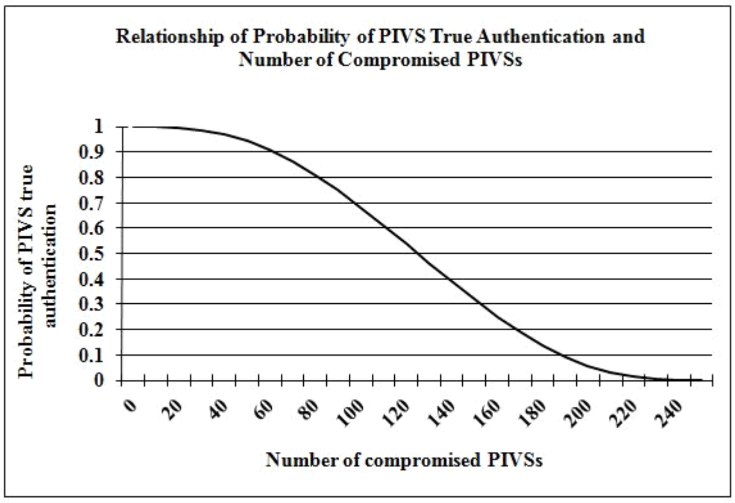


Figure 2.5: Relationship between the probability of PIVS true authentication and the number of compromised PIVSs.

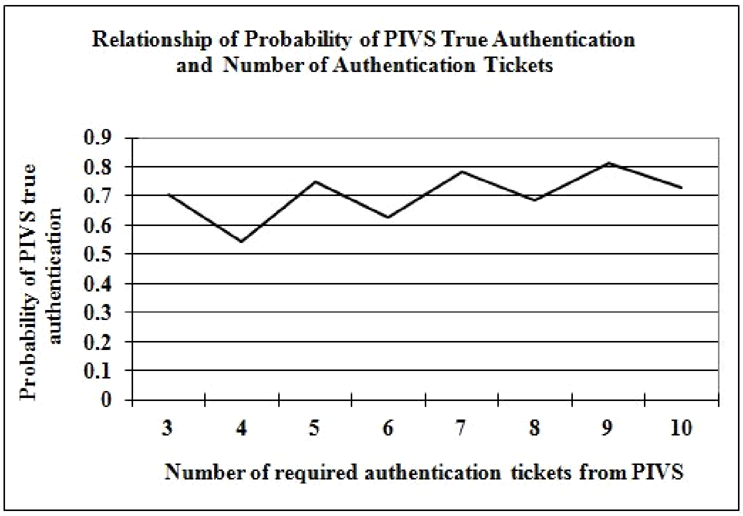


Figure 2.6: Relationship between the probability of PIVS true authentication and the number of required authentication tickets from PIVS.

a PIVS true authentication is around 75%. Therefore, N_{auth} can be determined by the desired P_{correct} for the protocol. Note that the graph in Figure 2.6 zigzags because we don't consider Byzantine failures in the network and thus we must choose more than $h = \lfloor \frac{N_{\text{auth}}-1}{2} \rfloor$ authentication tickets out of N_{auth} tickets for the majority authentication result.

2.5.4 Defense Against Various Attacks in Sensor Networks

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

2.5.4.1 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 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.

2.5.4.2 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.

Sybil attacks [20] are particularly harmful in sensor networks. A Sybil node illegitimately fakes to have multiple identities in the network, but DAPP intrinsically withstands such attacks. It is not possible for the attacker to launch Sybil attacks against DAPP since each node will need to have a pairwise key with its communicating node to authenticate its identity. Since each node will have a preloaded pairwise key function for establishing pairwise keys, no nodes can generate the pairwise keys and pretend to be another node without knowing the function.

Impersonation and man-in-the-middle attacks both require the attacker to fake a claimed identity. As with how DAPP withstands Sybil attacks, since each node needs to have a pairwise key with its communicating node to authenticate its identity, it will be impossible for an attacker to fake a claimed identity without knowing the pairwise key function to impersonate the node.

In DAPP, for a PIVS to pass by a sensor's authentication, it needs to present N_{auth} authentication tickets. If a PIVS is detected to have been compromised and is subsequently revoked by its neighbor PIVSs, they will not issue authentication tickets for the compromised PIVS; therefore, it will not be able to pass authentication. However, if the reference PIVSs are compromised and send false authentication tickets to cause the authenticating PIVS to fail authentication, then a majority of the reference PIVSs must be compromised and in collusion. Otherwise DAPP will mask the effect of d or fewer compromised PIVSs by selecting N_{auth} based on the network failure model as described in Section 2.4.2. Or, if the reference PIVS is detected to be compromised and revoked, then the authenticating PIVS will not ask the compromised PIVS to authenticate for it.

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. Since 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.

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 periodically re-verify their programs.

2.5.5 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.

2.5.5.1 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.

2.5.5.2 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 2.7, along with the flash memory layout of different program components 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

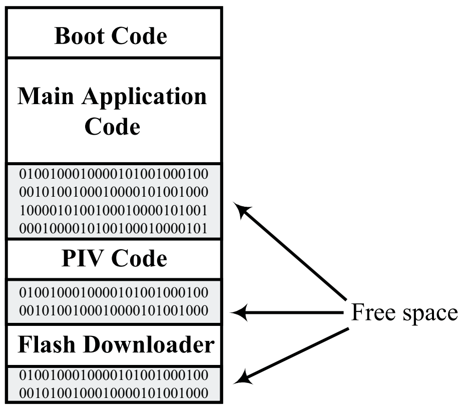


Figure 2.7: The sensor flash memory layout in PIV.

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.

2.5.5.3 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. Since SRAM, EEPROM, and the additional data flash memory are much slower than the flash and thus 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 *et al.* [48], 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.

2.5.5.4 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, then it can 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 use 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 it to pass their verification.

The tight hash-time interval mentioned above can also be used to handle this attack. Since 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.

2.6 Implementation of DAPP and PIV

This section describes our implementation of the DAPP and the PIV protocol [39] on Mica2 Motes [16] 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 [23] embedded with assembly to write the Boot code, C embedded with assembly to write the mobile agent PIV Code (PIVC), and assembly to write the flash downloader.

2.6.1 Changes to the Original PIV Design

We made some modifications to the original PIV designed by Park and Shin [39], which are listed with their justification.

2.6.1.1 Hash functions: RHF vs. HMAC-MD5

The authors of PIV [39] 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, i.e., one from the program in the sensor and the other from the digest of the sensor program stored in the PIVS. However, since the free space in the flash of a sensor can be used by an attacker to

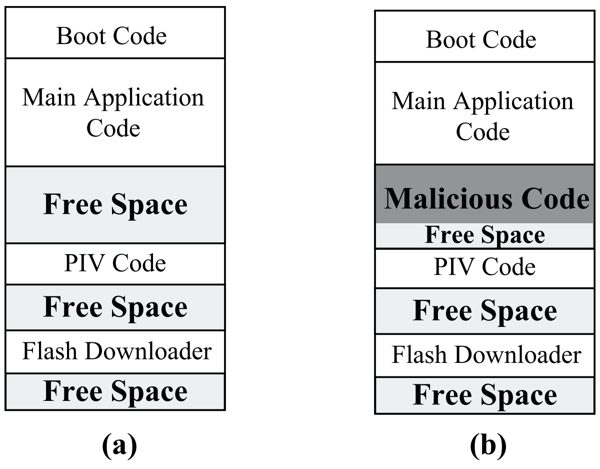


Figure 2.8: (a) Normal sensor flash memory layout; (b) Sensor flash memory layout with malicious code hidden in the free space.

hide malicious codes, as shown in Figure 2.8 and described in Section 2.5.5.2 as the flash free-space compression attack, we need to fill up 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 [39] 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, $X_l = \mathbf{x}_l \mathbf{x}_l^T = (\, x_{l,i}\, x_{l,j}\,)$. 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 since 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.

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

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, and thus, there was little advantage of using RHFs for hash computation. Therefore, we decided to store sensor programs instead of digests in PIVSs, and not use RHFs for hash computation.

Instead, we decided to use HMAC [33], a mechanism for message authentication using cryptographic hash functions, together with an iterative cryptographic hash function MD5 [41], in combination with a secret shared key. The reason for using HMAC for verification of sensor programs is that HMAC can be used in combination with any iterated cryptographic hash functions, such as MD5 [41] or SHA1 [21]. 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 [41] is a widely-used cryptographic hash function, and even though it has been shown to be vulnerable to hash collisions [53], 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.

2.6.1.2 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, since 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. Since 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 only need to request 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 [33, 41] computation.

Since the PIVC will not change very often, the PIVS will only transmit 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.

2.6.2 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 [33, 41] 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 [33]. CBC-MAC (Cipher Block Chaining Message Authentication Code) utilizes 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 [17], 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 *et al.* [11] 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.

2.6.3 Overview of the Implementation

Figure 2.9 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.

2.6.3.1 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

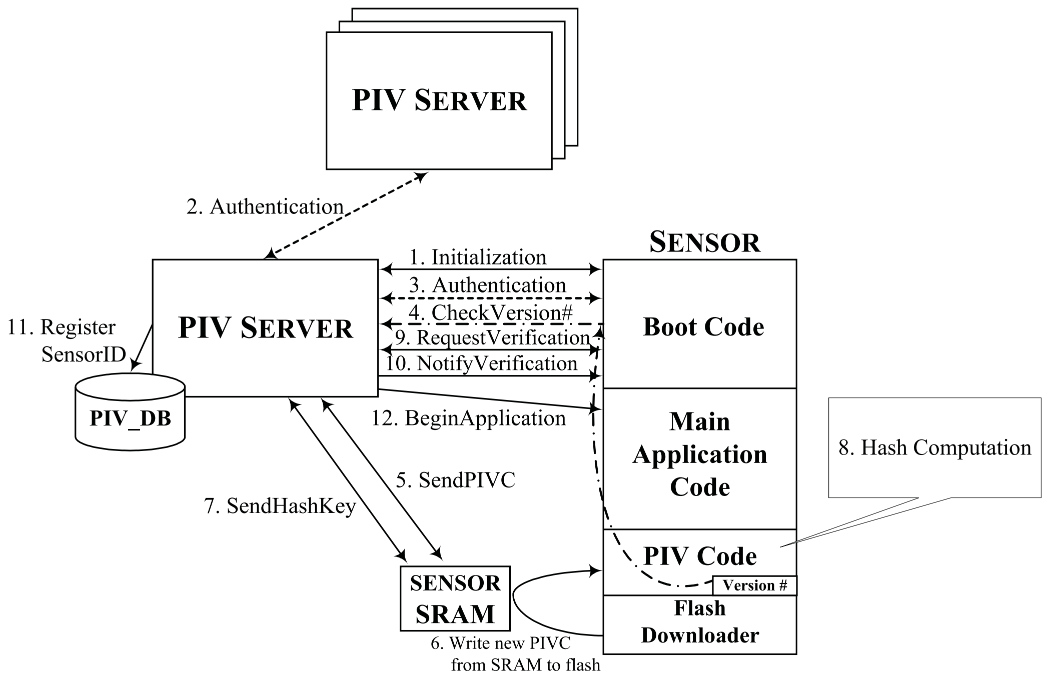


Figure 2.9: Overview of our implementation of DAPP and PIV.

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.

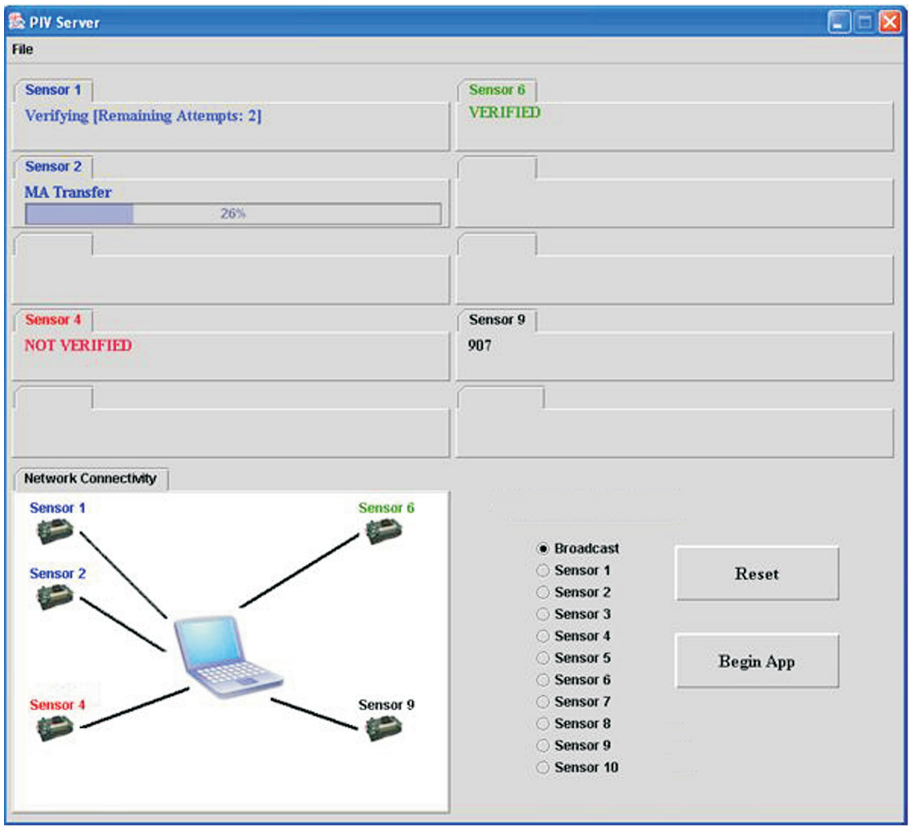
The PIVS is written in Java, and we implemented HMAC-MD5 [33, 41] to generate MAC when PIVSs run DAPP. PIVSs use the shared pairwise key between the two PIVSs to generate and verify MACs. The pairwise key length is 16 bytes and the MAC length is 10 bytes. The PIVS calls a C program to compute HMAC_MD5 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 send it over the radio to the sensor.

The PIVS takes care of sensors' requests for authentication, requests for update of a mobile agent PIVC, requests for the hash key, requests for verification, and requests for checking the verified sensors in its PIV_DB. The interactions between PIVSs and a sensor are shown in Figure 2.9 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.

Upon 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 re-generates 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 2.10. It shows the current status of the sensors interacting with the PIVS as well as the connectivity of the sensors to the PIVS. In particular, 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. The PIVS can choose to broadcast or unicast to a particular sensor and to reset sensors or to start the sensors' main applications.



Sensor 1
Verifying [Remaining Attempts: 2]

Sensor 6
VERIFIED

Sensor 2
MA Transfer
26%

Sensor 4
NOT VERIFIED

Sensor 9
907

[✓ Broadcast] Sensor 1 Sensor 2 Sensor 3 Sensor 4 Sensor 5 Sensor 6 Sensor 7 Sensor 8 Sensor 9 Sensor 10

Figure 2.10: A snapshot of the PIVS interface.

2.6.3.2 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 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 [23], 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 4 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. The reason why we didn't use network programming for PIVC transmission and update is 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 to PIVS for verification. Upon receiving the verification result from the PIVS, the Boot code will either activate the main application code on the sensor if the sensor passes the verification, or 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.

Since 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. Upon 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.

2.6.3.3 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 5 minutes to transmit the entire PIVC from a PIVS to a 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, since 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, and 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.

Since 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

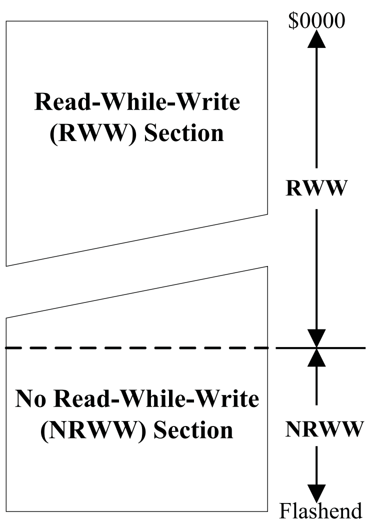

$0000
Read-While-Write (RWW) Section
RWW
No Read-While-Write (NRWW) Section
NRWW
Flashend

Figure 2.11: Read-While-Write section and No Read-While-Write section in the sensor flash.

as passing the PIVC version number from the PIVC to the Boot code.

2.6.3.4 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 [4]. Figure 2.11 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 in a page-by-page fashion. 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 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.

2.7 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.

2.7.1 Evaluation with Simulation

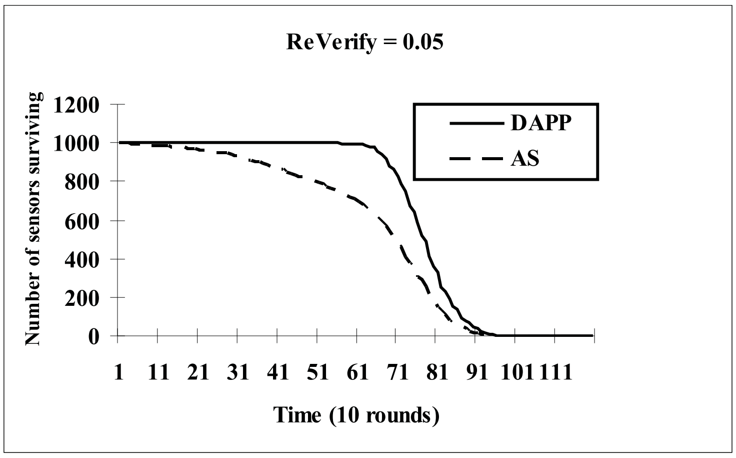
We simulated DAPP with randomly-generated networks consisting of 1000 sensors and 250 PIVSs in a 1000×1000 unit area. We assume each sensor has a communication range of 150 units, and that a PIVS normally communicates with others within 200 units from itself. Each sensor node is initialized with 0.5 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 a single authentication server (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 only needs to communicate with the authenticating PIVS. In our DAPP simulation, we chose $N_{\text{auth}} = 5$, i.e., a PIVS needs to present 5 authentication tickets to a sensor to pass authentication. In our AS simulation, the AS is placed at $(x = 1070, y = 1070)$. 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.

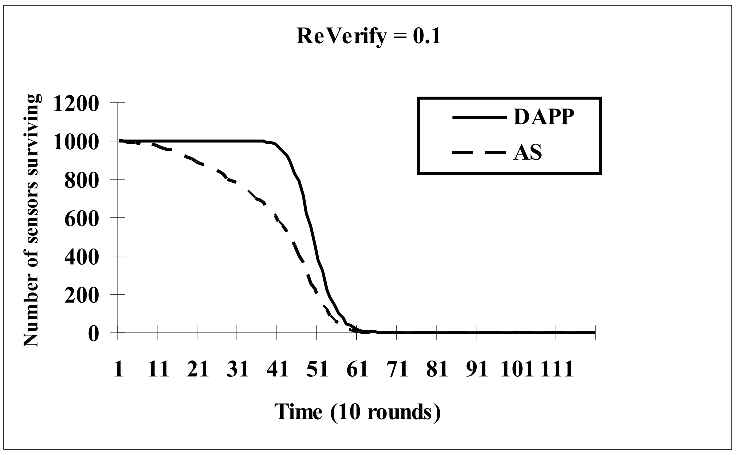
The simulation time is divided into rounds of actions. In each round, each sensor has the probability $ReVerify$ that it needs to re-verify its program with a PIVS and thus re-authenticate the PIVS. We performed simulation while changing $ReVerify$ and plotted the results in Figure 2.12, showing the 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 have a longer life time with DAPP than 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 re-verified more often.

Figure 2.12 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. When using DAPP for authentication, inter-PIVS communications will not interfere with sensor communications since 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 2 messages exchanged between the sensor and the PIVS. Since there are 1,000 sensors in the network, at



(A)



(B)

Figure 2.12: Numbers of sensors that survive with DAPP and with an AS for authentication, with (A) $ReVerify = 0.05$ and, (B) $ReVerify = 0.1$.

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 using a single AS placed at $(x = 1070, y = 1070)$ for authentication.

Using DAPP for authentication also reduces the energy consumption on each sensor. With DAPP for authentication, one sensor dissipates, on average, $1,114\mu$J to authenticate one PIVS. With a single AS for authentication, one sensor dissipates, on average, $7,624.5\mu$J to authenticate one PIVS. The DAPP's energy consumption on each sensor improves up to 85% over using an AS placed at $(x = 1070, y = 1070)$ 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 placed at $(x = 1070, y = 1070)$.

If the AS is placed in the middle of the network, at $(x = 500, y = 500)$, and when the AS is used in the network for authentication, then sensors exchange an average of 10,253 messages in the network by counting the messages relayed by other sensors as separate messages. Moreover, one sensor dissipates, on average, $3,290.8\mu$J to authenticate one PIVS. In this case, DAPP reduces the sensor communication traffic in the network by more than 80%, and improves the energy consumption on each sensor up to 65%.

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

We further simulate with multiple ASs placed in the network, in particular with four ASs placed at $(x = 750, y = 250)$, $(x = 250, y = 250)$, $(x = 250, y = 750)$, and $(x = 750, y = 750)$ respectively. When these ASs are used in the network for authentication at their respective quadrants, the sensors exchange an average of 6,276 messages in the network by counting the messages relayed by other sensors as separate messages. Moreover, one sensor dissipates, on average, $1,914\mu$J to authenticate one PIVS. In this case, DAPP reduces the sensor communication traffic in the network by more than 68%, and improves the energy consumption on each sensor up to 40%. This simulation shows even when deploying more ASs in the network, DAPP still has less communication and computation overhead compared to using ASs to authenticate PIVSs.

2.7.2 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.

2.7.2.1 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 [8]. Each node needs to compute k modulo multiplications and k modulo additions for a k-degree polynomial in order to generate a pairwise key. As stated by Zhu *et al.* [60], 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 *et al.* [35] 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 *et al.* [52] 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.

2.7.2.2 MAC Generation and Verification

We used HMAC [33] 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 *et al.* [42], HMAC consumes approximately 45.6 μJ if it runs on a Mote. Carman *et al.* [12] 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.

2.7.3 Communication Cost

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

2.7.4 Storage Requirement

Here we show that the memory requirement for sensors and PIVSs to store the pairwise keys are very small. Each sensor and PIVS needs to store a k-degree polynomial over a finite field F_q 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 $N_{\text{auth}} + 1$ keys before it can authenticate a PIVS. Since each key is 128 bits (or 16 bytes) long, if we choose $N_{\text{auth}} = 5$, then a sensor only needs to store 6 keys, and a total of 96 bytes suffices. Therefore, keys require only 2.3% of the sensor SRAM since a Mica2 Mote has 4K bytes of SRAM. Similarly, PIVSs use the same polynomial to establish pairwise keys, and since PIVSs are equipped with more memory than sensors, they can store more keys than sensors.

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

2.8 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 [8] is used as their design basis.

Weimerskirch and Thonet [54] 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 [1] 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 [18] in which as long as the number of compromised nodes is below a given threshold, the compromised nodes cannot harm the network operation.

Hubaux *et al.* [29] 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 [6] 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 *et al.* [44, 13] proposed secure, efficient and non-interactive 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 bi-variate 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 [39] which verifies the integrity of the program and data stored in a sensor device. SWATT [46] 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) [45] 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 *et al.* [48] 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 [8]. Liu *et al.* [35] used the Blundo scheme as a basis in their proposed scheme for establishing pairwise keys in distributed sensor networks. Zhu *et al.* [60] 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 multi-hop pairwise keys. Zhang *et al.* [57] 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.

2.9 Conclusion

In this chapter, we presented a distributed authentication protocol of PIVSs (DAPP) for sensors to authenticate PIVSs in sensor networks, and implemented DAPP and the PIV protocol [39] 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.

We have also demonstrated that DAPP is robust and secure against various attacks in sensor networks and offers many attractive properties including distributed authentication guarantee, resilience to node compromise, and low overhead in computation, communication, and storage.

Our main contribution in this chapter 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 [39]. DAPP maintains the distributed nature of sensor networks, but also reduces the sensor communication traffic in the network and the energy consumption on each sensor compared to the case of using a centralized trusted AS for authentication.

CHAPTER III

Application-Layer Intrusion Detection in MANETs

3.1 Introduction

A mobile ad hoc network (MANET) is built with mobile nodes which communicate via wireless radio links. Instead of using a central base station for nodes to communicate with one another, MANETs do not rely on any pre-defined infrastructure and operate in peer-to-peer mode. Nodes within the communication range communicate via wireless radio links, and for those outside the communication range, use other nodes to relay their packets. Mobile nodes may move away from their current locations and re-join the network from different locations in the network, thus dynamically changing their network topology and node density. MANETs are being developed and deployed for many mission- and life-critical applications such as military tactical operations (e.g., future combat system (FCS)), emergency search-and-rescue missions, and mobile tele-conferencing. However, the dynamically-changing topology of MANETs make them vulnerable to various attacks.

In recent years, security has become a primary concern to the communications in MANETs. Unlike wired networks, security in wireless networks is difficult to achieve due to the broadcast nature of inter-node communications. In MANETs, it is even more difficult to achieve security because of node mobility and constantly-changing group membership. Intrusion prevention is not guaranteed to work all time either, and can only combat outsider attacks. Therefore, we need a strong intrusion detection system (IDS) which plays a critical role in securing MANETs. An IDS can discover malicious activities or insider attacks mounted by compromised nodes in the network. The IDS then tries to prevent intrusions that compromise system security, and upon detection of an intrusion, it tries to recover from the damages inflicted by the intrusion. Considering continuous discovery of new vulnerabilities, the IDS must be effective and efficient in identifying attacks, and then neutralizing them.

The traditional IDSs developed for wired networks are difficult to use for MANETs because of their architectural differences. Without centralized audit points like routers, switches, and gateways, MANETs can only collect audit data locally and thus require a distributed and cooperative IDS. Other differences between wired networks and MANETs include traffic patterns, node mobility, and node constraints. These differences all render the traditional IDSs hard to be directly applied to MANETs. Nodes in MANETs can move freely through the network, and thus their dynamically-changing network topology makes MANETs very different from the traditional wired networks. Also, nodes in MANETs usually have slower communication links, limited bandwidth, limited battery power, and limited memory. Therefore, these constraints make the design of IDS in MANETs much more challenging than in wired networks.

Due to the dynamically-changing topology of MANETs, neighbor relationships and node density vary with time. For MANETs with high node mobility, it is very difficult to design an IDS that is distributed and light-weighted, and consists of cooperative nodes in physical proximity. To meet this challenge, we propose an MA-based application-layer IDS framework for MANETs. It utilizes both anomaly and misuse detection to identify attacks and also utilizes MAs to augment each node's intrusion detection capability. Our goal is to detect and prevent viruses, worms, and malicious applications on each node by using the MA technology to complement the IDS.

The main contribution of this chapter is the use of MAs to augment the application-layer IDS in MANETs which is a significant departure from most existing IDSs in MANETs that target the network layer. Our application-layer IDS uses system-call sequences to detect intrusions on each node. It also uses MAs to augment the IDS by updating attack signatures and normal application profiles, and patching and installing (new) programs on each node. MAs can also augment the detection capability by being dispatched for further analysis and diagnosis on network nodes when an anomaly is detected. Finally, MAs can be dispatched to verify the correctness of IDS agents. Another contribution of this chapter is the mechanism for dispatching MAs to network nodes for update, analysis, and verification.

3.1.1 Organization

The remainder of this chapter is organized as follows. We first list the advantages of using MAs in IDSs in Section 3.2. The system architecture is then presented in Section 3.3 along with the assumptions and the attack model used in this chapter. Section 3.4 details the design of our MA-based IDS, and the use of MAs in MANETs. Section 3.5 analyzes the security of the proposed IDS while Section 3.6 evaluates its performance. We then discuss the related work in Section 3.7. Finally, this chapter concludes with Section 3.8.

3.2 Why are MAs Needed for Intrusion Detection?

Since the mobile nodes in MANETs are energy-constrained, we need to design protocols that are lightweight and energy-efficient. MAs offer many advantages [30] when used in an IDS, and will help overcome the difficulty of building distributed systems and protocols. Below we list and describe the advantages of using MAs for intrusion detection.

Reducing Network Load MAs transfer the computation and detection function to the network nodes with audit data instead of transmitting large amounts of audit data to the servers for computation and detection, thus reducing the network load.

Overcoming Network Latency MAs can be dispatched from the servers to network nodes to detect malware and take corrective actions in real time. The MAs can operate directly on the nodes and respond faster to a potential intrusion than communicating with the servers for assistance.

Making the IDS Attack-Resistant MAs can be used in the IDS to avoid single-point-of-failures. The time of an MA's arrival at each node, the reporting mechanism, and the detection algorithm the MA uses are made unpredictable so that attackers may not know this information.

Autonomous Execution MAs can continue to function even when portions of the IDS or the network get destroyed or malfunction. MAs can increase the IDS's fault-tolerance by operating independently of the platform.

Dynamic Adaptation MAs have the ability to sense the execution environment and react to changes. Also, MAs can adapt to the environment as they can be retracted, dispatched, or put to sleep as the network and host conditions change.

Platform Independence MAs can operate in heterogeneous environments by having a virtual machine or interpreter on the host platform. This capability makes a perfect fit for MANETs as nodes in the network typically are comprised of many different computing platforms.

Upgradability MAs can perform program updates, and anomaly and misuse detection on each node. MAs can carry the most up-to-date program patches, normal application profiles, and attack signatures to the nodes for upgrade while the IDS keeps working on each node.

Scalability MAs help distribute the computational load to different nodes in the network instead of having all the computation processed on the servers, and reduce the network load. This advantage enhances scalability and makes the IDS more fault-resistant.

3.3 Assumptions, Attack Model, and System Architecture

We first state the assumptions we use, then present the attack model, and finally, describe the system architecture of our IDS design.

3.3.1 Assumptions

A1. A MANET is composed of a large number of mobile nodes and one secure stationary MA server.

A2. Mobile nodes and the MA server in the network have unique node IDs.

A3. A trusted offline certificate authority (CA) is used to bind public keys with respective mobile nodes and the MA server. Each node has a pair of private and public keys, and the public key can be known to other nodes via the certificate issued by the CA.

A4. The MA server and the mobile nodes have a shared symmetric key for message encryption. All messages between the MA server and the nodes in the network are encrypted with this symmetric key.

A5. Before deployment, mobile nodes in the network have agreement with the MA server about the security policies for the authorized actions an MA can perform on a node.

3.3.2 Attack Model

Security attacks in MANETs can be categorized into passive or active attacks. Passive attacks include eavesdropping of data, and traffic analysis and monitoring. Active attacks include replication, modification, insertion and deletion of data to be exchanged, external service attacks, resource consumption (e.g., DoS attacks), and physical attacks. Security attacks can also be categorized according to protocol layers. Some attacks on the application layer are data corruption, repudiation, application abuses, DoS attacks, and mobile virus and worm attacks. There are also some attacks that target across multi-layers, such as DoS attacks, impersonation, and man-in-the-middle attack.

In this chapter, we focus on detecting and defending against application-layer attacks mentioned above. Our goal is to detect malicious applications and mobile virus and worm attacks on the nodes in the network.

3.3.3 System Architecture

We consider a network of a large number of mobile nodes and one secure stationary MA server. There can be multiple MA servers deployed in the network to avoid single-point-of-failures. However, it will require coordination and interactions between the MA servers, so we focus on using one MA server in the following design. The MA server is used for managing MAs, normal application profiles, and attack signatures generated by MAs in the network, and is deployed in the network along with the mobile nodes. The MA server is akin to the command and control (C&C) center in a battle field. The C&C center can issue orders to the troops in the battle field. The MA server acts like the C&C center and dispatches MAs to the nodes in the network when needed. It will periodically broadcast beacon messages for nodes in the network to locate itself. When a node needs an MA for assistance, it will request an MA from the MA server. The proposed IDS architecture for MANETs is depicted in Figure 3.1.

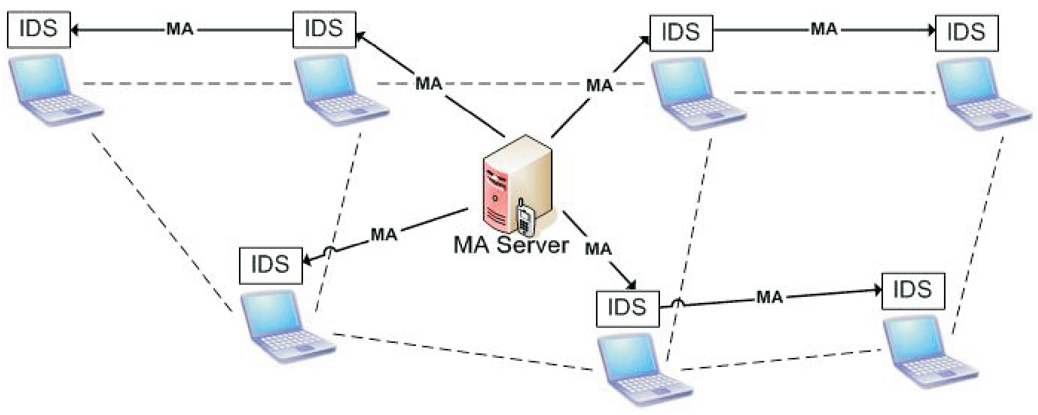


Figure 3.1: The IDS system architecture for MANETs.

Each mobile node has its own local IDS which is responsible for monitoring and detecting attacks, and for responding to the attacks detected. The local IDS performs anomaly and misuse detection. The misuse detection is used to detect known attacks on the node, while the anomaly detection is used for the detection of new or previously-unknown attacks. MAs are designed to update attack signatures and normal application profiles, patch and install programs, analyze and diagnose anomalous nodes, and verify the local IDS agents.

Each IDS consists of three agents: the monitoring and detection agent, the response agent, and the secure communication. There is also a local database in each node for storing system audit data, attack signatures, normal application profiles, and the IDS logs. For the execution of MAs, there is a mobile agent place on each node. The local IDS architecture in each mobile node is shown in Figure 3.2.

The monitoring and detection agent monitors the application-level activities and system calls on each node, and also compares the monitoring activities with the attack signatures and normal application profiles stored in the node's local database. Once a malicious activity is detected by the misuse detection via signature matching, a proper response will be formulated by the response agent to recover the node from the damages occurred to it. If the monitoring and detection agent detects intrusion by anomaly detection via an above-threshold deviation from the normal profile but

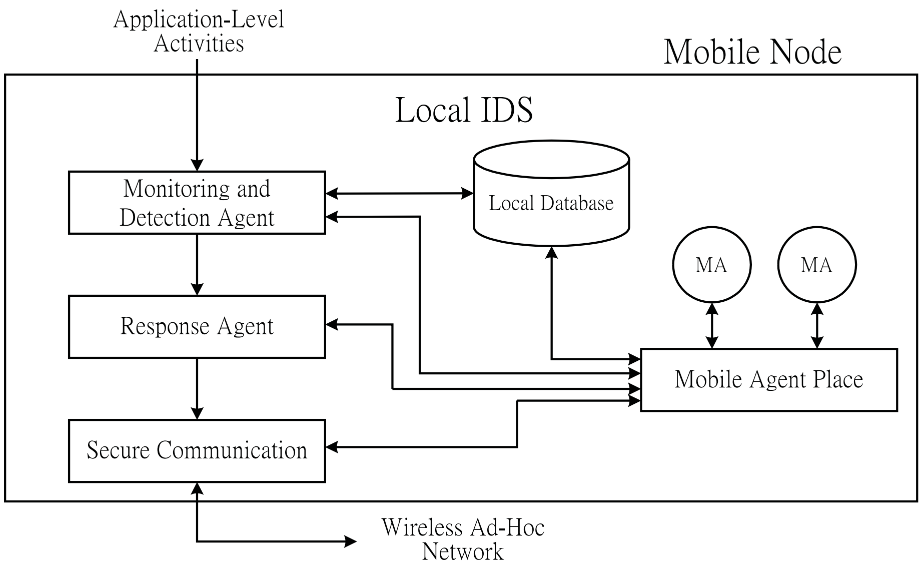


Figure 3.2: The local IDS architecture on a mobile node.

no signatures match the attack, then the response agent will request the MA server to dispatch an MA for further analysis and diagnosis. The secure communication component is used for the mobile nodes to securely communicate with the MA server and other nodes in the network.

3.4 Design of Application-Layer IDS

As mentioned earlier, our goal is to design an application-layer IDS that utilizes anomaly and misuse detection to identify malicious applications as well as mobile virus and worm attacks. MAs are utilized to augment each node's capability of intrusion detection in the network. We assume the existence of the MA server for managing and dispatching MAs to nodes in the network. Recall that each node in the network has its own local IDS.

Described below is the detailed design of using MAs for the application-layer intrusion detection.

3.4.1 Local IDS for Intrusion Detection

We use misuse detection as host-based intrusion detection of known attacks in each node. The monitoring and detection agent in the IDS audits the application-level activities and system calls in each node and compares them against the attack signatures stored in the node's database that represent worm and virus signatures, as well as misbehavior signatures of applications. The behavior-based attack signatures capture an incorrect order of instructions or incorrect control flow of programs, unlike traditionally-used signatures generated from raw byte sequences in malicious executables and takes up less storage space in the database.

The MAs dispatched to the mobile nodes can generate new attack behavior signatures, usually after the identification of an intrusion via anomaly detection, and the confirmation of the existence of intrusion after performing analysis and diagnosis of the nodes. The MAs will report these newly-generated attack signatures to the MA server, and the MA server will dispatch MAs to other nodes in the network to add the new attack signatures. For misuse detection, the MA server is used to store and update the attack signatures collected from the nodes in the network.

To use anomaly detection in our local IDS, a normal profile is computed for each application program using its audit data on the MA server. As we are targeting the detection of application-layer intrusions, the audit data collected on the MA server is the application-level activities and system calls invoked by the application programs. Specifically, the sequence of the system calls generated by each application program and collected in the audit data is used to compute the normal profiles.

The nodes will first be deployed in the network with the normal application profiles generated by the MA server for existing applications. When a new normal application profile has been generated due to a program patch or new application installation, MAs will carry the new profile to nodes in the network for update.

The monitoring and detection agent will compare the audit data of system traces on each node with the normal application profiles. The difference between a sequence of system calls against a normal application profile can be computed using Hamming distance. Any major deviation of abnormal activities from the normal application profiles will be detected and can be used as an alert to the local IDS. Once an anomaly is detected, the response agent will request an MA from the MA server for further analysis and diagnosis on the node. If the MA confirms the attack, then a proper response action will be taken at the node and a new attack signature will be generated for the detected attack.

Each node also logs all the IDS-related activities including both misuse and anomaly detection histories for MAs to analyze and report to the MA server. In addition to intrusion detection, MAs can also collect and analyze the audit data and IDS logs stored in local nodes. The collected information will be reported to the MA server and used for further analysis.

3.4.2 MA Functions

The MA server creates and dispatches three types of MAs: (1) update MAs, (2) analysis MAs, and (3) verification MAs. The update MAs are dispatched as needed, the analysis MAs are dispatched upon request by nodes, and the verification MAs are dispatched periodically. The three types of MAs are detailed next.

3.4.2.1 The Update MA

The update MAs are used by the MA server to add new attack signatures and normal application profiles, and patch and install programs on the mobile nodes. When the MA server receives new attack signatures generated by the analysis MAs, it will dispatch update MAs carrying the signatures to the nodes in the network. All nodes need to be updated with these new signatures for effective and up-to-date misuse detection.

If a vulnerability is detected and application programs on the nodes need to be patched, or new application programs are being installed, then the MA server will also dispatch to the nodes the update MAs carrying the patches and new programs. When a program is patched, then the normal application profile needs to be updated, or if a new program is installed, then the nodes also need to have a normal profile for the new application. The MA server will dispatch the update MAs along with the new normal application profiles generated by the MA server to the nodes for update.

The update MAs are dispatched to the nodes when needed, so the nodes need not request updates from the MA server. This will greatly reduce the number of requests from nodes and prevent the MA server from becoming a communication bottleneck. With the update MAs dispatched to the nodes, the MA server need not send the same updates to different nodes, and thus, the network load will be reduced significantly.

3.4.2.2 The Analysis MA

The analysis MAs are dispatched to the requesting nodes for further analysis and diagnosis for anomaly behaviors on them. When the monitoring and detection agent in a node's local IDS detects an anomaly but the anomaly did not match any attack signature in its database, then the response agent in the IDS will send an anomaly report to the MA server and request an analysis MA from the MA server for further investigation. The anomaly report includes the related IDS logs and the intrusion information about the anomaly detected on the node. Depending on the content of the anomaly report, the MA server will choose the most suitable analysis MA. The analysis MA is capable of a more detailed analysis and diagnosis than the local IDS, and can evaluate the detected anomaly behavior and determine if it is an intrusion.

If the analysis MA determines the anomaly behavior not to be an intrusion, then it will send a detection report to the MA server and destroy itself. However, if the anomaly behavior is determined as an intrusion, then the analysis MA will start the intrusion response through the response agent on the local IDS. Finally, the analysis MA will create an attack signature of the newly-identified intrusion and report the new attack signature to the MA server. After completing the response for the intrusion, the analysis MA will send a detection report to the MA server and destroy itself.

Note that if the analysis MA cannot determine if the anomaly is an intrusion or not, then it can request a different analysis MA from the MA server for help. The analysis MA can also migrate to neighbor nodes to perform further investigation. The investigation at the neighbor nodes is the multi-point network-based anomaly detection. The analysis MAs can determine new attacks by analyzing and diagnosing both on a node and on its neighbor nodes. If the analysis MA still cannot (un)confirm an anomalous behavior, it will then send the relevant IDS logs and audit data back to the MA server for further analysis.

The analysis MAs are dispatched to the nodes requesting for further analysis of anomaly behaviors. The reason for sending analysis MAs to the nodes instead of having the nodes send all the audit data, IDS logs, and related information to the MA server for analysis is to reduce the network load. Also, the MAs can overcome network latency as the analysis MAs can be dispatched from the MA server to perform analysis on nodes in real time. The analysis MAs can be executed on the nodes and can respond faster instead of needing to communicate back and forth with the MA server for assistance.

3.4.2.3 The Verification MA

The verification MAs are periodically sent by the MA server to verify the IDS agents on the nodes and check on the IDS logs and the local IDS execution states.

These MAs are used to prevent the local IDS from being compromised by attackers. Similar to the program integrity verification in PIV [39] and SWATT [46], we use hash verification to determine the integrity of the IDS agents on the nodes.

The MA server will periodically send the verification MA with a randomly-generated hash key to the network. The nodes that received the verification MA must execute it to check the integrity of the IDS agents in its local IDS. Since the MA server keeps copies of the IDS agents, we can use the verification MA to verify the integrity of these agents. When the verification MA is executed, it computes a hash over the IDS agents using the hash key it carries, and the hash value will be sent back to the MA server for verification. The verification MA will also check the IDS logs and see if there is any anomaly or unreported events. If a node fails the verification, then the MA server will either send an update MA to correct the IDS agents or shut down the entire node.

3.4.3 MA Authentication and Authorization

To authenticate MAs and nodes in the network, we use the public key infrastructure (PKI). We assume there is a trusted offline certificate authority (CA) that issues certificates to the MA server and nodes in the network. The certificates contain the public key and the ID of the owner node and let other nodes in the network verify the owner node's credential. The CA also issues each node a corresponding pair of private and public keys. When the MA server dispatches MAs to the network, it will include its certificate issued by the CA and sign the MAs with its private key for authentication.

When an MA is dispatched to the network and needs to send detection information back to the MA server, it will execute an encrypted function and secretly sign the detection reports for the MA server [43]. The MA carries a program to nodes in the network that implements an encrypted function for the digital signature. Before sending a detection report back to the MA server, the MA will execute the encrypted function on the local node to sign the report and attach the signature along with the report. When the detection report reaches the MA server, the MA server can check the signature to verify the authenticity of the report.

The complete authentication steps are described as follows.

S1. The CA issues the MA server S and each mobile node a pair of private and public keys, and a certificate that contains the public key and ID of the node. S and mobile nodes share a symmetric key K.

S2. Each update and verification MA MA_S sent by S carries S's certificate $Cert_S$, and S encrypts MA_S with K and signs $\{MA_S\}_K$ with its private key k_S.

S3. Mobile node A that receives $\{MA_S\}_K$ from S verifies $Cert_S$ and gets S's public key K_S from $Cert_S$. If the $Cert_S$ verification is valid, A will then verify S's signature with K_S. If the verification of the signature is valid, then $\{MA_S\}_K$ will be decrypted using K and then executed. MA_S will be deleted if any of the verification fails.

S4. If MA_S is an update or verification MA, then MA_S will be sent to another mobile node B for update or verification. The previous steps are repeated until MA_S expires and is deleted.

S5. If there is an anomaly detected on A by the local IDS, A will send an anomaly report $AReport_A$ to S requesting for an analysis MA. A will encrypt $AReport_A$ with S's public key K_S, and then sign $\{AReport_A\}_{K_S}$ with A's private key k_A for authentication. Finally, A sends the encrypted report $\{AReport_A\}_{K_S}$, the signature, and its certificate $Cert_A$ to S.

S6. When S receives $\{AReport_A\}_{K_S}$ from A, it will fist verify $Cert_A$ and get A's public key K_A. If the $Cert_A$ verification is valid, S will then verify A's signature with K_A. Finally, S can decrypt $\{AReport_A\}_{K_S}$ with its private key k_S if the verification succeeds. However, $AReport_A$ will be discarded if any of the verification fails.

S7. S will send analysis MA MA_S to A as described in S2, and A can authenticate MA_S as described in S3. After MA_S is executed on A, MA_S will need to send a detection report $DReport_{MA_S}$ to S for a status report or request for help. A will first encrypt $DReport_{MA_S}$ with S's public key K_S, then execute MA_S to get the digital signature DS_1 for $\{DReport_{MA_S}\}_{K_S}$ using the encrypted function carried by MA_S. A then signs $\{DReport_{MA_S}\}_{K_S}$ with its private key k_A to get digital signature DS_2 and combine it with DS_1. Finally, A sends the encrypted report $\{DReport_{MA_S}\}_{K_S}$, the signatures $DS_1 + DS_2$, and its certificate $Cert_A$ back to S.

S8. When S receives $\{DReport_{MA_S}\}_{K_S}$ from A, it will first verify $Cert_A$ and get A's public key K_A. If the $Cert_A$ verification is valid, S will then verify $DS_1 + DS_2$ with K_A and the encrypted function. Finally, S can decrypt $\{DReport_{MA_S}\}_{K_S}$ with its private key k_S if the verification succeeds. However, the encrypted report $\{DReport_{MA_S}\}_{K_S}$ will be discarded if any of the verification fails. Depending on $DReport_{MA_S}$, S will then decide whether to send more verification MAs to A for intrusion detection and response or to process and update the reported information received.

As for MA authorization, we assume that before deployment, mobile nodes in the network have an agreement with the MA server about the security policies for the authorized actions an MA can perform on the nodes. The MA server should only authorize legal actions for MAs to perform on each node. Before an MA can be executed on each node, it will need to pass the authentication. After the authentication, however, if the MA attempts to perform actions that are not on the authorized list, the node will disallow the MA to take them.

3.4.4 MA Dispatching

As described in Section 3.4.2, there are three types of MAs: the update MAs, the analysis MAs, and the verification MAs. The update MAs are sent as needed, the analysis MAs are sent upon request, and the verification MAs are sent periodically. The MA server dispatchs MAs and controls the generation of MAs, their quantities, and their communication timing.

The time interval for periodic verification MAs to be sent to the network can vary. Depending on the condition and state of each network area, the MA server can decide adaptively how often to dispatch verification MAs to a certain area to verify the local IDSs. However, the MA server is susceptible to DoS attacks from malicious nodes that request analysis MAs or send false detection reports. We can maintain packets' path histories and determine the source of attacks. We can also restrict the number of times a node can contact the MA server within a certain time duration. For confidentiality and authentication purposes, we use the network-wide symmetric key for encryption and PKI for digital signature, as stated in Section 3.4.3.

When MAs are dispatched to the network, they send back two types of reports to the MA server: periodic and detection reports. The periodic reports are for fault-tolerance and data collection. In case the MA server didn't receive the periodic reports from an MA, it will check for the MA or send more MAs to continue the MA's task. The update and verification MAs need to periodically send periodic reports back to the MA server. The periodic reports include the nodes the MA visited, the nodes' battery information, and the nodes' update/verification results including the related IDS information. The detection reports are sent back to the MA server by the analysis MAs to report the analysis result for the anomaly behaviors detected, to request more analysis MAs from the MA server, or to report relevant IDS logs, audit data, and intrusion evidence to the MA server for further analysis. The detection reports include where the anomaly behavior is detected, the related IDS logs for the anomaly, and if the anomaly is confirmed and countermeasures applied.

Each MA has a time-to-live (TTL) parameter set by the MA server that determines the number of nodes the MA can visit in the network. After each MA visits a node, the TTL will be decremented by one. When the TTL on the MA expires, the MA destroys itself.

The MA-dispatching protocol is summarized as follows.

1. When an MA server S dispatches MA_S to the network, it includes its certificate $Cert_S$ and encrypts MA_S with the network shared symmetric key K and signs MA_S with its private key k_S for authentication as described in Section 3.4.3.

2. If MA_S is an update MA, then S includes in MA_S the new attack signatures it collected from the network for signature update. S also includes any program patch or new program in MA_S for program update, and the newly-generated normal application profiles.

3. If MA_S is an update MA or a verification MA, then S will dispatch MA_S to a network area for random traverse. If MA_S is an analysis MA, then it will be dispatched to the requesting node.

4. Each MA will have a TTL parameter that determines the number of nodes the MA can visit in the network.

5. While MA_S travels in the network, it sends back periodic or detection reports to S to report its update/verification or diagnosis results on each node. The encryption and authentication details are described in Section 3.4.3.

6. An MA destroys itself upon expiration of its TTL.

After its execution on one node, the update and verification MA will move to other nodes in the network for update/verification, and the analysis MA can decide whether to migrate to neighbor nodes for further analysis and diagnosis or to destroy itself after it finishes the analysis. The MA server can send multiple MAs to different network areas. The more MAs sent out, the faster the update/verification will be done. If fewer MAs are dispatched, then it will take longer for MAs to visit other nodes and finish their tasks. Dispatching too many MAs will incur more traffic as the same MA may visit a node multiple times. We would like to limit the amount of communications between the MA server and the nodes in the network. Therefore, we need to find an optimum number of MAs to be dispatched to the network.

3.4.4.1 Message-Loss/Compromise Problems

Since MAs need to travel multiple hops through the network to reach certain nodes or send periodic and detection reports back to the MA server, we need to consider message-loss/compromise problems that might occur at the intermediate relay nodes. We use secure routing [38] to send MAs to the network and periodic and detection reports back to the MA server.

If MAs are lost or captured by malicious intermediate nodes, nodes in the network will not receive the periodically sent verification MAs or the requested analysis MAs. If nodes have not received MAs for longer than a certain time $Time_{\text{MA}}$, then they can request MAs from the MA server again.

In case periodic reports are lost or captured, the MA server will not receive the periodic reports from MAs nor know the MA status and the update/verification results from the reported information. If the MA server has not received periodic reports from an MA for longer than a certain time $Time_{\text{report}}$, then the MA server will look up the previous periodic reports the MA returned and then mark the visited/routing nodes as suspicious nodes. The MA server will then send query messages to the suspicious nodes to look for the MA and request periodic reports from the MA. If still no periodic reports are received by the MA server, then it will send analysis MAs to the suspicious nodes for anomaly detection. If the detection reports are lost or captured, then the MA server can send more analysis MAs to the node under investigation.

3.4.5 Intrusion Response

When an intrusion is detected on a node either by misuse detection (via attack signature matching) or by anomaly detection (via major deviation from a normal application profile), the response agent in the node's local IDS will respond to the detection. The response depends on the degree of damages done by the intrusion, the type of intrusion, and the type of the malicious application. When an anomaly is detected, the response agent will ask the MA server for an analysis MA for further analysis and proper intrusion response.

The response agent can try to re-program or disinfect the compromised node if the damage caused by the intrusion can be fixed by re-programming the node. Or, the response agent can ask the MA server for program patches (using update MAs). The response agent can also send notifications and alerts to the network for re-authentication, or exclude and shut down the compromised nodes.

3.5 Security Analysis

We first discuss the security of the MA-based IDS when MAs and nodes are compromised, and then analyze the security of the MA-based IDS protocol against various attacks.

3.5.1 Defense Against Compromised MAs and Nodes

An attacker must first be able to fake the signature of an MA with the MA server's private key before he wants to compromise the MA. Only when the attacker can get the compromised MA to be signed, will the MA pass the authentication and be executed on nodes in the network. Even if the attacker has successfully faked the signature and the compromised MA has passed the authentication but tries to perform illegal actions that are not in the authorized list of the node, then the node will protect itself by disallowing the MA to execute the unauthorized actions.

It might be possible for the compromised update MAs to carry the faked normal application profiles and attack signatures to the nodes, and let the node's IDS detect intrusions even when the node actually behaves normally. The IDS response agent will then respond to the attack and request analysis MAs from the MA server for further analysis and diagnosis. The analysis MAs will be dispatched to the node and will aid its detection and response to the intrusions. The compromised update MA can then be detected when the analysis MAs arrive and analyze the IDS logs on the node.

As for a compromised node that has subverted its local IDS, the verification MAs will arrive at the node periodically and verify the integrity of the IDS agents on the node. The compromised IDS will be detected if the IDS agents have been modified, or the IDS logs and execution states are incorrect. Note that a compromised node has to execute the verification MA; otherwise, the verification MA will suspect the node to have been compromised.

If the compromised node can successfully pass the verification and alter its IDS logs, then it must also pass the analysis performed by the analysis MAs that may be sent to the node for further investigation. Therefore, for a compromised node to go undetected by the analysis MAs, it cannot perform abnormal activities during the MA's diagnosis. If the node discards the analysis MA and hence no detection reports are delivered to the MA server, then the MA server will dispatch more analysis MAs for further detection. Also, when the neighbor nodes detect the abnormal behaviors from the compromised node, they will request analysis MAs, and the compromised node can be excluded from the network when all the nodes are required to be re-authenticated.

3.5.2 Defense Against Various Attacks in Ad Hoc Networks

We now describe how the MA-based IDS protocol can defend against various attacks in MANETs.

3.5.2.1 Defense Against Passive Attacks

When an MA is to be dispatched by the MA server, it is encrypted with the network-wide shared symmetric key, along with the signature signed by the MA server's private key. The encryption is to prevent attackers from eavesdropping the network and seeing the content of the MAs. However, if one node is compromised and the shared key is revealed, then it is necessary to renew the shared key among the nodes in the network. We can also group nodes depending on the geographical areas of the network and assign different group keys to different groups. When a node leaves or joins a group, the group key will need to be renewed or distributed securely to the joining node [36, 55].

As for the periodic and detection reports that the MAs send back to the MA server, they are all encrypted with the MA server's public key, so only the MA server can decrypt the reports. The reports are also signed by the MA's signature function and the node where the MA resides. The MA digital signature function is encrypted, so neither the attacker nor the node can see the signature function. Therefore, the attacker will not get the contents of the periodic and detection reports by simply eavesdropping on the network.

3.5.2.2 Defense Against Active Attacks

To prevent an attacker from spoofing or inserting false data, we sign every MA, the periodic and detection reports, and the anomaly reports from nodes with the MA server's or the nodes' private keys to achieve authenticity and integrity. We also encrypt the MAs with the network-wide symmetric key. The MAs also carry an encrypted function for digital signature to ensure the authenticity of the periodic and detection reports.

The node compromised by an attacker can be detected by the local IDS, and the response agent in the IDS will handle the intrusion. If the local IDS is compromised, then the periodically-sent verification MAs will be able to detect the faulty IDS agents, and the MA server will dispatch analysis MAs for diagnosis and response.

Malicious nodes can cause service disruption and Denial-of-Service (DoS) attacks. There is not an easy way to prevent such nodes from launching attacks, but they can be detected and then removed from the network. We rely on the local IDS at each node to detect the nodes' malicious behaviors. Since we let a local IDS monitor and detect known intrusions and anomalies on each node, and let MAs aid the detection, once a node identifies a malicious or anomaly behavior, its IDS response agent can evict the compromised node from the network and the neighbor nodes will ignore any messages from the compromised node.

Another potential attack is for a node to launch DoS attacks to the MA server, requesting MAs from or sending reports to the MA server. This kind of DoS attacks targeting the MA server can be handled by having the MA server keep path histories of the messages sent to it in order to pinpoint the attacker and restrict the number of times a node can contact it within a certain time duration.

Sybil attacks [20] are particularly harmful in MANETs where a Sybil node illegitimately fakes to have multiple identities in the network. Our MA-based IDS withstands such attacks since each node will need to have a private key and a matching certificate to authenticate its identity. Since each node will have a preloaded private key and certificate, no node can generate the private key and certificate, and pretend to be another node without compromising the node.

3.6 Evaluation

We use simulation to evaluate the trade-offs of different design parameters in the MA-based IDS. We developed our own simulator using C. We first simulate with different MA time to live (TTL) values and numbers of MAs dispatched by the MA server to the network to learn the number of nodes in the network that did not receive any MA. Our goal is to have as few nodes in the network as possible that did not receive the dispatched MAs. In this simulation, we study the trade-offs between dispatching more MAs to the network or allowing the MAs to travel more hops in the network and have larger TTL values. We can also study the number of nodes in the network that have received multiple MAs during dispatching. By studying the number of nodes that did not receive MAs and the number of nodes that did receive multiple MAs, we can determine the suitable number of MAs to be dispatched and the appropriate MA TTL value.

We also simulate the deployment of multiple MA servers in the network, and examine how this affects the MA distribution. In this simulation, we study the trade-offs between positioning the MA servers randomly or uniformly. We also simulate the deployment of more MA servers in the network.

Last, we simulate networks that have non-uniformly distributed nodes in skewed distributions. We compare and discuss the results of the number of nodes in the network that did not receive any MA and the number of nodes in the network that did receive multiple MAs to the results in uniformly randomly-generated networks.

3.6.1 Simulation Setup

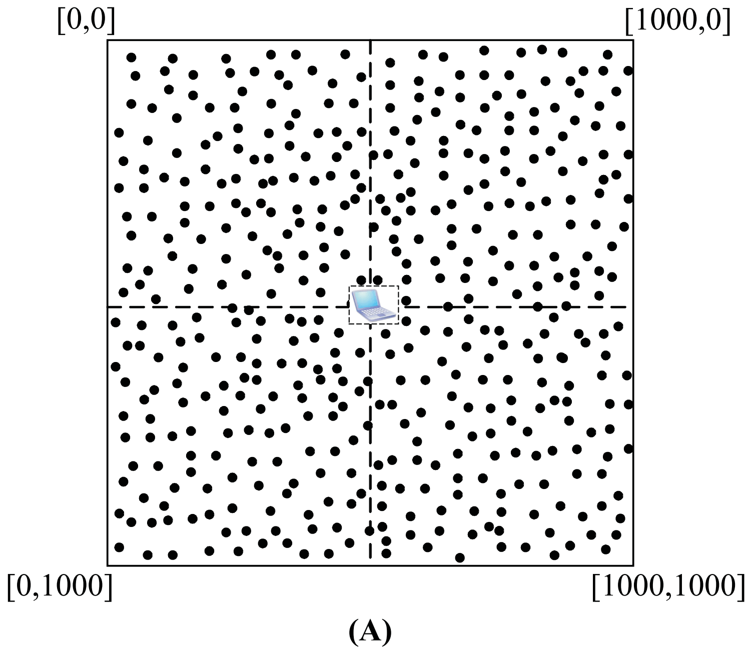
We first simulate the MA-based IDS with uniformly randomly-generated networks consisting of 1000 nodes and an MA server in a 1000 $\times$ 1000 units^2 area. Nodes and the MA server are assumed to have a communication range of 75 units. The MA server is deployed at the center of the network at (500, 500). On average, the MA server has 15–20 neighbor nodes. This simulation environment is depicted in Figure 3.3(A).

We simulate and compare the number of MAs dispatched by the MA server with different MA TTL values. By changing the MA TTL values and the number of MAs dispatched, we examine the number of nodes that did not receive the MAs dispatched from the server and the number of nodes that did receive multiple MAs. In the simulation, the server will first dispatch some MAs to its neighbor nodes in the network. After receiving and executing the MA, the MA server neighbor node will forward the MA to another node; this will repeat until the MA's TTL expires, at which time the MA will destroy itself.

Next we simulate the case with the deployment of four MA servers in the network. The four MA servers are deployed at (250, 250), (750, 250), (250, 750), and (750, 750), respectively. On average, each MA server has 15–20 neighbor nodes. The simulation environment is depicted in Figure 3.3(B). While changing the MA TTL values and numbers of MAs dispatched, we study and compare the MA distribution results with only one MA server in the network. We also simulate the case with the random deployment of 4 and 5 MA servers in the network and compare the MA distribution results. With MA servers randomly deployed, each MA server has an average of 10–20 neighbor nodes.

3.6.2 Results

The MA-based IDS is simulated with 10 different MA TTL values, ranging from 20 to 200, with 20-hop increments. The network can be separated into four quadrants,



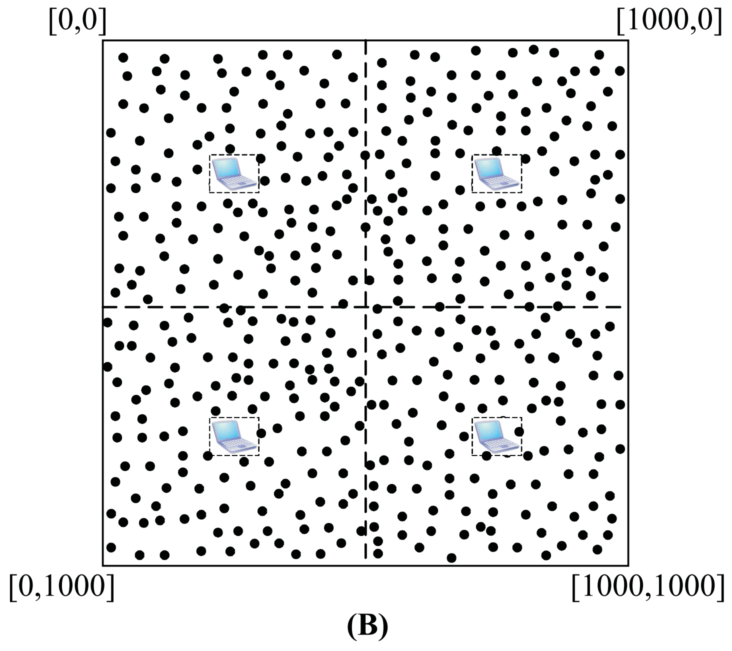


Figure 3.3: The simulation environment for MA-based IDS. Laptops represent the MA servers and black circles represent mobile nodes in the network.

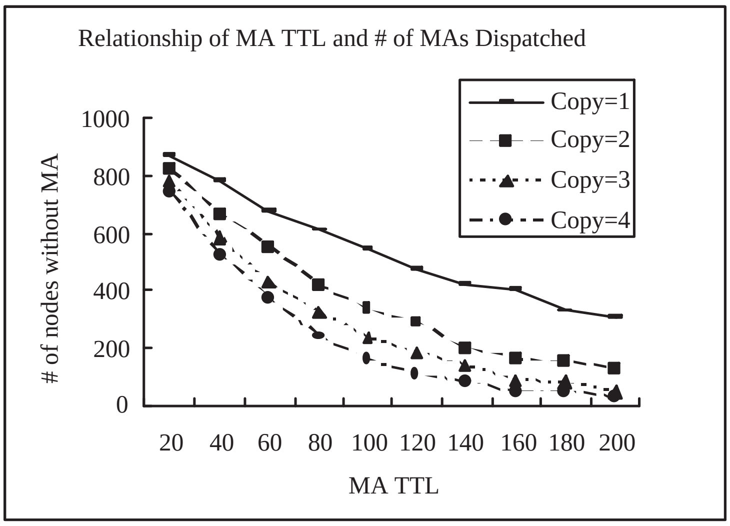
with one MA server located at the center of the network, as shown in Figure 3.3(A). Since on average the MA server has 15–20 neighbor nodes, it will dispatch 10 MAs by sending them to ten of its randomly-selected neighbors. After receiving the MAs from the servers, the nodes will execute the update or verification MA, then forward the

MAs to their randomly-selected neighbor nodes. Note that a node will not forward the MA to the node where it received the MA from, unless the node has only one neighbor.

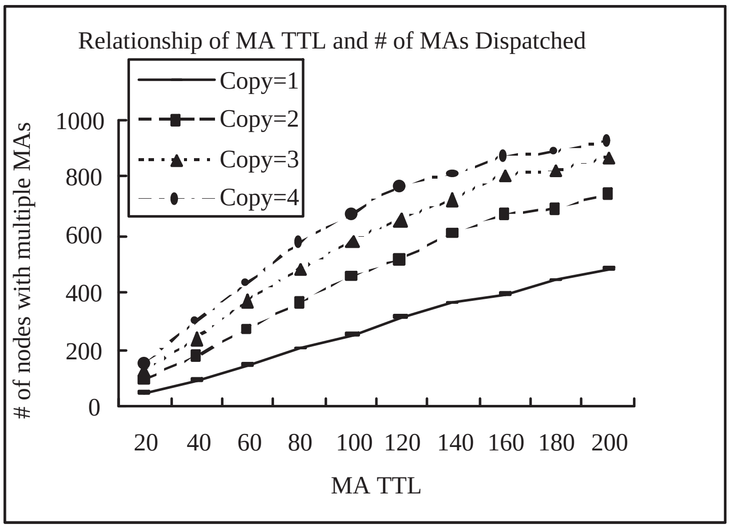
If the MA server dispatches only 10 MAs with TTL=20, then at most 200 nodes will receive the MAs. There are, however, 1000 nodes in the network; those that have not received MAs will then need to request MAs from the server. We further let the server send extra MAs to its neighbors and let them forward the MAs without executing them. We simulated the system while the server sends 0 extra MA (Copy=1), 1 extra MA (Copy=2), 2 extra MAs (Copy=3), and 3 extra MAs (Copy=4), respectively.

The simulation results for different MA TTL values and numbers of MAs dispatched by the server (including the extra sent MAs) are plotted in Figure 3.4. The four curves represent different numbers of MAs dispatched by the server. The Copy=1 curve represents the case of the MA server dispatching 10 MAs, whereas the Copy=4 curve represents the case when each MA server dispatches 40 MAs. As shown in the four curves in Figure 3.4(A), the more MAs dispatched, the fewer nodes in the network are left without MAs visiting them. The MA TTL values also affect the number of nodes receiving MAs. The larger the MA TTL value, the more nodes the MA can visit during its lifetime. Therefore, the larger the MA TTL value, the fewer nodes in the network are left without MAs visiting. However, as shown in Figure 3.4(B), the more MAs dispatched, the more nodes in the network will have more than one MA being dispatched to them. Also, the larger the MA TTL value, the more nodes will have more than one MA dispatched to them.

From Figure 3.4, we can see a trade-off between minimizing the number of nodes in the network without MAs and the number of nodes that have more than one MA dispatched to them. The goal is to minimize the number of nodes in the network without MAs, but at the same time, we should not have too many nodes with the



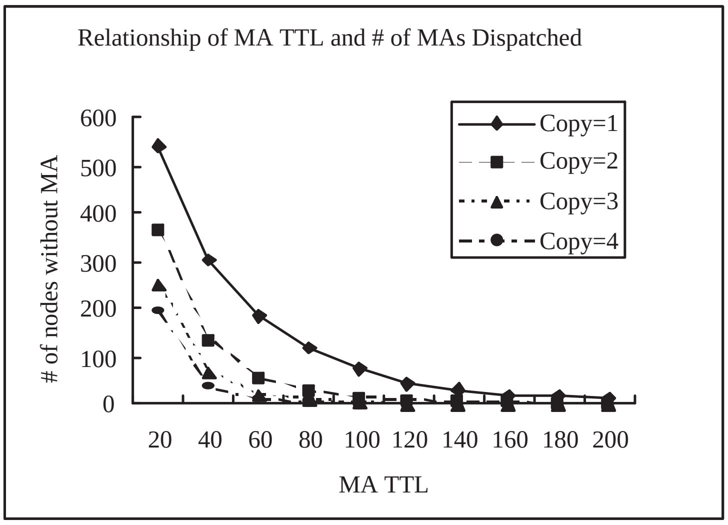
(A)



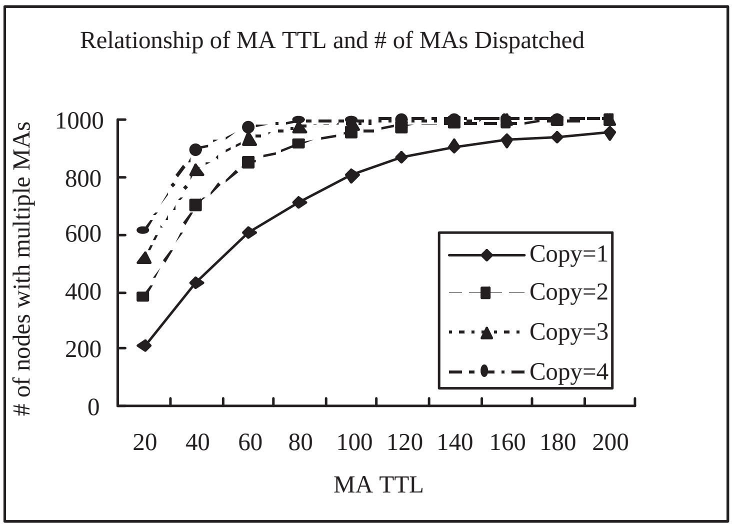
(B)

Figure 3.4: The relationship between different MA TTL values and numbers of MAs dispatched (including the extra sent MAs) by one MA server using the simulation environment in Figure 3.3(A). The number of nodes that did not receive any MA is shown in (A) and the number of nodes that did receive multiple MAs is shown in (B).

same MAs dispatched to them. There is also a trade-off between whether to dispatch more MAs to the network or to make the MA TTL larger. The more MAs are dispatched, the greater the traffic and communication overheads. However, the larger the MA TTL, the longer the MAs will live, the longer it takes for all nodes to receive



(A)



(B)

Figure 3.5: The relationship between different MA TTL values and numbers of MAs dispatched (including the extra sent MAs) by four uniformly positioned MA servers using the simulation environment in Figure 3.3(B). The number of nodes that did not receive any MA is shown in (A) and the number of nodes that did receive multiple MAs is shown in (B).

the MAs and the higher risk for the MAs to be captured or compromised.

The simulation results for four MA servers deployed in the network, with one MA server located at the center of each quadrant as shown in Figure 3.3(B), are plotted in Figure 3.5. From Figure 3.5, we can see that when the MA TTL becomes 40, both

Copy=3 and Copy=4 curves have less then 100 nodes that have not received MAs (65 nodes when Copy=3 and 35 nodes when Copy=4). This is acceptable for a network of 1000 nodes. Again, there is a trade-off between whether to dispatch more MAs in each period or to make the MA TTL larger.

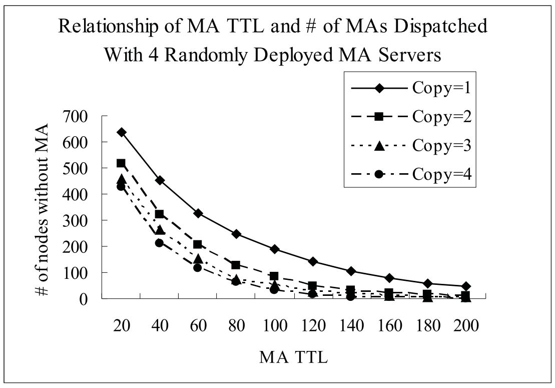
From a comparison of Figure 3.4 (one MA server in the network) with Figure 3.5 (four MA servers in the network), we can see that if there are more MA servers deployed in the network, then more MAs will be dispatched and fewer nodes will go without receiving MAs. However, there will also be more nodes in the network to have received more than one MA.

The simulation results for randomly-deployed MA servers are plotted in Figure 3.6. From Figure 3.6 and Figure 3.5, we can see that when four MA servers are randomly deployed in the network, the number of nodes without MAs generally increases and the number of nodes with multiple MAs decreases relative to when the MA servers deployed at the center of each network quadrant. The reason is because when MA servers are randomly deployed, there might be multiple MA servers in one quadrant and no MA server in another quadrant. Therefore, uneven distribution of MA servers will cause the MA distribution to be less effective.

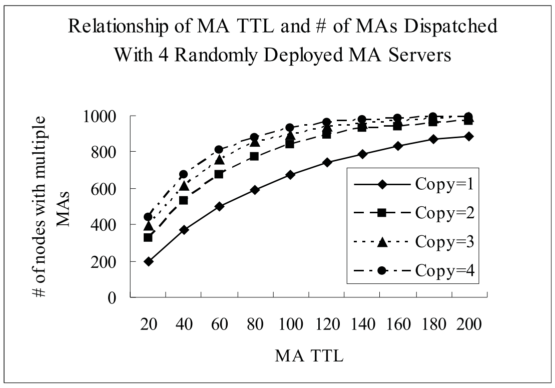
From a comparison of the MA distribution results of having four randomly-deployed MA servers (as shown in Figure 3.6 (A) and (B)) against five randomly-deployed MA servers (as shown in Figure 3.6 (C) and (D)), we can see that the result is not much different. This is because there are already enough MAs dispatched to the network; therefore simply dispatching more MAs or deploying more MA servers in the network will not help the MA distribution.

3.6.3 Non-uniformly Distributed Network Simulation and Results

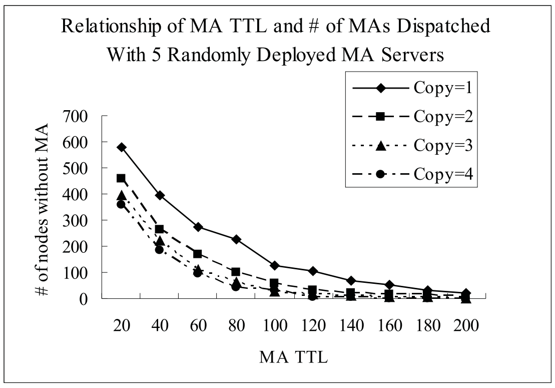
We have also simulated the MA-based IDS with non-uniformly but rather skewed randomly-generated networks consisting of 1000 nodes and an MA server in a 1000×1000



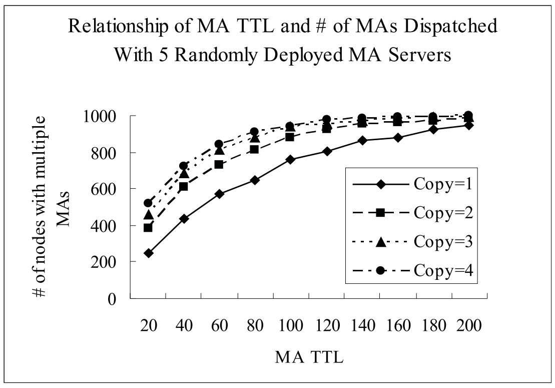
(A)



(B)



(C)



(D)

Figure 3.6: The relationship between different MA TTL values and numbers of MAs dispatched (including the extra sent MAs) by randomly-deployed MA servers. With four MA servers deployed, the number of nodes that did not receive any MA is shown in (A) and the number of nodes that did receive multiple MAs is shown in (B). With five MA servers deployed the number of nodes that did not receive any MA is shown in (C) and the number of nodes that did receive multiple MAs is shown in (D).

units^2 area. The simulation environment is the same as previous simulations and the MA server is deployed at the center of the network at (500, 500). We divide the network into four quadrants, as depicted in Figure 3.7. We simulate three different network deployments: (1) 700 nodes randomly deployed in quadrant 1, and 100 nodes randomly deployed in quadrant 2, 3, and 4 respectively; (2) 100 nodes randomly deployed in quadrant 1 and 3, and 400 nodes randomly deployed in quadrant 2 and 4; and (3) 100 nodes randomly deployed in quadrant 1 and 2, and 400 nodes randomly

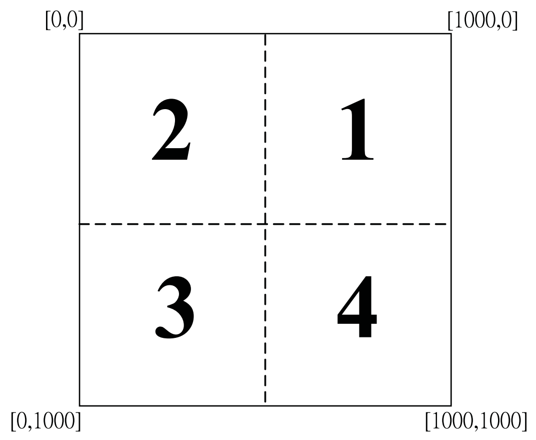
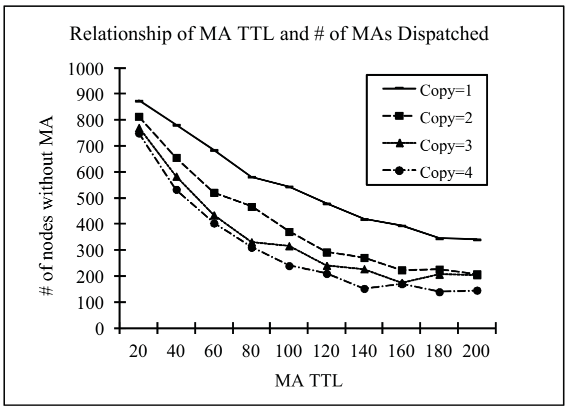


Figure 3.7: The simulated network quadrant layout.

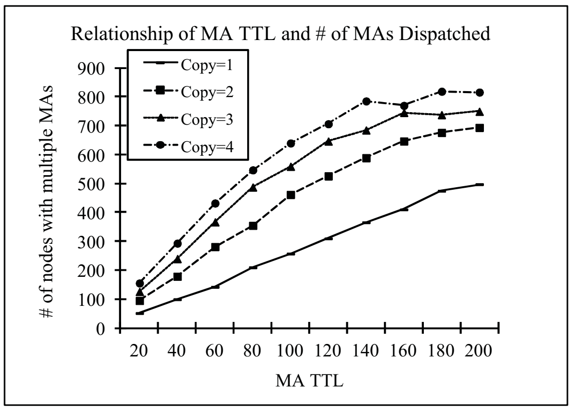
deployed in quadrant 3 and 4.

The simulation results for a centrally deployed MA server with three different skewed network deployments are shown in Figure 3.8. With network deployment (1), the number of nodes that did not receive any MA is shown in (A) and the number of nodes that did receive multiple MAs is shown in (B). Simulation results for network deployment (2) and (3) are shown in (C), (D), and (E), (F) respectively. As shown in Figure 3.8, the results don't differ much from the results for a uniformly distributed network shown in Figure 3.4. This is because nodes that cluster in one quadrant will be closer to each other and more easily allow the distribution of MAs, but nodes in other quadrants will be further from each other and more prone to be isolated from the network. Figure 3.9 shows the MA distribution network diagrams for different network deployments having a centrally deployed MA server distributing 10 MAs to the network with MA TTL values set to 80. Empty circles represent nodes in the network that did not receive any MA, and solid circles represent nodes that have received one or more MAs. We can see from the diagrams that when networks are clustered, then the MAs are distributed more in the clustered area; in a uniformly distributed network, the MAs are evenly distributed in the network.

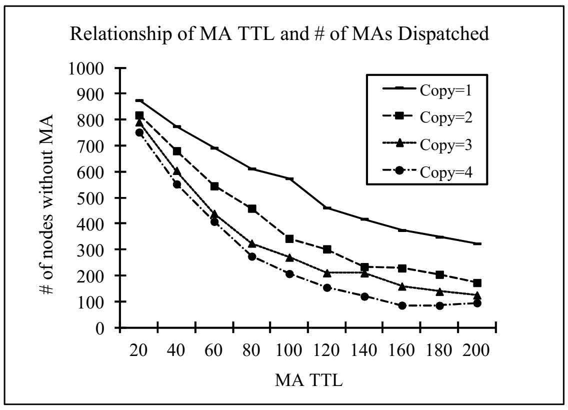
Last we simulated the MA-based IDS with skewed randomly-generated networks



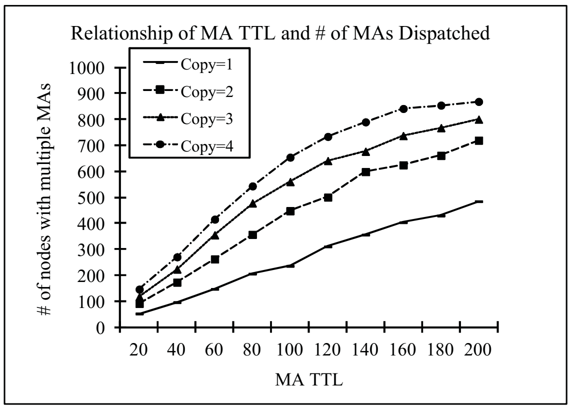
(A)



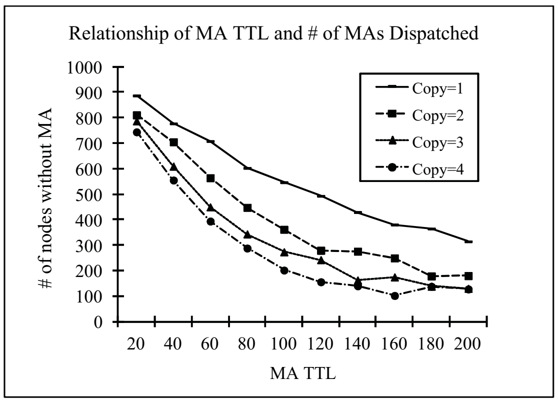
(B)



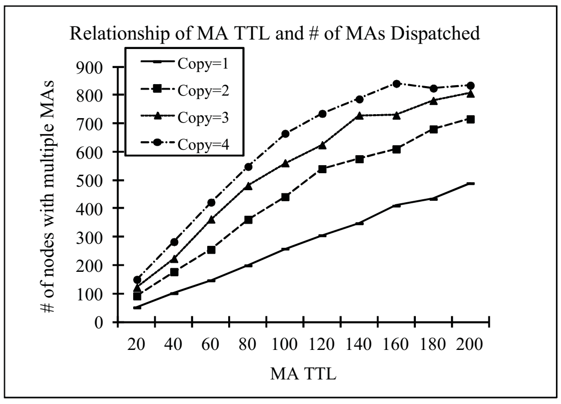
(C)



(D)

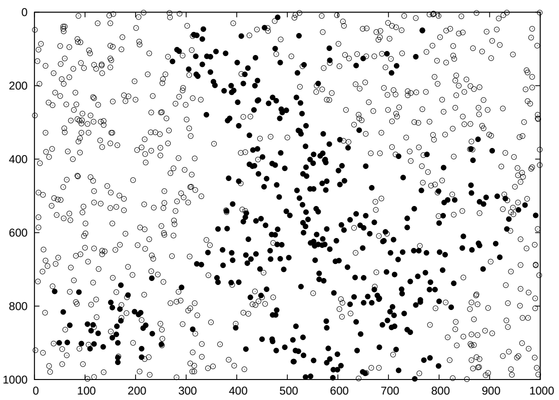


(E)

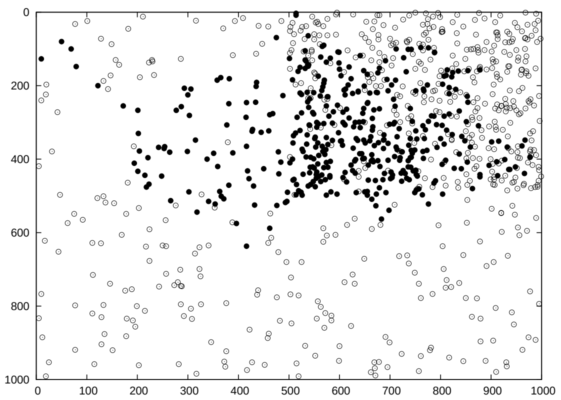


(F)

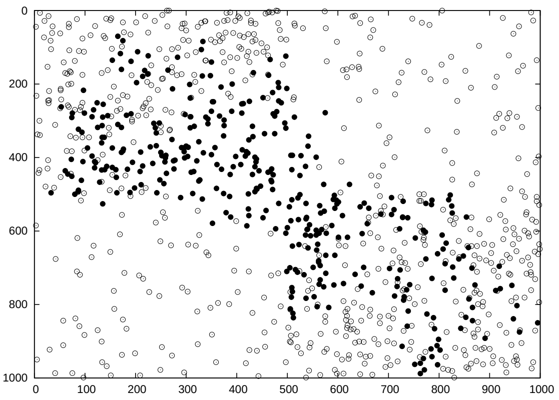
Figure 3.8: The relationship between different MA TTL values and numbers of MAs dispatched (including the extra sent MAs) by one MA server using the simulation environment in Figure 3.3(A). With network deployment (1), the number of nodes that did not receive any MA is shown in (A) and the number of nodes that did receive multiple MAs is shown in (B). Simulation results for network deployment (2) and (3) are shown in (C), (D), and (E), (F) respectively.



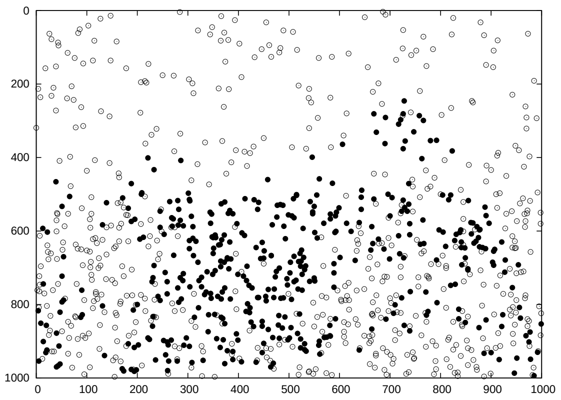
(A) Uniform network deployment



(B) Network deployment (1)



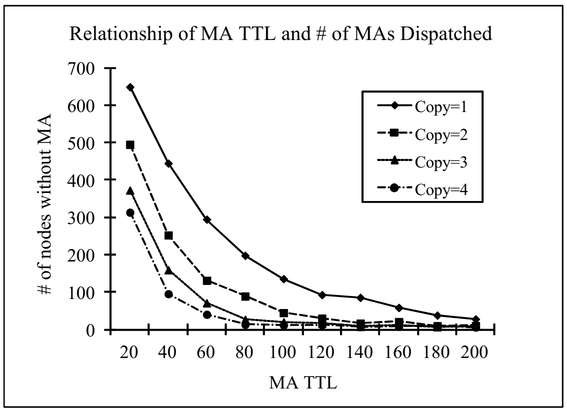
(C) Network deployment (2)



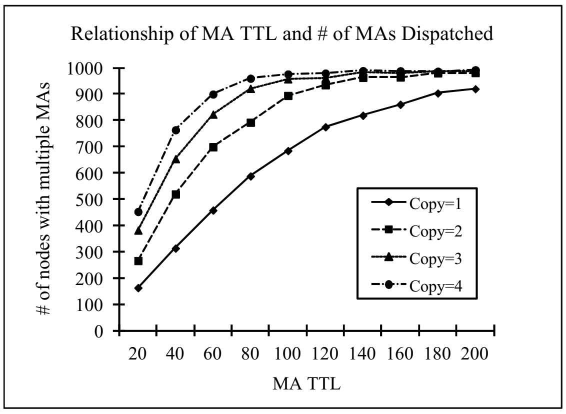
(D) Network deployment (3)

Figure 3.9: Comparison of MA distribution results between different network deployments. Empty circles represent nodes in the network that did not receive any MA, and solid circles represent nodes that has received one or more MAs.

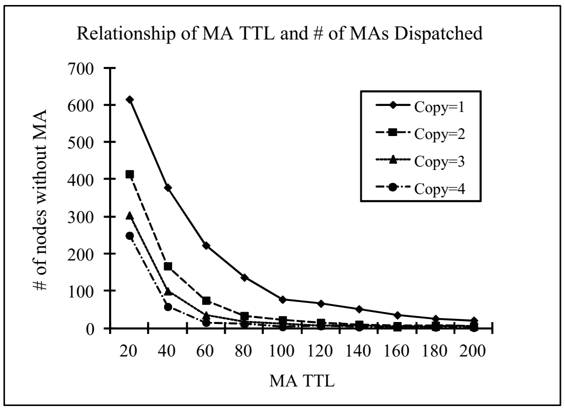
consisting of 1000 nodes in a 1000 $\times$ 1000 units^2 area and four MA server deployed at (250, 250), (750, 250), (250, 750), and (750, 750), respectively. The simulation results for four uniformly deployed MA servers with three different skewed network deployments are shown in Figure 3.10. With network deployment (1), the number of nodes that did not receive any MA is shown in (A) and the number of nodes that did receive multiple MAs is shown in (B). Simulation results for network deployment (2) and (3) are shown in (C), (D), and (E), (F) respectively. As with the previous simulation, the results here don't differ much from the results in a uniformly distributed network as shown in Figure 3.5.



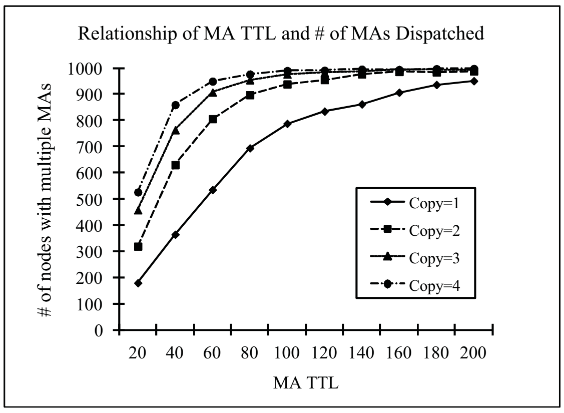
(A)



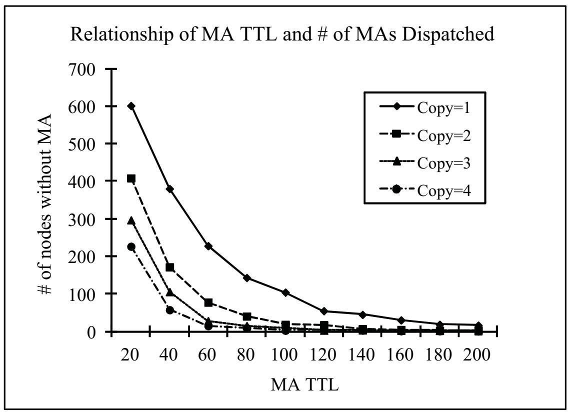
(B)



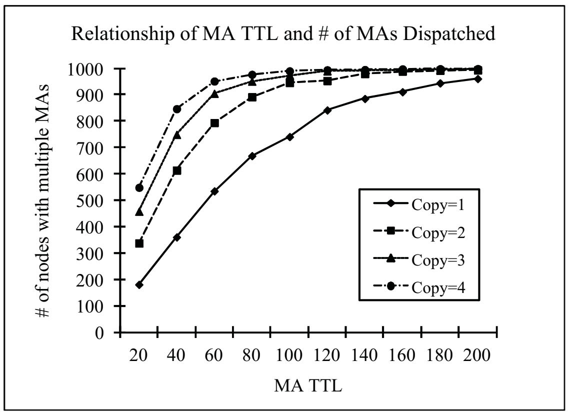
(C)



(D)



(E)



(F)

Figure 3.10: The relationship between different MA TTL values and numbers of MAs dispatched (including the extra sent MAs) by four uniformly positioned MA servers using the simulation environment in Figure 3.3(B). With network deployment (1), the number of nodes that did not receive any MA is shown in (A) and the number of nodes that did receive multiple MAs is shown in (B). Simulation results for network deployment (2) and (3) are shown in (C), (D), and (E), (F) respectively.

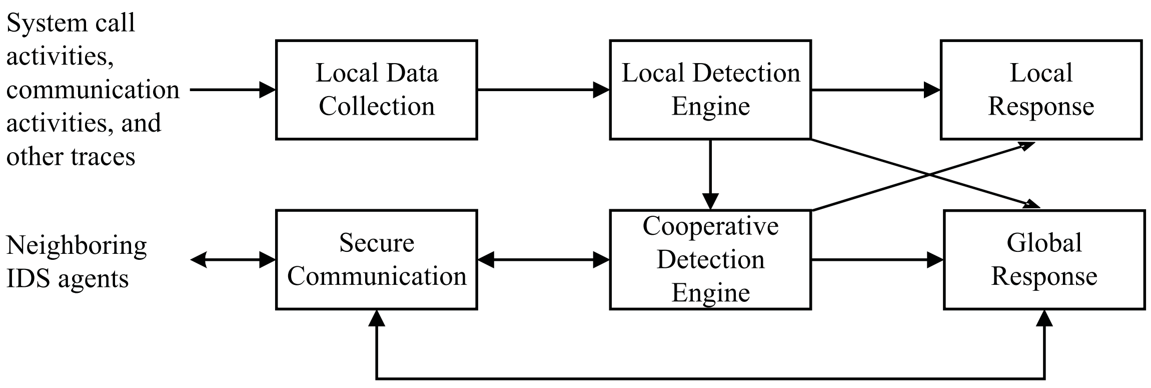


Figure 3.11: The intrusion detection system for MANETs proposed in [57].

Therefore, given the simulation results in different network deployments, we can conclude that the MA distributions will not be affected by the network deployment and topology, but will be affected by both the MA server deployment and the number of MAs distributed and the different MA TTL values.

3.7 Related Work

There have been several different proposals for the design of IDSs for MANETs. We list them below and compare some of them with our proposed approach.

Zhang and Lee [57] proposed a distributed and cooperative intrusion detection model for MANETs. In their model, every node in the network runs an IDS agent, and performs data collection and intrusion detection locally, with cooperative detection and global response triggered whenever a node reports detection of an anomaly. The intrusion detection architecture is based on statistical anomaly detection techniques [57]. The internal structure of the IDS agent is divided into six models, as shown in Figure 3.11.

Sergio Marti *et al.* [37] discussed two techniques, watchdog and pathrater, that improve throughput in MANETs in the presence of compromised nodes that agree to forward packets but fail to do so. Watchdogs will identify misbehaving nodes, and pathraters will aid routing protocols to avoid misbehaving nodes. However, the watchdog technique might not detect misbehaving nodes under certain conditions.

Several researchers proposed the use of MAs for intrusion detection in MANETs. Local Intrusion Detection System (LIDS) proposed by Albers *et al.* [3] utilized MAs on each node in a MANET. LIDSs on different nodes collaborate by using security data to obtain complementary information from collaborating hosts, and intrusion alerts to inform others of a locally-detected intrusion. The LIDS agent could use either anomaly or misuse detection. Our approach differs from LIDS in that LIDS is a cooperative approach and the decision is based on the data collected from collaborating nodes. Therefore, LIDS will not function well if compromised nodes broadcast false intrusion-related information.

An intrusion detection architecture based on a static stationary database has been proposed by Smith [50], where each node in a MANET has an IDS agent running on it. The IDS agents on each node work together via a cooperative intrusion detection algorithm to decide when and how the network is being attacked. The architecture is divided into two parts, the mobile IDS agents that work on each node and the stationary secure database that has the signatures of all known misuse attacks and normal activity patterns of each node. This approach used both anomaly and misuse detection. Nevertheless, with a centralized database, the mobile nodes need to move and physically connect to the database periodically to stay up-to-date with the intrusion information. Also, the stationary secure database can become a single-point-of-failure if it is compromised. Our approach does not require the mobile nodes to physically move and connect to the MA server for update and can have multiple MA servers deployed in the network to avoid a single centralized server as an attack target.

Kachirske and Guha [31] also proposed a distributed IDS for wireless ad hoc networks based on the MA technology. By efficiently merging audit data from multiple network sensors, they analyzed the entire network for intrusions and tried to thwart intrusion attempts. They suggested not to use every possible node as IDS, but those chosen by a special "clustered network monitoring node selection algorithm." Designated nodes decide on intrusion. To avoid malicious votes, a modified independent decision-making system is used. This system uses a state machine for each known node, which is updated with threat information gathered by the monitoring agents. When a certain level or threat is reached, a command is sent to the node in danger, requesting necessary actions. Our approach has local IDS on each node and differs from the IDS in [31] where the monitoring and decision agents are on different nodes, thus requiring more communication and coordination.

3.8 Conclusion

We have proposed a distributed MA-based application-layer IDS framework for MANETs. The IDS utilizes both anomaly and misuse detection to identify attacks in MANETs. Also, the MAs augment each node's intrusion detection capability in the network by updating attack signatures and normal application profiles, patching and installing programs, further analyzing and diagnosing each node, and verifying the integrity of the IDS agents on each node. We have completed the design of the IDS architecture and the overall network structure, and described the design of protocols using MAs for the IDS. Our evaluation has demonstrated trade-offs between different design parameters of MAs.

CHAPTER IV

TGIS: Booting Trust for Secure Information Sharing in Dynamic Group Collaborations

4.1 Introduction

Over the past decade, mobile computing has drawn much attention because of the various types of mobile devices (e.g., smartphones) and communication mechanisms (3G/4G and WiFi) that are becoming available. Even though mobile devices have been increasingly used for entertainment and social applications, there has been an important oversight: more mobile applications will appear at public safety, health-care, and even military facilities/sites. For example, the military has started to use smartphones in the battlefield for communication and collaboration purposes. Of these trends, in this chapter we explore the use of mobile devices for dynamic and ad-hoc group communications. For example, soldiers from different units may form a group for a particular task, or, agents of local police offices, homeland security, and the FBI may dynamically react to an accident in a local area.

There are several ways for mobile devices to communicate in an ad hoc dynamic group. For example, an online service can be deployed, and all devices can communicate with variant public network conductivities, e.g., with 3G/4G or WiFi. The service implements basic group management mechanisms such as user registration and group creation. Alternatively, mobile router or base station technologies such as Femtocell [1] can allow communication between mobile devices within a geographical local area. For example, it is necessary to communicate using mobile routers when military soldiers are in forests or wilderness for military operations since no base stations will exist. Even if there are existing base stations, the military will still choose to use their own mobile routers or tactic radios for security purposes.

Several security requirements are desired for dynamic group communication and information sharing with mobile devices. First of all, devices must securely communicate and collaborate with one another within a group. That is, the information shared in one group must only be accessible to its group members, e.g., authorized by a group leader. Furthermore, even within a group, information may not be freely shared by all group members, e.g., due to different security levels, expertise, and job duties. In addition, cross-group communication is necessary in many scenarios. For example, one soldier in a military team may want to request assistance from another team on identifying some weapons, while not wanting to share the information with every soldier in that team. We find this a common requirement in many dynamic group communications such as healthcare and first responders (cf. Section 4.2 for more detailed description). All of the above-mentioned security requirements require trust management between group members. Also, in the mobile computing environments, there are many services that provide confidential information to clients that have various access rights. Therefore, trust, security, and access control are necessary for mobile devices collaboration.

Recent years have seen increasing reliance on mobile devices for organizational usage, such as in enterprises, government agencies, and military units. Aiming to bootstrap trust for dynamic group based information sharing and access control, we propose and implement trusted group-based information sharing (TGIS). TGIS is

[1] `http://www.femtoforum.org/femto/`

a distributed security protocol built upon existing trust infrastructures in individual organizations to enable trust management for group collaborations. We assume that each device belongs to one organization, which has implemented mechanisms to deploy credentials for trust management of users within its organization. We then leverage the user identity hierarchies to establish trust between group members by exploiting hierarchical identity-based encryption (HIBE) [24]. Specifically, a group leader can use a user's hierarchical identity as a public key to distribute group keys. For controlling information sharing within a group, we use attribute-based encryption (ABE) [7] for secure access control, whereby the group leader defines group-wide attributes, generates attribute secret keys, and distributes them to individual group members. By exposing public information of a group in an authentic manner, other out-group users can also send information to in-group users with controlled sharing, i.e., by specifying a user with the particular attributes to access the information.

Mobile devices have some limitations that make it very challenging to establish trust relationship among devices. First, as mobile devices are often required to be small, light, and easy to hold, this puts strict limitation on the devices resources. Thus, traditional trust management mechanisms might not be able to perform effectively on mobile devices. Also, mobile devices form dynamic collaborative relationships because of their mobility. Therefore, there must be a distributed method for trust establishment among devices. Furthermore, devices that belong to different domains require different collaboration and access rights. Our goal is therefore to consider these limitations when we design our protocol.

TGIS defends against passive attacks such as eavesdropping and active attacks such as replay, spoof, drop, or insert false data and impersonation attacks by using encryption and digital signature with mobile devices' organizational keys. TGIS can also prevent attackers and legitimate group members from accessing shared information without proper privileges.

4.1.1 Organization

The remainder of this chapter is organized as follows. We first present the motivation and applications of TGIS in Section 4.2. In Section 4.3, we review the cryptographic primitives in TGIS. We then give an overview of TGIS design in Section 4.4 and describe the details of TGIS protocol in Section 4.5. Section 4.6 presents how to distribute the group leader's tasks of group private key generation and distribution. Section 4.7 analyzes the security of TGIS, which is followed by our implementation and performance evaluation in Section 4.8. We discuss the challenges faced in designing a TGIS application user interface in Section 4.9. Finally, we discuss related work in Section 4.10 and conclude this chapter in Section 4.11.

4.2 Motivation

The main goal of TGIS is to let users use their mobile devices to establish a trust relationship to collaborate and communicate with their collaborators, and to have access control over their shared information among their collaborators. There are many real life scenarios that would require TGIS for trust management and dynamic group collaboration. Below we elaborate on the motivational examples of first responders in emergency rescue events and military soldiers in battlefields.

First responders are trained rescuers that would go to emergency scenes and perform search and rescue. For example, the police departments, fire departments, emergency medical services (EMS), and the Department of Homeland Security (DHS) are all considered to be first responders in the events of large-scale emergencies. First responders collaborate in emergency rescues, and many of them are from different organizations. For them to cooperate and share information in the rescue scenes, they will need to boot trust and establish trust relationship between one another. As different organizations have different privileges, they will need the dynamic group collaboration protocol to communicate securely either within or between different rescue groups.

Take the example of an earthquake first responders disaster search and rescue. EMS, fire departments, police departments, and DHS agents will all collaborate to assist with the recovery efforts. TGIS allows the first responders from different organizations to collaborate in a secure manner and establish trust relationship with one another. The first responders will be able to authenticate themselves and join a dynamic collaboration rescue team for disaster relief. On the other hand, news reporters or other curious people may not be able to successfully authenticate themselves to join the rescue team and access the shared information. In the rescue team, there are classified information that only people with the appropriate level of clearance and access can view. Such information might only be accessible to polices or DHS agents, and TGIS also helps with data access control in such scenario.

Another application of TGIS is for the collaboration of military soldiers in the battlefield. Since soldiers need to collaborate and form dynamic coalition in battlefields for ad hoc military tasks, they need trust management and dynamic group collaboration. Soldiers that form dynamic teams to perform tasks have different clearance levels for the information shared among team members, so secure communication and information access control are required in the coalition. And soldiers in different coalitions can also share information based on their clearance. For example, in a military task where a scout team is sent out to detect if there are weapons or mines in the battlefield, or to eavesdrop or intercept on the enemy's conversation, the scout team would consists of weapon specialists and soldiers with different ranks. TGIS can be used for the scout team members to form a dynamic collaboration coalition using their devices or mobile devices. It would save device processing time and power if the devices in the scout team can collaborate or offload heavy computation programs to other devices without their information been eavesdropped by the enemies. With

TGIS, team members can share information to other members with the appropriate level of clearance and help execute programs for other members, like help to execute the translation program on the eavesdropped enemy conversation.

As for inter-group collaboration, group members may want to share information between users in different groups. First responders share information about different scenes of accidents, and only members with the appropriate clearance can read certain classified information. Military soldiers exchange information in the battlefield regarding specific tasks, and only soldiers with the appropriate clearance to the task can read the information. In our scout team example, weapon specialists may exchange related intelligence between different scout teams and only weapon specialists are authorized to view any confidential information. Our proposed TGIS protocol is designed to be used in all of the above scenarios.

4.3 Cryptographic Primitives

In this section, we review the cryptographic primitives we use for the construction and design of TGIS.

4.3.1 Hierarchical Identity-Based Encryption

Identity-based cryptography (IBC) was first introduced by Adi Shamir in 1984, who implemented identity-based signature (IBS) and proposed identity-based encryption (IBE) in [47]. However, IBE was not realized until 2001 by Boneh and Franklin [9], and then by Cocks [15]. IBC is a type of public-key cryptography. In IBC, a public identity is used as a public key string to simplify certificate management in public key infrastructure (PKI). The public identity could be an email address, phone number, or a hierarchical identity within an organization. IBC is different from traditional PKI, where an entity (e.g., a user or a host) generates its public/private key pair and obtains public key certificate from a certificate authority

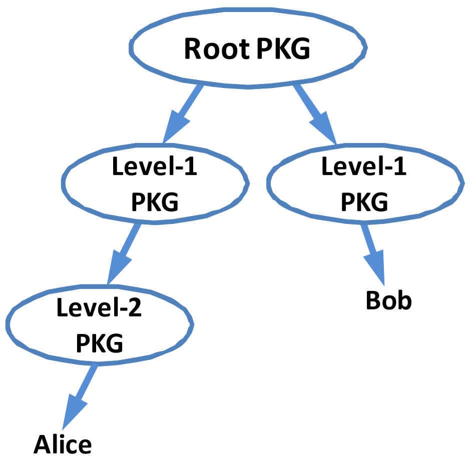


Figure 4.1: HIBE system architecture.

(CA). In IBC, the private key is generated by a trusted third party called the private key generator (PKG) with its corresponding identity and system parameters.

More specifically, in an identity-based system, a PKG generates a master secret key (MSK) and public system parameters (SP). The MSK is kept as a secret and used by the PKG only to generate corresponding private keys for individual users, and the SP is published publicly. Any user can use the published SP and the publicly known user identity to generate public keys for other users.

Based on IBE, hierarchical identity-based encryption (HIBE) [24, 10] was introduced to create hierarchies of PKGs and allow higher-level PKG to control the keys given to its subordinate lower-level PKGs. HIBE allows root PKG to distribute private key computation workload to lower-level PKGs and ease the private key distribution problem and improve scalability. It also removes a single-point of failure and the disclosure of a lower-level PKG's secret will not compromise higher-level PKGs' secrets or other parts of the hierarchy. HIBE system architecture is shown in Figure 4.1.

HIBE consists of five algorithms: `Root Setup`, `Lower-level Setup`, `Extract`, `Encrypt`, and `Decrypt`.

- `Root Setup`: The root PKG runs this algorithm that takes in a security parameter as input and generates the master secret (MS) and public system parameters (SP). The MS is kept secret by the root PKG and SP is public.

- `Lower-level Setup`: The lower-level PKG runs this algorithm and picks a random secret (S), which is kept secret by the PKG.

- `Extract(`PrK, S, ID`)`: A PKG runs this algorithm that takes in as inputs its private key PrK, secret S, and a subordinate user identity $ID = (ID_1, ID_2, ..., ID_l)$ if the user is in level l. The algorithm generates a private key (PrK) for the corresponding user ID.

- `Encrypt(`SP, ID, M`)`: An encrypter runs this algorithm that takes in the system parameters SP, the receiver's ID, and a message M as inputs and generates a ciphertext C of M.

- `Decrypt(`PrK, C`)`: A decrypter runs this algorithm that takes in its PrK and the ciphertext C as inputs and returns the message M if M is encrypted with its ID.

4.3.2 Attribute-Based Encryption

Attribute-based encryption (ABE) enables complete access control on encrypted data by specifying the expressive access policies/rules in private keys and ciphertexts [7, 25]. There are two categories of ABE, the ciphertext-policy ABE (CP-ABE) [7] and key-policy ABE (KP-ABE) [25]. In CP-ABE [7], the private keys are associated with a set of attributes, and messages are encrypted to access policies which specifies what private keys with the desired attributes will be able to decrypt the ciphertexts. Whereas in KP-ABE [25], it's the ciphertexts that are associated with sets of attributes and the private keys are associated with the access policies.

We use CP-ABE in TGIS for ensuring access control in data sharing. In CP-ABE, a user will specify an access tree structure of access policy for the message to be encrypted. Only if another user with a private key that is associated with the desired attributes will be able to decrypt the ciphertext.

CP-ABE consists of four algorithms: `Setup`, `Encrypt`, `KeyGen`, `Delegate`, and `Decrypt`.

- `Setup`: This algorithm takes in a security parameter as input and generates the master key (MK) and public key (PK). The MK is kept secret by the authority party and PK is made public.

- `Encrypt(`PK, T, M`)`: An encrypter runs this algorithm that takes in the public key PK, a tree access structure T that defines the access policy, and a message M as inputs and generates a ciphertext C of M.

- `KeyGen(`MK, S`)`: This algorithm takes in a set of attributes S and the master key MK and generates a secret key SK that is associated with the corresponding set of attributes S.

- `Decrypt(`SK, C`)`: A decrypter runs this algorithm that takes in its SK and the ciphertext C as inputs and returns the message M if the attributes of SK satisfies the access tree T used for encryption.

4.4 Overview of TGIS

Trust relationship between entities indicates that an entity has certain assurance that it can share data with another entity without releasing information to any other entity. This is typically achieved by identity authentication, shared keys, and data encryptions. To establish trust relationship among users, we first bootstrap trust in dynamic groups for secure collaboration and communication, and then enforce access control for data sharing. To bootstrap trust among group members, we leverage the existing organization identity hierarchies to establish trust between group members and let a group creator/leader generates the private keys for group members and securely distributes the keys using HIBE [24]. Since group members have different privileges, we use CP-ABE [7] for secure access control within a group and also among different groups.

Below we present the system architecture and assumptions, the attack models, and the design overview for TGIS.

4.4.1 System Architecture and Assumptions

Our system consists of users carrying mobile devices for communication and collaboration with other users in the network. Each user belongs to an existing hierarchical domain organization. The identity of a user/device is a hierarchical domain structure and is unique. For each user, the user's hierarchical identity is the concatenation of the identities from the root to the user. For example, Alice in the Surveillance Unit in the Police Department will have "Police/Surveillance/Alice" as her ID.

Here we use HIBE as described in Section 4.3.1 for the security basis. We assume each domain/organization has a hierarchical architecture and each intermediate user is a private key generator (PKG) that is responsible for assigning private keys for its subordinate users. The intermediate users are different levels of managers or authorizers in an organization. The top level is the root PKG that is responsible for generating the public known system parameters (SP). Each user gets its private key from their immediate upper level PKG. There is no private/public key pair for each user and instead, user identity is used as the public key in HIBE. A nice property of HIBE is only the domain SP and user ID is needed in order for others to generate the user's public key. It is very flexible and one does not need to know the user's intermediate PKG's public key to generate the user public key.

The system architecture of our proposed TGIS protocol is shown in Figure 4.2. And Figure 4.3 is an example of the system architecture using The University of

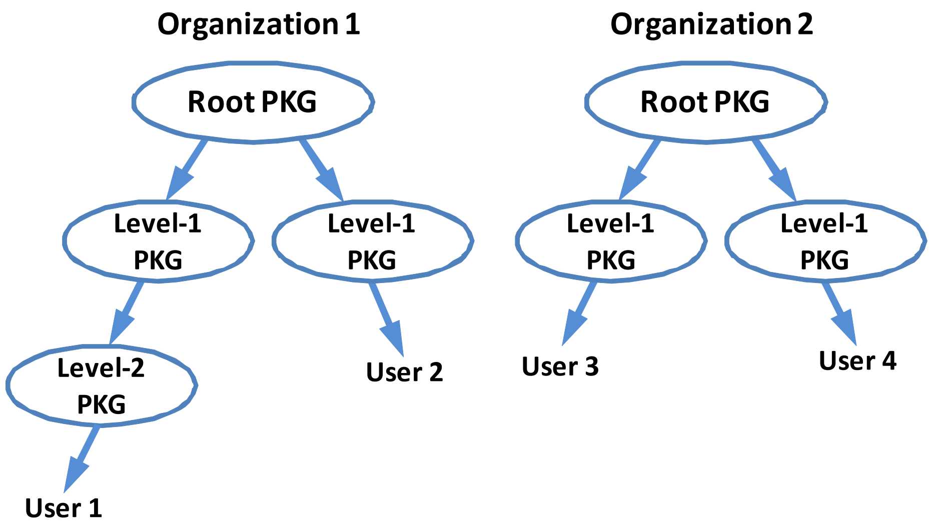


Figure 4.2: TGIS System architecture.

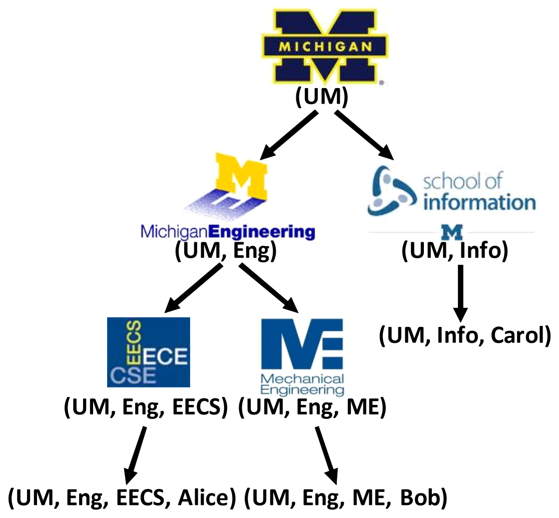


Figure 4.3: TGIS System architecture using The University of Michigan as an example of a hierarchical organization.

Michigan as the hierarchical organization.

We assume users create dynamic groups for different events and purposes. In a dynamically formed group, the group creator (or leader) controls access to data and user privileges in the group and generates corresponding group private keys for each group member. Users can share information with other members in the same group or even with users in other groups. Group members can be from different organizations

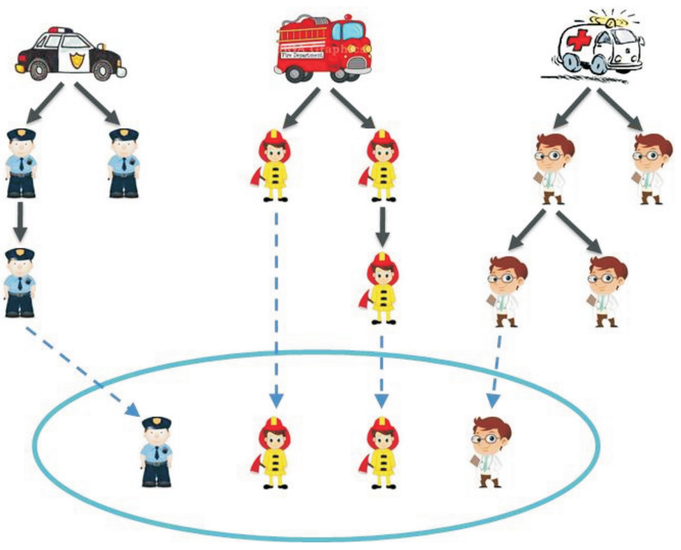


Figure 4.4: System architecture of users in different organizations forming a dynamic group.

to form a dynamic group. Therefore, the group leader acts as the PKG of the group to generate and distribute group private keys for other members.

We assume that when users communicate in a group, they can either use the existing base stations or setup mobile routers when they are in the wilds and no base stations are available. For example, femtocell, which is a small cellular base station designed for use in a home or small business, can also be setup for group communication. Figure 4.4 presents the system architecture of users in different organizations forming a dynamic group using a mobile router.

4.4.2 Attack Model

In our attack model, attackers can eavesdrop on the communication channel between users and can also replay, spoof, or insert false data into the network. Also, attackers can masquerade as legitimate users to join group collaboration. Furthermore, an attacker can be a group member such that it tries to access and propagate data without proper privileges. The attacker can also launch Sybil [20] attack and fake to be multiple identities in the network.

4.4.3 Design Overview

To establish a trust relationship between devices and let devices form dynamic collaboration groups with proper privileges assigned to group members, we propose to exploit hierarchical identity-based encryption (HIBE) [24] and ciphertext-policy attribute-based encryption (CP-ABE) [7] in TGIS.

We use HIBE as the security basis, where users' devices receive private keys from their belonged domain PKGs, and their identities are used as public keys for communication. HIBE binds public keys with respective user identities and allows users to bootstrap secure collaboration protocols. Also, the benefit from using identity-based encryption (IBE) is the users can form secure channels with only user identities, and there is no need to retrieve public key certifications before sending secret data to a particular user. That is, a group leader only needs to know the SP of a domain and a user's identity in order to distribute group keys without verifying the user's public key certificate as in the traditional PKI scheme.

To allow users to form dynamic groups and share sensitive data, we use CP-ABE to manage data access control within and between groups, where the group leader defines group-wide attributes corresponding to individual users' privileges. CP-ABE allows users to apply fine-grained access control policies on data by assigning attributes to secret keys and assigning an access tree, a logical expression over attributes, to the data upon encryption.

4.5 TGIS Protocol

TGIS is a protocol used for users to establish trust relationship with other users using their mobile devices. By establishing trust relationship with one another, users can form dynamic collaborative groups and share information securely inside the group with proper data access control.

The TGIS protocol consists of five phases, the offline domain setup, group setup, user enrollment, intra-group communication, and inter-group communication. This section explains the details of each. Listed below are the notations used in the rest of the chapter.

- $a, b, \ldots$ are entities such as users/devices.
- ID_a is the identity of user a.
- $A, B, \ldots$ are domains/organizations or dynamic groups.
- HSP_A is the HIBE system parameters for domain A.
- HMS_A is the HIBE master secret for domain A.
- $A.HPK_a$ is the HIBE public key for user a in domain A.
- $A.HPrK_a$ is the HIBE private key for user a in domain A.
- $AMSK_A$ is the ABE master secret key for group A.
- APK_A is the ABE public key for group A.
- $A.ASK_a$ is the ABE secret key for user a in group A.
- S_A is the attributes set of group A.
- $A.attr_a$ is the attributes set assigned to user a in group A, and $A.attr_a \subset S_A$.
- T is the access tree built with logical expression over attributes.

The system architecture is as described previously in Section 4.4.1, where each user belongs to a pre-established hierarchical domain and has a unique identity that is assigned from the domain. The TGIS protocol consists of five phases, the offline domain setup phase, the group setup phase, the user enrollment phase, the group communication phase, and the inter-group collaboration phase. Below we present each phase in detail.

4.5.1 Domain Setup

Before deployment, the mobile devices are in the offline domain setup phase, and all devices are assumed to be secured. In this phase, each user registers its device with an identity in its own hierarchical domain and receives the domain HIBE private key. Each domain root PKG generates the domain HSP and HMS. HSP is made public and is used for generating HIBE public keys (HPK) together with user identities. HMS is kept secret by the domain root PKG and is used for generating HIBE private keys ($HPrK$) for users. Each user has a $HPrK$ that is generated and assigned by its parent PKG. The detailed protocol for domain setup with HIBE private and public key generation and distribution is listed as the following.

DomainSetup(Root PKG $r \in$ Domain D)

r:	RootSetup(Domain D) $\rightarrow HSP_D, HMS_D$
u_{t-1}:	ExtractHIBEKey($D.HPrK_{u_{t-1}}, D.S_{u_{t-1}}, ID_{u_t}$) $\rightarrow D.HPrK_{u_t}$
	for user $u_t \in Level_t$ that is u_{t-1}'s child
$u_{t-1} \rightarrow u_t$:	$D.HPrK_{u_t}$
u_1:	CreateHIBEPubKey(HSP_D, ID_{u_2}) $\rightarrow D.HPK_{u_2}$

In this phase, a domain D root PKG r generates the domain HSP HSP_D and HMS HMS_D. The root PKG and each lower-level PKG generates and distributes $HPrK$ for its child users with the users' identities and the PKG's own secret and private key. User public key HPK can be generated by anyone using HSP_D and the user identity.

Example: The Michigan State Police Department *PoliceStateMI* is the root PKG for the police departments in Michigan. *PoliceStateMI* generates HSP_P and HMS_P for the entire Michigan State Police Department and its subordinate bureaus.

The city police departments in Michigan are the level-1 PKGs and receive private keys from $PoliceStateMI$. Alice is a police officer in the Ann Arbor City Policy Department $PoliceCityAA$. Therefore, Alice's ID is $PoliceStateMI/PoliceCityAA/Alice$. $PoliceStateMI$ generates $P.HPrK_{PoliceCityAA}$ for the Ann Arbor City Police Department using its private key $P.HPrK_{PoliceStateMI}$, its secret HMS_P, and the Ann Arbor City Police Department ID $PoliceStateMI/PoliceCityAA$. $PoliceCityAA$ generates $P.HPrK_{Alice}$ for Alice using its private key $P.HPrK_{PoliceCityAA}$, its secret $P.S_{PoliceCityAA}$, and Alice's ID $PoliceStateMI/PoliceCityAA/Alice$. One needs to know HSP_P and Alice's ID in order to generate Alice's public key $P.HPK_{Alice}$ and encrypt a message for Alice.

4.5.2 Group Setup

After the offline domain setup phase, users carry their devices with installed domain private keys. In an event that requires user collaboration and dynamic group setup, the users enter the group setup phase. In this phase, a group leader creates a group and generates group parameters and keys.

During group setup, the group leader generates the group APK, $AMSK$, and defines the privileges/attributes set S of the group. APK and S are made public to the network and $AMSK$ is kept secret by the group leader. The detailed protocols for group setup is listed as the following.

```
GroupSetup(Group Leader $l \in$ Domain $D$)

$l$:          CreatGroup(Group $G$) $\rightarrow APK_G, AMSK_G$, Attributes set $S_G$

$l \rightarrow u$:    DistributeGroupAPK($D.HPrK_l, APK_G, S_G$) $\rightarrow K$
              where $K = \{APK_G, S_G, sign(APK_G, S_G)_{D.HPrK_l}\}$
$u$:          CreateHIBEPubKey($HSP_D, ID_l$) $\rightarrow D.HPK_l$
$u$:          RetriveGroupAPK($D.HPK_l, K$) $\rightarrow APK_G, S_G$
              if $sign(APK_G, S_G)_{D.HPrK_l}$ is verified by $D.HPK_l$
```

In this phase, when a group of users want to form dynamic trust collaboration, a group leader l generates a group G and group keys APK_G and $AMSK_G$ and a set of group attributes S_G. APK_G and S_G are made public in clear text, but l signs the message using its HIBE private key $D.HPrK_l$ for authentication purpose. Group members can use l's HIBE public key $D.HPK_l$ to verify the message.

Example: When police officers are in a rescue mission and need to create a rescue team with other first responders, Alice in the police department becomes the group leader and creates the rescue team A-team. Alice generates $AMSK_{A-team}$, APK_{A-team}, and attributes set $S_{A-team} = \{security\_level, profession\}$ to represent levels of information clearance. And attribute values are $security\_level$ ={top_secret, secret, public} and $profession$ ={general, medical, detective}. Alice signs APK_{A-team} and S_{A-team} with $P.HPrK_{Alice}$ and publishes the A-team public parameters to the A-team.

4.5.3 User Enrollment

After the group setup phase, a group is created by the group leader and users enter user enrollment phase to join a group. Upon accepting a user's request to join a group, the group leader decides what kind of privileges that the user can have, and assigns the user some attributes *attr* that correspond to the privileges. The assigned attributes $attr$ are attributes in S. The group leader then assigns an ASK for the user based on $attr$. The ASK is distributed to the user encrypted with its HPK, which is generated from its identity as described in Section 4.5.1. Thus, only the intended receiving user with the matching $HPrK$ can decrypt the ciphertext and receive its assigned ASK. The detailed protocol for user enrollment is listed as the following.

UserEnrollment(Group Leader $l \in$ Domain D)

$u \to l$:	RequestJoinGroup(G) where $u \in$ Domain E
l:	AssignAttrToMember(u) $\to G.attr_u$ where $G.attr_u \subset S_G$
l:	CreateMemberKey($u, AMSK_G, G.attr_u$) $\to G.ASK_u$
l:	CreateHIBEPubKey(HSP_E, ID_u) $\to E.HPK_u$
$l \to u$:	DistributeMemberKey($E.HPK_u, G.ASK_u$) $\to K = \{G.ASK_u\}_{E.HPK_u}$
u:	RetrieveMemberKey($E.HPrK_u, K$) $\to G.ASK_u$

In this phase, for each group member u in group G, the group leader l assigns it the corresponding attributes $G.attr_u$. Then l generates the group private key $G.ASK_u$ for u which binds $G.attr_u$ with $AMSK_G$. Next l distributes $G.ASK_u$ to u by encrypting it with u's HIBE public key $E.HPK_u$ generated from u's ID and sending it to u. Only u can decrypt the ciphertext with its HIBE private key $E.HPrK_u$ and receive $G.ASK_u$ sent by l. Note that l and u don't need to belong to the same domain.

Example: When a fire fighter Bob wants to join the rescue team A-team created by Alice, he sends $RequestJoinGroup(A - team)$ to Alice. Alice then grants Bob membership and decides he has a low-level clearance so she grants him $A - team.attr_{Bob}$ ={public, general} and generates $A - team.ASK_{Bob}$ with Bob's clearance level. Alice distributes $A - team.ASK_{Bob}$ to Bob by encrypting the key with Bob's HIBE public key $F.HPK_{Bob}$, which is generated by Bob's ID and the fire department HSP_F. Bob can retrieve the member key by decrypting the message with his HIBE private key $F.HPrK_{Bob}$.

4.5.4 Intra-group Communication

After the user enrollment phase, group members can have secure group communication and collaboration using their ASK and APK. Group members can encrypt data to be shared with flexible and expressive policies defined by access tree structures. Only users with the required attributes/privileges can decrypt the ciphertext and access the shared data.

In the group communication phase, ABE construction ensures that only group members with the corresponding attributes are able to decrypt data. ABE keys guard access to user data and group member u that encrypts message M to ciphertext C controls which attributes can decrypt C. Specifically, u uses the group APK and an access trees T to encrypt M for members with matching attributes. Only members with ASK that satisfies T can decrypt C and read M. The detailed protocol for intra-group communication is listed as the following.

IntraGroupComm(user $u_1 \in$ Group G, $u_2 \in G$, Message M)	
$u_1 \rightarrow u_2$:	ABEEncrypt(APK_G, M, Access Tree T) $\rightarrow$ Ciphertext C
u_2:	ABEDecrypt($C, G.ASK_{u_2}$) $\rightarrow M$ only if $G.Attr_{u_2}$ satisfies T

Example: When Alice wants to share her location with members of A-team, she encrypts it with APK_{A-team} and the access tree T ={public} since her location is a low-level clearance information. Bob can decrypt Alice's shared message and read her location with his member key $A-team.ASK_{Bob}$ since $A-team.attr_{Bob}$ ={public, general} satisfies T.

4.5.5 Inter-group Collaboration

For groups that want to collaborate and share information, secure communication between groups can be done in a similar way as intra-group communication. The users in one group request for the APK and attributes set S of the collaborating group from its group leader for data encryption and access control.

As shown below, for group collaboration, members in Group B need to know Group A's APK_A and attribute set S_A. Members in B use APK_A and an access tree T created from S_A to encrypt data for A's group members. Members in A are able to decrypt the shared data using their ASKs with matching privileges. The detailed protocol for inter-group collaboration is listed as the following.

```
**InterGroupComm**(Group Leader $l_a \in$ Group $A$ and Domain $D, u_b \in$ Group $B$)
$u_b \to l_a$: GetGroupAPK($ID_{u_b}$)
$l_a \to u_b$: DistributeGroupAPK($D.HPrK_{l_a}, APK_A, S_A) \to K$
        where $K = \{APK_A, S_A, sign(APK_A, S_A)_{D.HPrK_{l_a}}\}$
$u_b$: CreateHIBEPubKey($HSP_D, ID_{l_a}) \to D.HPK_{l_a}$
$u_b$: RetriveGroupAPK($D.HPK_{l_a}, K) \to APK_A, S_A$
        if $sign(APK_A, S_A)_{D.HPrK_{l_a}}$ is verified by $D.HPK_{l_a}$

$u_b \to u_a$: ABEEnrypt($APK_A, M, T) \to C$ for $u_a \in A$
$u_a$: ABEDecrypt($C, A.ASK_{u_a}) \to M$ only if $A.Attr_{u_a}$ satisfies $T$
```

Example: When the rescue team $B - team$ members would like to share information with $A - team$ members, they need to receive $A - team$'s APK_{A-team} and S_{A-team} from its group leader Alice. Then the $B - team$ members can encrypt the missing people list with APK_{A-team} and access tree T ={top_secret} and send the message to $A - team$. Bob in $A - team$ can not decrypt the message since $A - team.attr_{Bob}$ ={public, general} does not satisfy T. But members in $A - team$

with high-level clearance can decrypt the message and read the missing people list.

4.5.6 Message Authentication

We would like to note that to achieve message authentication and integrity for messages exchanged in intra-group and inter-group communications, we propose to use identity-based signatures (IBS) [27]. IBS allows the users to sign messages with their own identities and achieves message authenticity. IBS combined with HIBE can be viewed as a complete package to provide authenticity, integrity, and confidentiality.

4.6 Decentralizing TGIS

We can further apply multi-authority attribute-based encryption [34] to TGIS so group leaders are no longer necessary to assign attributes and create private keys for group members. By applying multi-authority ABE, any user can assign attributes and create private keys for other members after they have established mutual trust. A user can create his own attributes and create a public key and issue private keys to other group members, thus reducing the risk of a compromised group leader revealing group secrets and member private keys to an attacker.

Multi-authority ciphertext-policy attribute-based encryption (CP-ABE) [34] consists of five algorithms: `Global Setup`, `Authority Setup`, `Encrypt`, `KeyGen`, and `Decrypt`.

- `Global Setup(`λ`):` This algorithm takes in the security parameter λ as input and generates the global parameters GP for the system.
- `Authority Setup(`GP`):` Each authority runs this algorithm with GP as input and generates its own secret key (SK) and public key (PK) pair.
- `Encrypt(`$GP, (A, \rho), \{PK\}, M$`):` An encrypter runs this algorithm that takes in the global parameters (GP), an access matrix (A, ρ), the set of public keys

for relevant authorities $\{PK\}$, and a message M as input and generates a ciphertext C of M.

- `KeyGen(`ID, GP, i, SK`):` This algorithm takes in an identity ID, the global parameters GP, an attribute i belonging to some authority, and the secret key SK for this authority and generates a secret key $K_{i,\mathrm{ID}}$ for user ID associated with the corresponding attribute i.

- `Decrypt(`$GP, \{K_{i,\mathrm{ID}}\}, C$`):` A decrypter runs this algorithm that takes in the global parameters (GP), a collection of keys corresponding to the attribute and identity key pairs all with the same identity $\{K_{i,\mathrm{ID}}\}$, and the ciphertext C as input and returns the message M if the collection of attributes i satisfies the access matrix (A, ρ) used for encryption.

When multi-authority CP-ABE [34] is used in TGIS, in the group setup phase, a group leader only needs to generate and distribute the global parameters (GP). In the user enrollment phase, when a user wants to join a group, no attributes or private keys are assigned to the user from the group leader. Instead, attributes and private keys are issued by individual group members. The group members that assign attributes and private keys to others are called the authorities. Authorities will assign attributes and distribute private keys for other members after they have established mutual trust. No global coordination exists between authorities in a group, and authorities might not even be aware of each other.

During the intra-group communication phase, a group member encrypts messages with GP, an access matrix, and the set of public keys belonging to the relevant authorities for the used attributes. Only a group member with the collection of corresponding private keys that satisfies the access matrix can decrypt the message. As for inter-group collaboration, users in other groups will need to know the attributes a group uses in order to share access-controlled information with the group.

The multi-authority CP-ABE [34] is proved to be collusion resistant. The authors use a hash function on the user's identity to manage collusion resistance across multiple key generations issued by different authorities. A user's identity is bound with the various attributes assigned to him so those attributes cannot be used with another user's attributes and collude in decryption.

4.7 Security Analysis

We first discuss the network survivability in the event of mobile device compromises, and then analyze the security of TGIS against the attack models presented in Section 4.4.2.

4.7.1 Network Survivability

After compromising a mobile device, an attacker can discover the keys in the device, such as the domain HIBE keys and group ABE keys. If the compromise of the devices is undetected, we need to analyze the survivability of the network, or the ability of the network to maintain an acceptable level of performance under device compromises. For this, we consider the general attacks an adversary can mount after it compromises a device.

In TGIS, each mobile device is loaded with a domain HIBE private key in the offline domain setup phase. If a device is compromised and its HIBE private key revealed to the attacker, then the attacker can fake to be that device and join groups as the identity of the compromised device. If the compromised device is a parent user in the hierarchical domain, then the HIBE private keys of its children devices can also be computed. However, the HIBE private keys of other devices that belong to different parts of the hierarchy remains unknown to the attacker.

In HIBE, the private key of a user depends on the secret that is only known to its parent user, so no ancestor other than the parent can compute the user's private key. However, the user's ancestors can still decrypt the messages sent to the user. Therefore, if an attacker compromised a device and have its HIBE private key, it can also decrypt the messages sent to its descendants. Gentry and Silverberg presented in [24] a mechanism to restrict key escrow using dual-HIDE which does not allow the ancestors of a user to decrypt its messages.

When an attacker compromises a device and gets the group ABE key stored in the device, it can successfully decrypt some of the group messages that the compromised device belongs to. For the group ABE keys that are revealed to the attacker, since each user has different attribute keys assigned even when they have the same set of attributes, the revealed keys can only decrypt the messages that are encrypted with the corresponding policies, and the attacker can only read the messages that belongs to the compromised device's privilege level.

4.7.2 Defense Against the Attack Model

We now describe how TGIS can defend against various attacks in group collaboration and trust management.

4.7.2.1 Defense Against Passive Attacks

In TGIS, whenever a group leader wants to announce the group public key and attributes set, it sends the message in plain text, along with its digital signature using its HIBE private key. Even though an attacker can eavesdrop on the message, the parameters are meant to be public to the network so the attacker cannot gain any benefit from getting the message.

For group ABE key distribution, the group leader encrypts the keys with the designated user's identity, and only the user has the corresponding HIBE private key can decrypt the message and receive the ABE key. As for group and inter-group communication, all the messages are encrypted with the group public key and access policies. Only users that have the corresponding group ABE keys that satisfy the access policies can decrypt the messages.

4.7.2.2 Defense Against Active Attacks

To prevent an attacker from spoofing or inserting false data in the network, we use identity-based signature (IBS) [27] as noted earlier in Section 4.5. IBS combined with HIBE can achieve authenticity, integrity, and confidentiality. Replay attacks and faulty devices that keeps sending non-verifiable signatures can be detected with intrusion detection systems.

Sybil attack is a kind of attack where a Sybil node illegitimately fakes multiple identities in the network. TGIS withstands Sybil attacks because each user needs to have a HIBE private key corresponding to its identity in order to join a group and receive the group ABE key. Since each user is preloaded with a HIBE private key in the offline domain setup phase, no attacker can pretend to be another user and has the corresponding HIBE private key unless compromising multiple devices. Thus, no attacker can masquerade as legitimate users to join group collaboration if it didn't compromise a device and hold a HIBE private key corresponding to a user identity.

When an attacker compromises a device and successfully joins a group using the device identity, it gets assigned some attributes that matches its privilege level by the group leader. So even when the attacker belongs to a group and receives its group ABE key, it still is not able to access all the data in the group without proper privileges. And with an important property of CP-ABE, even when two users collude with each other in a group and combine their group ABE keys, they still cannot decrypt and get access to the data that should not be readable by their individual attributes.

4.8 Implementation and Evaluation

4.8.1 Prototype Implementation

We have implemented a prototype of TGIS on Android over a local WiFi access network. Our implementation includes a set of Nexus S devices running Android 2.3. We further run an OpenFire XMPP server [2] in the same network for message broadcast. XMPP provides flexible one-to-one and one-to-many communication and push services between online entities with XML format over HTTP.

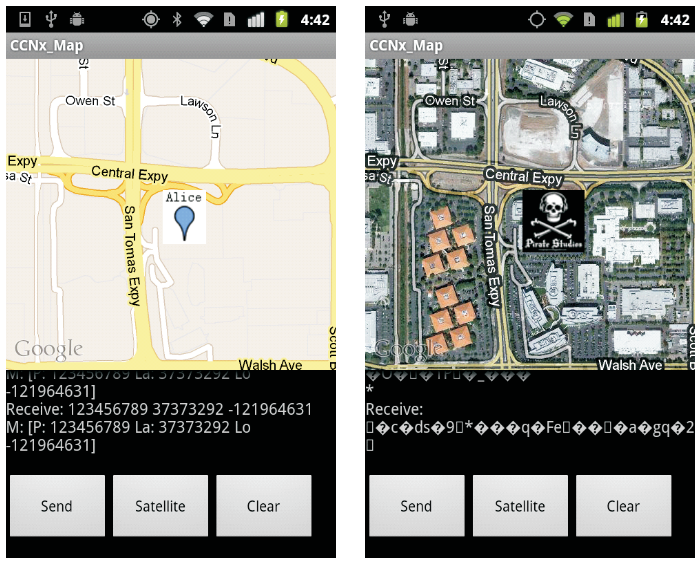
We used the open source pairing-based cryptography (PBC) library [3] to implement HIBE [24] and CP-ABE [7] algorithms. All system parameters, master secret keys, and attributes keys are generated and stored as individual files in the mobile device SD card, which can be shared with data sharing applications.

We implemented a data sharing application over TGIS. The application is a Google Latitude-like [4] location-based application on Android to share a user's location data to others. Upon selecting to share his location, a user selects some pre-defined polices to specify who can access his location data (concatenation of GPS coordinates).The data is then encrypted with the policy and broadcasted to all online group members via the XMPP server. Figure 4.5 shows the snapshots of the location-based application over TGIS running on Android. The left snapshot shows when the application successfully decrypts a location message, and the right snapshot shows when the application failed to decrypt a location message.

4.8.2 Performance

The OpenFire XMPP server acts as both the root PKG and level-1 PKG in our implementation. For the first time the user logins, the client application receives the

[2]`http://www.igniterealtime.org/projects/openfire`
[3]`http://crypto.stanford.edu/pbc`
[4]`https://www.google.com/latitude`



```
M: [P: 123456789 La: 37373292 Lo
-121964631]
Receive: 123456789 37373292 -121964631
M: [P: 123456789 La: 37373292 Lo
-121964631]
```

```
�U��P�_���
*
Receive:
�c�ds�9*���q�Fe���a�gq�2
```

Figure 4.5: Snapshots of the location-based application over TGIS on Android.

private key for the user, which is a one-time operation. Similarly, the group creation and group attribute key distribution are also one-time operations for a single group.

We implemented HIBE with Java Pairing Based Cryptography Library (jPBC) [5], which is a Java porting of the PBC library written in C, and run the evaluation on Nexus S devices running Android 2.3. We measure the processing time for HIBE operations taking on Android. For HIBE encryption, it takes around 1.714 seconds to encrypt a message. This value is the average time to encrypt 30 messages for the size of 50 bytes to 5120 bytes. HIBE decryption averages 0.650 seconds, with IBS signature generation taking an average of 2.034 seconds and IBS signature verification 2.072 seconds. We observed similar performance with CP-ABE. Although the performance seems worse than traditional PKI approach such as RSA encryption and decryption, we believe that mechanisms such as key encapsulation can improve the performance.

[5] `http://gas.dia.unisa.it/projects/jpbc/index.html`

4.9 TGIS Application User Interface

For TGIS applications to be implemented and used on mobile devices in the real world, we must make the user interface straightforward and easy to use. This is especially important as first responders and soldiers are not trained technical personnel and they need to perform information sharing with access control and data encryption in a timely manner.

Designing such an interface can be a challenge as Clark *et al.* have raised some usage problems and difficulties for the P25 radio system in [14]. In [14], Clark *et al.* showed that for the Motorola XTS5000 handheld P25 radio, the outbound encryption is simply controlled by a rotating switch. However, because the rotating switch is located near the channel selector knob, it is easy for the users to accidentally turn off the encryption function when switching channels. Moreover, the encryption indication symbol on the screen, the flashing LED, and the audible warning are all poorly designed and difficult to identify if a radio is in the encrypted mode. The mentioned user interface ambiguities result in easy to transmit in the clear and leak confidential information.

When designing the TGIS application user interface, one guideline would be to allow the encryption function to be always on for all messages. We could employ a default encryption policy to make sure if the users do not specify other access control policies, the data is still encrypted in the default group sharing mode. We could also create a list of possible access control policies for the users to choose from, instead of letting them specify the access tree for their data to be encrypted. It's also nice to have a function to show the effect of the selected access control policies and see who in the group will be able to decrypt the encrypted shared data.

4.10 Related Work

In this section, we review some related work that provides trust management for bootstrapping security in mobile ad hoc networks (MANETs). We also discuss the related work that exploits identity-based encryption (IBE) and attributed-based encryption (ABE) for access and privacy control.

4.10.1 Trust Management

Trust management and security bootstrapping in MANETs is typically difficult to achieve because of the lack of an online trusted entity. There are several papers discussing about trust management and bootstrapping security for MANETs in a distributed manner and without the need of a trusted entity. In [28], Hoeper and Gong introduced two identity-based authentication and key exchange (IDAKE) schemes for MANETS. IDAKE schemes allow two nodes in MANETS to compute a pre-shared secret key for secure communication using their private keys. In [49], Shin *et al.* also proposed solutions for session key establishment between two nodes exploiting pairing-based cryptography. However, all the schemes proposed are geared more toward one-to-one communication between nodes and TGIS allows nodes in the network to have secure group communication.

4.10.2 Access Control

Hengartner and Steenkiste proposed a proof-based access-control architecture that exploits HIBE in pervasive computing [26]. In their scheme, multiple hierarchies are established as policies for access control, and multiple HIBE private keys are used for different policies. Baden et al. proposed Persona, a protocol providing access control for user data over online social networks [5]. Persona uses ABE to allow users to apply access control policies over their data, and let them control who can view their data. ABE are also widely used in cloud computing in providing access control to the data stored in the cloud [53, 56, 58]. In [2], Akinyele *et al.* use ABE to protect electronic medical records on mobile devices and implement a mobile application on the iPhone for secure offline access of medical records. In TGIS, we use CP-ABE for information sharing access control in group communication.

4.11 Conclusion

To bootstrap trust for dynamic group based information sharing and access control, we propose TGIS for dynamic group collaboration and information sharing with mobile devices. TGIS is a distributed security protocol built upon existing trust infrastructures in individual organizations to enable trust management for group collaborations. Specifically, we have shown how HIBE and CP-ABE can be combined to provide trust management and flexible access control in dynamic group collaborations on mobile devices. We have implemented and evaluated TGIS on Android phones and demonstrated its use in different applications. The average time for TGIS to perform a HIBE encryption is 1.714 seconds and decryption is 0.650 seconds on an Android phone. The performance is acceptable since the HIBE operations are invoked infrequently during group setup.

CHAPTER V

Conclusions

Mobile and sensor networks have significantly changed people's lives around the world. As mobile devices proliferate, we are exposed to more risks and attacks than ever before. This dissertation aims to design security and collaboration protocols to create a comprehensive trust framework to protect mobile and sensor networks, and to achieve secure distributed authentication, authorization, and accounting by applying cryptographic algorithms. Its major contribution is in the design of security and collaboration protocols that enable resource-constrained mobile devices and sensors to achieve security and realize information sharing. Our security and collaboration protocols avoid traditional cryptographic approaches intended to be used for resource-sufficient devices, but instead focus on distributed approaches that emphasize the collaboration and cooperation among nodes and devices themselves. This dissertation makes the following contributions.

In Chapter II, we proposed and implemented a distributed authentication protocol called DAPP in wireless sensor networks to allow sensors to authenticate servers without requiring a commonly-used trusted authentication server (AS). Along with DAPP, we also developed a server revocation mechanism for servers to revoke malicious servers detected in the network. The main contribution of this chapter is the development of DAPP to achieve the authentication of servers in a distributed manner without requiring a dedicated and trusted AS. DAPP maintains the distributed nature of sensor networks, and reduces the sensor communication traffic in the network and the energy consumption on each sensor as compared to the case of using a centralized trusted AS for authentication. We also showed that DAPP is robust and secure against various attacks in sensor networks.

In Chapter III, in order to attain security for nodes in mobile ad hoc networks (MANETs), we presented a distributed mobile agent (MA) based intrusion detection system (IDS) architecture at the application layer to help identify malicious nodes in MANETs. The IDS can utilize both anomaly and misuse detection to discover attacks in MANETs. Also, the MAs augment each node's intrusion detection capability in the network by updating attack signatures and normal application profiles, patching and installing programs, further analyzing and diagnosing each node, and verifying the integrity of the IDS agents on each node. We have presented the design of the IDS framework and the overall network structure, as well as the methods for authenticating and dispatching MAs to augment each node's IDS.

Finally, in Chapter IV, to bootstrap trust for dynamic group-based information sharing and access control, we designed a trusted collaboration protocol called TGIS for dynamic group collaboration and information sharing with mobile devices. TGIS is a distributed security and collaboration protocol built upon existing trust infrastructures in individual organizations that enables trust management for group collaboration. We exploit existing organizational identity hierarchies of mobile users to establish trust between group members and further leverage attribute-based encryption (ABE) for secure and flexible access control in dynamic group collaboration on mobile devices. We have implemented and evaluated TGIS on Android phones and demonstrated its use in different applications.

There are several open research problems that can be pursued following this dissertation.

- **Location-Based Information Sharing:** Mobile devices now offer advanced capabilities, often with PC-like functionality, and can run third-party application software. Due to the convenience of mobile devices that provide many different services and their all-in-one characteristic, users can deploy them in meetings and office environments to share confidential information. Therefore, having a location-based information sharing protocol for mobile devices in different scopes of information sharing is important. Moreover, a user should be able to delegate his privileges to other users for accessing the shared information. The protocol should include location-based encryption to ensure the shared information can only be accessed in a certain area and access control to allow information delegation between users. This problem is an effort to protect the network with distributed authentication and authorization.

- **Mobile Malware Detection System:** As with the popularity and rapid evolution of mobile devices such as smartphones and tablets, mobile malware has become a serious threat. With the advanced functionality of mobile devices, more and more users now use their mobile devices for financial and bank transactions, and even for accessing confidential office data. Consequently, attackers are motivated to develop new mobile malware to compromise sensitive user information and breach privacy by tracking individuals' locations. Therefore, it is important to have an efficient and effective mobile malware detection system to classify the malware behavior and propagation vectors, detect the existing and new-generation mobile malware, and respond in real time to deal with the infection. The detection system should be lightweight and energy-efficient for resource-constrained mobile devices.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Alfarez Abdul-Rahman and Stephen Hailes. A distributed trust model. In *Proceedings of the 1997 Workshop on New Security Paradigms*, September 1997.

[2] Joseph A. Akinyele, Matthew W. Pagano, Matthew D. Green, Christoph U. Lehmann, Zachary N.J. Peterson, and Aviel D. Rubin. Securing electronic medical records using attribute-based encryption on mobile devices. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM 2011)*, October 2011.

[3] Patrick Albers, Olivier Camp, Jean-Marc Parcher, Bernard Jouga, Ludovic Me, and Ricardo Puttini. Security in Ad Hoc Networks: a General Intrusion Detection Architecture Enhancing Trust Based Approaches. In *Proceedings of the International 1st Workshop on Wireless Information Systems (Wis 2002)*, April 2002.

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

[5] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: an online social network with user-defined privacy. In *Proceedings of the ACM Conference of Special Interest Group on Data Communication (SIGCOMM 2009)*, August 2009.

[6] Kevin Bauer and Hyunyoung Lee. A distributed authentication scheme for a wireless sensing system. In *Proceedings of the Second International Workshop on Networked Sensing Systems (INSS 2005)*, June 2005.

[7] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of IEEE Symposium Security and Privacy (S&P 2007)*, May 2007.

[8] Carlo Blundo, Alfredo De Santis, Amir Herzberg, Shay Kutten, Ugo Vaccaro, and Moti Yung. Perfectly-secure key distribution for dynamic conferences. In *Proceedings on Advances in Cryptology (CRYPTO 1992)*, August 1992.

[9] Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. In *Proceedings of Advances in Cryptology (CRYPTO 2001)*, August 2001.

[10] Dan Boneh, Eu-Jin Goh, and Xavier Boyen. Hierarchical identity based encryption with constant size ciphertext. In *Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2005)*, May 2005.

[11] Matthew Burnside, Dwaine Clarke, Todd Mills, Srinivas Devadas, and Ronald Rivest. Proxy-based security protocols in networked mobile devices. In *Proceedings of ACM Symposium on Applied Computing (SAC 2002)*, March 2002.

[12] David W. Carman, Peter S. Kruus, and Brian J. Matt. *Constraints and Approaches for Distributed Sensor Security.* Technical Report 00-010, NAI Labs, Network Associates, Inc., September 2000.

[13] Claude Castelluccia, Nitesh Saxena, and Jeong Hyun Yi. Self-configurable key pre-distribution in mobile ad-hoc networks. In *The 4th International IFIP-TC6 Networking Conference*, May 2005.

[14] Sandy Clark, Travis Goodspeed, Perry Metzger, Zachary Wasserman, Kevin Xu, and Matt Blaze. Why (special agent) johnny (still) can't encrypt: A security analysis of the apco project 25 two-way radio system. In *Proceedings of the 20th USENIX Security Symposium (USENIX Security 2011)*, August 2011.

[15] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *Proceedings of the 8th IMA International Conference on Cryptography and Coding*, December 2001.

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

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

[18] Yvo G. Desmedt and Yair Frankel. Threshold cryptosystems. In *Proceedings on Advances in Cryptology (CRYPTO 1989)*, August 1989.

[19] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[20] John R. Douceur. The sybil attack. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, March 2002.

[21] Donald Eastlake and Paul Jones. *US Secure Hashing Algorithm 1 (SHA1).* IETF Network Working Group, RFC 3174, September 2001.

[22] Saurabh Ganeriwal, Srdjan Capkun, Chih-Chieh Han, and Mani B. Srivastava. Secure time synchronization service for sensor networks. In *Proceedings of the 4th ACM Workshop on Wireless Security (WiSe 2005)*, March 2005.

[23] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation 2003*, June 2003.

[24] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2002)*, December 2002.

[25] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for ffine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS 2006)*, October 2006.

[26] Urs Hengartner and Peter Steenkiste. Exploiting hierarchical identity-based encryption for access control to pervasive computing information. In *Proceedings of the 1st IEEE International Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm 2005)*, September 2005.

[27] Florian Hess. Efficient identity based signature schemes based on pairings. In *Proceedings of the 9th Annual International Workshop on Selected Areas in Cryptography (SAC 2002)*, August 2002.

[28] Katrin Hoeper and Guang Gong. Bootstrapping security in mobile ad hoc networks using identity-based schemes with key revocation. Tech Report CACR 2006-04, Centre for Applied Cryptographic Research, Waterloo, Canada, 2006.

[29] Jean-Pierre Hubaux, Levente Buttyàn, and Srdan Capkun. The quest for security in mobile ad hoc networks. In *Proceedings of the Second ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2001)*, October 2001.

[30] Wayne Jansen, Peter Mell, Tom Karygiannis, and Don Marks. Applying Mobile Agents to Intrusion Detection and Response. In *NIST Interim Report (IR) 6416*, October 1999.

[31] Oleg Kachirski and Ratan Guha. Intrusion Detection Using Mobile Agents in Wireless Ad Hoc Networks. In *Proceedings of the IEEE Workshop on Knowledge Media Networking (KMN 2002)*, July 2002.

[32] Chris Karlof and David Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.

[33] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. *HMAC: Keyed-Hashing for Message Authentication.* IETF Network Working Group, RFC 2104, February 1997.

[34] Allison Lewko and Brent Waters. Decentralizing attribute-based encryption. In *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT 2011)*, May 2011.

[35] Donggang Liu, Peng Ning, and Rongfang Li. Establishing pairwise keys in distributed sensor networks. *ACM Transactions on Information and System Security (TISSEC)*, 8(1):41–77, February 2005.

[36] Ling Luo, Rei Safavi-Naini, Joonsang Baek, and Willy Susilo. Self-organised Group Key Management for Ad Hoc Networks. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security (ASIACCS 2006)*, March 2006.

[37] Sergio Marti, T. J. Giuli, Kevin Lai, and Mary Baker. Mitigating Routing Misbehavior in Mobile Ad Hoc Networks. In *Proceedings of the 6th International Conference on Mobile Computing and Networking (MobiCom 2000)*, August 2000.

[38] Panagiotis Papadimitratos and Zygmunt J. Haas. Secure Routing for Mobile Ad hoc Networks. In *Proceedings of the SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, January 2002.

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

[40] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. Spins: Security protocols for sensor networks. In *Proceedings of the 7th International Conference on Mobile Computing and Networks (MobiCom 2001)*, July 2001.

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

[42] Serdar Sancak, Erdal Cayirci, Vedat Coskun, and Albert Levi. Sensor wars: Detecting and defending against spam attacks in tactical adhoc sensor networks. In *2004 IEEE International Conference on Communications (ICC 2004)*, June 2004.

[43] Tomas Sander and Christian F. Tschudin. Towards Mobile Cryptography. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1998.

[44] Nitesh Saxena, Gene Tsudik, and Jeong Hyun Yi. Efficient node admission for short-lived mobile ad hoc networks. In *Proceedings of the 13th IEEE International Conference on Network Protocols (ICNP 2005)*, November 2005.

[45] Arvind Seshadri, Mark Luk, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Scuba: Secure code update by attestation in sensor networks. In *Proceedings of the 5th ACM Workshop on Wireless Security (WiSe 2006)*, September 2006.

[46] Arvind Seshadri, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Swatt: Software-based attestation for embedded devices. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.

[47] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of Advances in Cryptology (CRYPTO 1984)*, August 1984.

[48] Mark Shaneck, Karthik Mahadevan, Vishal Kher, and Yongdae Kim. Remote software-based attestation for wireless sensors. In *European Workshop on Security and Privacy in Ad-hoc and Sensor Networks (ESAS 2005)*, July 2005.

[49] Wook Shin, Carl A. Gunter, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka. How to bootstrap security for ad-hoc network: Revisited. In *Proceedings of IFIP International Information Security Conference (SEC 2009)*, May 2009.

[50] Andrew B. Smith. An Examination of an Intrusion Detection Architecture for Wireless Ad Hoc Networks. In *Proceedings of the 5th National Colloqium for Information System Security Education*, May 2001.

[51] Kun Sun, Peng Ning, and Cliff Wang. Secure and resilient clock synchronization in wireless sensor networks. *IEEE Journal on Selected Areas in Communications*, 24(2):395–408, February 2006.

[52] Arvinderpal S. Wander, Nils Gura, Hans Eberle, Vipul Gupta, and Sheueling Chang Shantz. Energy analysis of public-key cryptography for wireless sensor networks. In *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications (PERCOM 2005)*, March 2005.

[53] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*. Cryptology ePrint Archive, Report 2004/199, August 2004.

[54] Andre Weimerskirch and Gilles Thonet. A distributed light-weight authentication model for ad-hoc networks. In *Proceedings of the 4th International Conference on Information Security and Cryptology (ICISC 2001)*, December 2001.

[55] Bing Wu, Jie Wu, and Yuhong Dongand. An Efficient Group Key Management Scheme for Mobile Ad Hoc Networks. *International Journal of Security and Networks*, 4(1/2):125–134, 2009.

[56] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *Proceedings*

of the 29th IEEE Conference on Information Communications (InfoCom 2010), March 2010.

[57] Wensheng Zhang, Hui Song, Sencun Zhu, and Guohong Cao. 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*, May 2005.

[58] Zhibin Zhou and Dijiang Huang. Efficient and secure data storage operations for mobile cloud computing. In *IACR Cryptology ePrint Archive*, 2011.

[59] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia. Leap: Efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003)*, October 2003.

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