TRUSTED APPLICATION CENTRIC AD HOC NETWORK

BY GANG XU

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Liviu Iftode

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

August, 2008

ABSTRACT OF THE DISSERTATION

Trusted Application Centric Ad hoc Network

by Gang Xu

Dissertation Director: Liviu Iftode

The past few decades have witnessed rapid development of Mobile Ad hoc Networks (MANETs) technologies. However, in contrast to the huge potential and convenience enabled by MANETs, many people are still reluctant to allow their mobile computing devices to join MANETs and run MANET applications. One of the main reasons is that it is difficult (1) to guarantee trustworthiness of the MANET applications executed on remote nodes, i.e., *the lack of trusted application*, (2) to ensure fair and secure communication between multiple network nodes, i.e., *the lack of trusted communication*, and (3) to authenticate network nodes, i.e., *the lack of trusted identity*.

In this dissertation, we exploit low cost trusted hardware and the application centric nature of MANETs to address the problems of the lack of trusted application, trusted communication and trusted identity in MANETs. We present the design, implementation, and evaluation of four systems: (i) a service-aware trusted execution monitor (Satem) that ensures trusted code execution, (ii) a distributed method that creates a protected MANET to shield all network member nodes from being attacked, (iii) a distributed network communication policy enforcement mechanism that ensures secure and cooperative communication between network participants, and (iv) a locality driven key management architecture that ensures trusted identity authentication.

In distributed computing, users can initiate a transaction with a service running

on a remote node. Satem guarantees that the service will only execute trusted code across the whole transaction. The Satem architecture consists of an execution monitor residing in the operating system kernel on the service provider platform, a trust evaluator on the service requester platform, and a service commitment protocol. During this protocol, executed before every transaction, the service requester demands and verifies against its local policy a commitment from the service provider platform, which promises trusted code execution. Subsequently, the monitor enforces this commitment for the duration of the transaction. To initialize the trust on the monitor, we used the Trusted Platform Module specified by the Trusted Computing Group. We implemented Satem under the Linux 2.6.12 kernel and tested it for a peer-to-peer file sharing service and AODV routing service. The experimental results demonstrate that Satem does not incur significant overhead to the protected services and does not impact the unprotected services.

By leveraging Satem, we developed a distributed method that allows trusted nodes to create protected networks in MANETs. A protected network is created to run a specific application and enforce a common network access control policy associated with that application. To become a member in the protected network, a node has to demonstrate its trustworthiness by proving its ability to enforce policies. Attacks from untrusted nodes are impossible because these nodes are not allowed to establish wireless links with member nodes. Attacks from member nodes are stopped at the originators by the network access control policy. The trusted execution of all programs involved in policy enforcement is guaranteed by a Satem based kernel agent. We demonstrated the correctness of our solution through security analysis and its feasibility through a prototype implementation tested over an IEEE 802.11 ad hoc network.

The policy enforcing mechanism further extends the idea of enforcing application centric network policy for MANETs from link layer to the application layers. Under this mechanism, each application or protocol has an associated policy. Two instances of an application running on different nodes may engage in communication only if these nodes enforce the same set of policies for both the application and the underlying protocols used by the application. In this way, nodes can form trusted application

centric networks. The mechanism employs Satem to verify a node's trustworthiness of enforcing the required set of policies before allowing the node to join such a network, and protect the policies and the software enforcing these policies from being tampered with. We demonstrated the correctness of our solution through security analysis, and its low overhead through performance evaluation of two MANET applications.

We developed a locality driven key management architecture to address trusted identity problem for Mobile Ad hoc Networks. The method exploits locality of trust implied in application centric MANETs to enhance trustworthiness of certificates. This is achieved by having certificate authorities (CAs) be established only within a neighborhood in which all nodes are performing a common task or executing a common application. The CAs generate certificates using threshold cryptography and therefore, achieve high fault tolerance against network partition and malicious nodes. Different CAs maintain trust relationship, called trust chains, for cross-CA authentication. We implemented the prototypes for laptops and PDAs and simulated our solution in a variety of scenarios. The results verify the viability of our method.

The main conclusion of this dissertation is that the emerging low cost trusted hardware combined with the application centric nature of MANETs can be exploited to provide solutions to the problem of lack of trust in MANETs, which would otherwise be impossible.

Acknowledgements

In my pursuit of the PH.D, I have received lots of help from many people. Without them, the thesis would not have been possible.

It is difficult to overstate my gratitude to my Ph.D. supervisor, Dr. Liviu Iftode. His enthusiasm, inspiration and encouragement have always been my first resource to rely on when my research was at a stalemate. But the help I have obtained from him is not just this. Through the years, I missed many group discussions due to my full-time job. Liviu generously offered maximum flexibility to me. He not only tolerated my absences, but spent countless hours to help me catch up. I have been so luck to study under his guidance.

I would like to thank Dr. Naftaly Minsky. Dr. Minsky has served both my qualifying and defense committees. Although we have not got other cooperation opportunities, Dr. Minsky has treated me like one of his own students. He has been always available to explain problems regarding security policies. The discussions with him, though not many, have turned out to be very constructive and inspirational. I also thank Dr. Vinod Ganapathy and Dr. Anand Tripathi, the other two member of my defense committee, for reviewing my thesis.

I am grateful for Dr. Eric Allender. As the former graduate advisor, he gave me valuable advice that helped me determine the correct direction to my research and study at Rutgers.

Special thanks to Cristian Borcea, for the long and very productive collaboration. With his dedicated help and creative thinking, many of our dirty and rough ideas have been finally refined and published. I am gratitude to many of Discolab members: Steve Smaldone, Pravin Shankar, Nishkam Ravi, Vivek Pathak, Tzvika Chumash, Porlin Kang, Arati Baliga, and Lu Han, who have always been ready to help me with debugging

code, discussing research issues and reviewing my papers and talks.

As a full-time employee of AT&T, I also want to acknowledge my company for supporting my PH.D study. I am very indebted to my colleagues at AT&T. They have showed great understandings of my study and provided all sorts of assistance.

Lastly, I wish to thank my parents, from whom I inherited everything I need to complete the thesis. I owe everything to my family, my wife Xiaoling and my 5-year old son Bradley. They have been supporting me with their confidence and their love. To them, I dedicate this thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The Thesis

[1] The thesis of the dissertation is that trusted computing for Mobile Ad hoc Networks (MANETs) can be achieved by exploiting the *application centric nature* of MANETs and augmenting network nodes with *low cost hardware based trusted computing system*.

1.2 Development of Mobile Ad hoc Networks

A mobile ad hoc network (MANET) is a self-configuring wireless network in which the routers can move and organize themselves arbitrarily [11]. Although ad hoc networking was first defined by IEEE in 802.11 protocol set, the concept can be traced back to the Packet Radio Network (PRNet) projects in 1972 [65]. Because a MANET does not rely on the infrastructure and central management like the traditional Internet-like networks, it is deemed as a promising solution to support highly decentralized or mobile applications.

The earlier research efforts on MANETs were mainly concentrated on the wireless technologies and ad hoc routing. The wireless technologies, such as multihop cellular systems [121], HiperLAN [18], Bluetooth [58] and IEEE 802.11 (WiFi) [22], enable connectivity at physical and link layers. The ad hoc routing mechanisms accomplish network layer multi-hop connectivity. A large family of protocols have been developed, such as AODV [39], DSR [66], DSDV [90], and Directed Diffusion [63].

Despite the tremendous achievements in wireless and ad hoc routing technologies, the long waited MANET era has not arrived because of the far lagged development

[1] Last Revision 07/29/2008

of applications that can exploit MANETs. MANETs were traditionally envisioned to only support special applications in military battlefield [96, 76] or disaster scene [27]. This situation was improved to some extent with appearance of low cost sensors and sensor networks, which enabled civilian applications. However, the sparse resource of the sensors, in particular, low performance processors and the lack of lasting power supply, imposes a stringent restriction on development of new applications. As a result, sensor based MANETs are only suitable for a very narrow spectrum of simple and dedicated applications, such as object monitoring [51, 43, 114] and tracking [109, 104, 108]. The incapability of fostering and supporting novel and significant applications made researchers doubt the use of MANET and divert attention away from it.

Recent years, with the explosive spread of mobile computing devices ranging from laptops, smart phones to vehicular systems, enthusiasm for MANETs is coming back. What causes the renaissance of MANETs is the greatly enhanced end devices: most of them are equipped with powerful processors and wireless communication devices. For example, the first generation Apple iPhone is equipped with a 700Mhz ARM processor, and can connect to both WiFi and cellular data networks. Therefore, they can run function-rich applications rather than just simple data sensing and collection. To harness the computing power embedded in these devices, the focus of MANET research has shifted to programming models and system architectures suitable for MANETs. Numerous programming models, such as [36, 97, 95], were presented, which laid a foundation for the emergence of new generation MANET applications, such as vehicular ad hoc networks (VANETs) and mobile social networks (MSN).

A VANET is formed by hundreds or even thousands VANET-enabled cars on the road to provide safety and comfort for passengers. Prototypes from both industry and academia have been developed, such as FleetNet [53], CarNet [84] and TrafficView [46]. Mobile social networking allows the users to form a virtual community and to connect with one another using their mobile devices. It resembles the Internet social networking, such as MySpace [13] and Facebook [8], but is more than its mobile version. In addition to the traditional service provider mediated communication, the users may communicate directly with one another through so-called Pocket Switch Networks (PSN) [41, 62] and

run a variety of applications, such as chatting [83], imaging [57] and file sharing [47], without any infrastructure.

The development of the new MANET applications, such as VANET and MSN, indicates the transition of MANETs from a closed environment, where all mobile nodes are controlled or owned by a central authority, to an open environment, where mobile nodes are owned and controlled by anonymous and unrelated principals. This transition makes the security of MANET an issue more crucial than ever. A key problem is how to ensure trust to the users of MANET applications in the new open environment. The fear of becoming victims and getting their own computers compromised by untrusted network peers has driven people away from MANET applications despite their unprecedented convenience and functions. As a consequence, how to establish trust in MANETs will likely determine whether MANET applications will be well accepted in the future.

1.3 Trusted Computing: The State of The Art

Trust is defined as "assured reliance on the character, ability, strength, or truth of someone or something" [5]. In a broad sense, trusted computing addresses a large spectrum of security problems including encryption, data and code integrity, authentication, software copy protection and digital rights etc. In this dissertation, we discuss trust in distributed systems from three perspectives:

1. *Trusted application*: how to ensure an application user on one network node that the application running on another node can be trusted?

2. *Trusted communication*: how to ensure fair and secure communication between multiple network nodes?

3. *Trusted identity*: how to authenticate a network node?

The trusted application, communication and identity constitute the three pillars of trust as shown in Figure 1.1. First, trusted identity ensures that the node the user

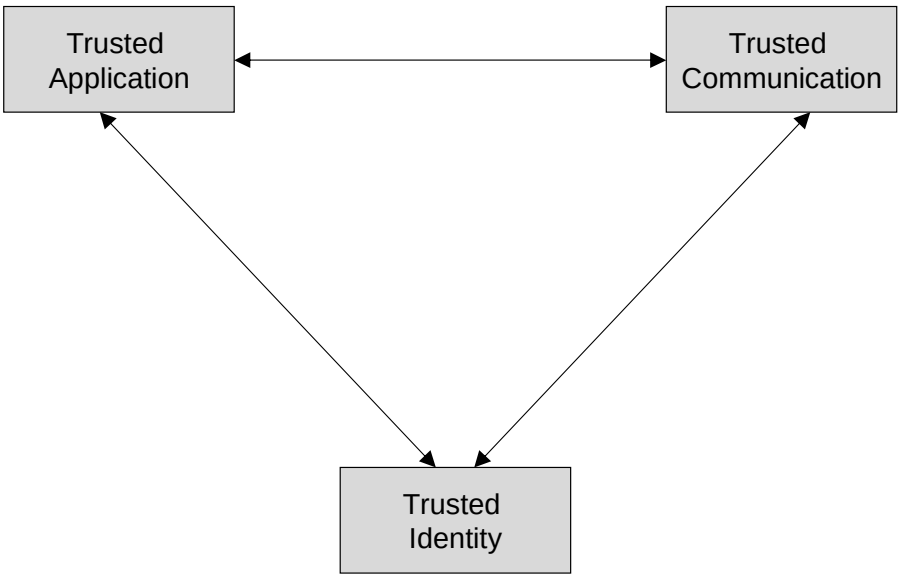

Figure 1.1: The Three Pillars of Trust

intends to communicate with is owned by some trusted principal. Second, trusted application ensures that the code base of application the user likes to call is not tampered with. Finally, trusted communication ensures that all nodes in the network will communicate with each other in a secure and cooperative manner. All the three issues have been studied in traditional Internet based distributed systems. In this section, we provide a brief survey of them.

1.3.1 Ensuring Trusted Application

To ensure trusted application, we need a mechanism to guarantee that the application will only execute trusted code base. The existing methods can be classified into three categories: (1) hardware based, (2) pure software based, and (3) a combination of low-end trusted hardware and software.

In the hardware based solutions, the target application is run in trusted processors or coprocessors. Methods following this direction include trusted processor based systems, such as uABYSS [113] and Citadel [115], and trusted coprocessor based systems like Dyad [122]. uABYSS and Citadel include a powerful processor, e.g. Intel 80386 in Citadel, and special sensing circuitry to detect physical intrusion into the board. The

main goal of these systems is to provide board level tamper-proof protection. Dyad is a physically secure hardware module consisting of a processor, bootstrap ROM and non-volatile RAM. This hardware module becomes a secure coprocessor in addition to the system main processor and is controlled by the operating system. It is dedicated to execute the trusted code and preserve the confidentiality and integrity of the data.

In distributed systems, establishing trust on applications executed on remote nodes requires verifying the integrity of the application code base. A common way to achieve this is software attestation, in which a oneway hash (e.g., SHA1) is computed over the code in question and verified by the code user [55]. An important application of software attestation is the AEGIS system [25], which secures PC booting by having each component in the boot process, such as the BIOS and OS loader, attest the next component before transferring control to it until OS kernel is loaded.

Cerium [42] and XOM [72] combine trusted processors with software attestation. Both Cerium and XOM use a tamper-resistant CPU to attest software execution stacks and protect trusted software with strong process partition. Both Cerium and XOM do not trust OS or main memory and protect code from being tampered with in the main memory.

The hardware based methods provide strong protection to the code. However, due to the high cost of the trusted processors, they are unlikely to be installed on the relative low-end portable computing devices. Therefore, they are only suitable for enterprise computing but not for MANETs.

Opposite to this trusted processor approaches is the software approach, such as [68], SWATT [102] and Pioneer [101]. In these methods, the target system is challenged to compute a checksum of its system image using a user-defined procedure. The correctness of these methods depends on two assumptions. First, the users must have sufficient knowledge about target system's hardware, such as its clock speed etc. Second, forging the same checksum of the trusted system by the actually compromised system causes noticeable delay to users. Therefore, they are more suitable for users who are familiar with and also have direct connection to the target systems. For general MANET applications, the users usually know little about the hardware of the other nodes that

run the application of interest. Moreover, given the dynamic nature of MANETs, the response time becomes completely unreliable and invalid for judging the attestation accuracy.

An emerging trend is to balance between the hardware based and software based approaches using low-end secure coprocessors. This is driven by the appearance of the Trusted Platform Module (TPM) specified by Trusted Computing Group [106]. TPM is a coprocessor embedded on the main board of the computers, which provides the capability of securely generating and storing keys and verifying the integrity of the software environment. Due to its low cost and broad support by computer makers, the TCG TPM has been already integrated in many laptops. In the near future, it will also be installed on smaller mobile devices, such as PDAs and mobile phones [107], which makes the TPM-based approach usable for many types of ad hoc networks.

Methods exploring TPM include Terra [54], Microsoft NGSCB [77], and IBM TCGLinux [100]. These methods use TPM to bootstrap trust on a set of software components, which further ensure trustworthiness of the execution of target programs.

Both NGSCB and Terra explore virtual machine monitor (VMM) [56, 111, 54, 50] to partition a tamper-resistant hardware platform into multiple isolated virtual machines. In NGSCB, a system is partitioned into two parts: trusted and untrusted, and only the trusted part is attested. Therefore, to ensure trustworthiness of an application, the provider has to treat the application and all its dependencies as trusted, which may not be true all the time.

Terra aims at allowing closed-box and open systems to co-exist. It also partitions the system into virtual machines, each of which may be dedicated to a single application, e.g., a service. As such, the trustworthiness of a service can be evaluated by attesting its VM. Moreover, it achieves higher assurance of the attestation because of the strong process isolation. A weakness of Terra is that it only guarantees trust at the time of attestation but does not prevent the service from being tampered with afterwards. In addition, it measures the VM at the partition block level, which is not verifiable to the user.

TCGLinux is the first secure integrity measurement system that explores TCG

TPM. It is useful for the users of a remote system to obtain an overall assessment of its trustworthiness. TCGLinux provides an overall assessment of the entire remote system by attesting a broad range of files. This makes it difficult for a user who is only interested in calling a specific application in the system, e.g., the game application she wants to play, to verify the trustworthiness of that application. Moreover, TCGLinux provides no assurance about trust after it is granted.

Two common problems of all above methods are the difficulty of verifying the attestation results and the lack of persistent trustworthiness. The former is due to their coarse-grained attestation methods in which either all files or the entire memory image are attested. The latter is because of the gap between time-of-test and time-of-use. Bind [103] addresses both problems. It achieves fine-grained attestation by tying the attestation of the code with the data it produces. Moreover, it ensures that the code being attested is the code being executed by protecting the code in a sandbox. BIND guarantees that the result is really produced by the application. However, this only serves as a posteriori proof. Besides, it assumes that the code being attested has to be a contiguous memory region. But ensuring trusted execution for a complex service, which typically spreads over a large number of discrete memory regions, will incur high performance overhead because each region must authenticate its precedent one.

1.3.2 Ensuring Trusted Communication

In a narrow sense, trusted communication demands protecting network nodes from being attacked by others. The de facto solution to this issue is to enforce access control policies on the "choke points", where all traffic must flow through. Enforcement of access control policies can be implemented by use of reference monitors. In the Internet based networks, the reference monitors are managed by a trusted entity in a centralized way, such as in [67, 116], which is suitable for enterprise computing rather than ad hoc environments. Recent research efforts have been seen to distribute the monitors [64, 94]. However, these methods are in essence still server centric and rely on trusted servers to host the monitors.

Sailer et al. proposed a client side firewall in [99], which enables network access

policies to be enforced on each VPN client. The method targets clients owned by legitimate users but improperly configured. As an attestation-only approach, it is insufficient to ensure trusted enforcement of the policy in face of malicious host owners.

In a broad sense, trusted communication ensures secure and proper collaboration between network nodes through the application. Research in this area focuses on either improving the expressiveness and management of the security policies [35, 34] or ensuring correct enforcement of specific policies for security mechanisms [48]. [75] implemented a general purpose policy enforcement framework. But it is mainly concerned with providing API's to integrate various policy enforcing software components.

Minsky et al. developed a model of Interaction Control(IC) [79] for the regulation of heterogeneous distributed systems. The model is based on Law-Governed Interaction [81, 80], which governs the communication between a group of nodes by a unified group policy. IC and LGI relay on a distributed TCB (DTCB) consisting of a set of *controllers* each of which is a trusted application, in our sense. The current implementation of this DTCB of LGI is via trusted software, meant to be deployed within corporate intranets [85, 94], to serve enterprise systems, or over the Internet, to support peer-to-peer communities [78]. But by itself, this implementation is not safe enough to be deployed in MANETs.

1.3.3 Ensuring Trusted Identity

In practice, trusted identity problem is reduced to key authentication. [124] is one of the first efforts addressing the key management issue in MANET. The authors proposed a conceptual model of distributed public key infrastructure, where a group of servers collectively act as a certificate authority. To achieve this, the service private(signing) key is broken into pieces, each of which is kept by one server. Threshold cryptography [105] is used to ensure that the certificate service will not be subverted unless over a quorum of servers are not available or incorrect. The solution improves both availability and security of the certificate authority, in that the system can continue to function as long as at least a threshold number of servers are still available and functional. On the other hand, when the network consists of a large number of nodes,

it may incur significant communication and computation overheads since every request needs to be distributed to and handled by all participating servers. The idea was followed up and implemented in COCA [125], whereas the implementation was targeted for infrastructure-based networks, such as the Internet.

MOCA [123] follows the same direction by building a distributed certificate service with the help of threshold cryptography. It improves security by discriminatively picking more secure nodes as CA candidates. MOCA also reduces communication overhead by caching routes to the CA servers and by using unicast instead of flooding when sufficient cached routes exist. As with COCA, MOCA inherits high communication costs from threshold cryptography. Caching alleviates the problem to some extent when the network stays static such that the cached routes are valid for a relatively long period. However, in more volatile MANET topologies, which are changing rapidly, this optimization will be insufficient.

[70] takes a step further by letting every node hold a share of the certificate authority secret key. Hence, any quorum number(K) of nodes are able to recover the key. The security depends on the system-wide parameter K. Since each node is a certificate server and compromising any K of them will disclose the private signing key, it actually endangers security to have a small K relative to the total number of nodes. However, if K is too big, it degrades to the basic form of threshold CA as [124].

Pathak et al proposed in [89] a voting based scheme for both public key authentication and group membership control. In this method, the decision of trust is made collectively by a group of n principals via voting. The system achieves high fault tolerance when it satisfies Byzantine condition. Compared to the above threshold based CA solutions, the method does not require a shared trusted principal (the dealer) and therefore, does not have any single point of failure. However, the group does not own a single signing key. Consequently each individual principal has to know the public keys of all the n voters and perform n signature verifications to authenticate one public key.

[40] addresses the key management in MANET differently. It extends the "web of trust" concept of PGP [126]. The basic idea is to let each node be its own certificate authority. As a self-organized key management system, this approach eliminates the

need to have central certificate directories in PGP for certificate distribution. Instead, each node picks and maintains a set of certificates according to a special selection algorithm. To authenticate a public key, the node u merges its own directory with the directory of the key owner v and tries to find a path from u to v. However, because the selection algorithm does not guarantee to find a path between two nodes, this method can only provide a probability result.

[29] proposed a group based method, where nodes are organized into groups identified by the group public key. Each group has a leader who owns the group private key and is responsible for certifying memberships by creating certificates to member nodes. At group level, key authentication is in a way similar to other PGP-like protocols but not fault-tolerant, in that a single compromised team leader can subvert the membership authentication. Since the method is targeted for scalable authorization, it does not support individual node authentication.

In general, authentication implies non-repudiation, which makes it desirable to use asymmetric keys, such as public and private keys. However, public key computation demands high performance processors and high power consumption. TESLA [92] addresses this problem by exploiting symmetric MAC functions to accomplish asymmetric properties. Due to its low cost, it has been used to secure group broadcasting [93] and routing [59] in sensor networks. However, TESLA requires all nodes to be loosely synchronized, which may not be practical for general MANETs.

[28] and [31] solve the problem of key authentication in certain special cases. In [28], a strong key can be derived from some prior context, a shared weak secret among all nodes. In [31], a secure side channel is assumed to exist to help initial key exchange between two nodes. However, it is questionable whether the prior context or the side channel is available for general MANETs.

1.4 Lack of Trust in MANETs

As discussed in the previous section, trust establishment has been well understood in the traditional Internet based distributed systems and mature solutions have been

Term of Comparison	Infrastructure based Networks	MANETs
Rely on Infrastructure	Yes	No
Availability of Central Authority	Yes	No
Topology	Static	Dynamic
Physical Protection of Computers	Yes	No
Application Specific	No	Yes

Table 1.1: Infrastructure based Networks vs MANETs

developed and applied in real-life practice. However, these existing solutions offer little help to MANETs due to the unique properties of MANETs. Table 1.1 compares MANETs with traditional infrastructure based networks (e.g., the Internet).

The key difference between MANETs and infrastructure based networks is that MANETs do not have a fixed infrastructure and may not have constant access to the Internet central authorities. As a result, the existing security services embedded in the Internet may not be available to MANET applications. To make the problem more challenging, the security services must be able to cope with dynamic network topology. Furthermore, different from the servers that are well protected in corporate data centers, the portable devices in MANETs have much weaker physical protection and are subject to being taken over by dedicated attackers.

Another important distinction between MANETs and the Internet is that a MANET is usually created to perform a specific task or run a specific application. In contrast to the above characteristics, which exacerbate the problem of lack of trust, we believe that the application centric nature of MANETs mitigates the problem. This is because trust does not have to be general to all potential applications running on top of the network. Instead, it is specific to the application that drives the creation of the network.

In the rest of the section, we will discuss in details the problem of lack of trust in MANETs from the three aspects of trust: trusted application, trusted communication and trusted identity.

1.4.1 Trusted Application in MANETs

For a client in client-server applications running in infrastructure based networks, the trustworthiness of the server application itself is usually of little concern. More attention is paid to ensure the trustworthiness of the server's owner. The rationale is that the servers are usually owned by well-reputed businesses and companies and are well protected in commercial data centers. As long as there is a way to authenticate the ownership of the server, the trustworthiness of the server and the applications running on it can be assured.

The above reasoning is not always valid in MANETs. A MANET is often constructed by anonymous nodes. For instance, cars on the highway can use their embedded computing and car-to-car communicating devices to form a MANET like the FleetNet [53] and run TrafficView application [46, 86]. In such cases, knowing that the application running on a specific node does not directly render trust on the application. Even if there are some "trusted" nodes, due to the lack of physical protection, the chance of having these trusted nodes captured and compromised by malicious attackers is much higher and can not be neglected. Therefore, to ensure trusted application, the user must be guaranteed that the application will only execute trusted code. However, as discussed earlier, the existing methods developed for Internet computing are not suitable for MANET applications because they incur high deployment cost, e.g., hardware based methods, are based on impractical assumptions, e.g., software based methods, or fail to ensure persistent and fine-grained trust, e.g., attestation based methods.

On the other hand, the problem may also be mitigated by the application centric nature of MANETs. The network only exists for the short time when a certain application is run. Hence, the period when trust on the application must be guaranteed is limited and typically, much shorter than in the Internet, where servers usually need to run permanently.

1.4.2 Trusted Communication in MANETs

For a group of nodes forming a MANET to run an application, protecting these nodes from unauthorized access or even network attacks is crucial. This requirement can be trivially accomplished in an infrastructure based network by enforcing network access control policies on the the "choke points", such as routers, proxies and firewalls, where all data traffic must flow. These choke points are deployed and maintained by the network owner and thereby, are fully trusted to enforce the policies. However, the choke points do not exist in MANETs. Due to the lack of infrastructure, a node can potentially establish a direct connection with another node by roaming into its wireless signal range, which bypasses any pre-deployed choke points. Even if such choke points can be created, as discussed earlier, they can not be fully trusted. As a result, it is difficult to protect MANETs from being reached by unauthorized traffic.

A more general issue than protecting nodes from being attacked by their network peers is how to assure secure communication and proper collaboration among all participant entities. This is important especially for spontaneously formed MANETs, since the services offered by these networks are collectively provisioned by all nodes. To address this problem, communication policies that govern the interactions between entities must be defined and enforced. Similar to enforcing network access control policy, this is difficult in MANETs due to the lack of trusted choke points. Moreover, in client server applications, the goal of application communication policy is usually server-centric, meaning that it aims at protecting the server from unauthorized access from the client. However, since MANET applications are more likely to be peer-to-peer, every node can be a server and a client at the same time, and no entity can be trusted more than another. Therefore, the communication policy must be application centric and all nodes must be subject to the communication policy to the same extent.

1.4.3 Trusted Identity in MANETs

Generally, authenticating a node can be further reduced to the problem of authenticating the node's public key. In the infrastructure based networks, the public key can be

sealed in a digital certificate signed by a third-party, centrally trusted, certificate authority (CA). By verifying the certificate with the CA, the public key is authenticated. In practice, CA's and certificates are organized in layers and managed by public key infrastructure (PKI) [33].

It is challenging to implement the PKI-like method in MANETs. The main obstacle is the availability of the CA. In addition to the lack of trusted central authority, the dynamic nature of MANETs makes it difficult to guarantee ubiquitous accessibility of the CA. For instance, the CA node can run out of power or temporarily roam out of the wireless signal range of other nodes, which may cause disruption of the key authentication service.

Another challenge is to ensure trustworthiness of certificates. Different than the Internet, the CAs in MANETs lack the methods of conducting investigation on every certificate requester, many of which involve manual or social background checks. The MANET CAs may have to evaluate the trustworthiness of the certificate requester by interacting with it. Therefore, the CAs should have close interaction with the certificate requesters. Moreover, different MANETs may need to talk to each other to get help for another task. For instance, in a highway, a few cars going to a common destination may establish a MANET to share directions. They may talk to other cars (in other MANETs) to get information about traffic situation [45]. This demands a mechanism for the CA in one MANET to authenticate a certificate issued by a CA in another MANET.

The conclusion of the above arguments is that the unique characteristics exhibited by MANETs have made the traditional methods to establish trust in distributed systems unsuitable for MANETs. With the emergence of more MANET applications, there will be increasing demand for mechanisms to ensure trusted application, communication and identity in MANETs. In details, we need (i) a method that builds trust on applications running on remote nodes purely based on the applications code base rather than their owners; (ii) a method that ensures nodes to communicate with each other in a secure and fair manner without relying on any pre-existing trusted central authorities; and (iii) a method that provides highly available authenticity service to the network even if

some of the service providing nodes lose connectivity or are compromised.

1.5 Dissertation Contributions

This dissertation has four main contributions to ensure trust in MANETs by employing low cost trusted hardware and the application centric nature of MANETs. In details, we developed

- a service-aware trusted execution monitor (Satem) to ensure trusted code execution [117];
- a distributed method to create a protected MANET that shields all network member nodes from being attacked [118];
- a distributed network communication policy enforcement mechanism to ensure secure and cooperative communication between network participants [119];
- a locality driven key management architecture to ensure trusted identity authentication [120];

The results of the above work were published in the Proceedings of the 1st IEEE International Conference on Mobile Ad hoc and Sensor Systems (MASS04) [120], the Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems (SRDS06) [117], the Proceedings of the 4th IEEE International Conference on Mobile Ad hoc and Sensor Systems(MASS07) [118], and Rutgers University Computer Science Technical Report DCS-tr-635, 2008 [119] (also submitted to a journal).

Service-aware Trusted Execution Monitor (Satem) [117] is a trusted computing system that achieves trusted code execution of services. The Satem architecture consists of a *service commitment protocol*, a *trusted execution monitor* in the operating system kernel of the service provider platform, and a *trust evaluator* on the service requester platform. For peer-to-peer systems, each network node is both the service provider and requester and thereby, has all components. The service commitment protocol is the key to providing a priori guarantees on trusted service code execution across transactions. During this protocol, before starting a transaction, the trusted execution

monitor on the service provider sends a *commitment* to the service requester, which describes all the code files the service may execute in all circumstances, such as executables, libraries, etc. We define a procedure for the service provider to generate the commitment through cooperation with the service software vendors and a third-party trusted authority. The service requester uses the trust evaluator to verify the commitment against its local policy and then starts the transaction. On the service provider side, the monitor enforces the commitment after the service was started to ensure that the trusted code execution promised by the commitment will not be compromised (i.e., it forbids the service to load any code files that are either undefined in the commitment or tampered with).

To initialize trust on the execution monitor, the operating system kernel (including the trusted execution monitor) of the service provider is attested through a trusted boot process using the Trusted Platform Module (TPM) [106] specified by the Trusted Computing Group (TCG). The attestation results along with the commitment are presented to and verified by the service requester during the service commitment protocol. Therefore, the successful verification of the OS kernel, the monitor, and the commitment convinces the service requester that (1) the service has executed only trusted code up to the time of commitment; and (2) the service will continue to do so during the transaction due to the enforcement of the service commitment.

By leveraging Satem, we developed a distributed mechanism to build **protected ad hoc networks** [118] to shield network members from being attacked. In this method, a common network access control policy specific to the application that triggered the formation of the network is defined and agreed upon by all participants who want to benefit from the application and foil attacks. All member nodes have to enforce the policy. A node's trustworthiness in enforcing the policy is verified before it can establish link layer connectivity with the protected network. Attacks at the network or above layers from untrusted external nodes are impossible because these nodes cannot establish wireless links with any member nodes. Attacks from trusted member nodes are suppressed at the originators by the common network policy.

The trusted enforcement of the network access control policy is guaranteed by Satem [117], which ensures that a node can communicate with other nodes in the network only if the execution of all programs involved in policy enforcement is not tampered with; otherwise,the agent tears down the links of its node.

The **policy enforcing mechanism for trusted ad hoc networks** [119] further extends the idea of enforcing application centric network policy for MANETs from link layer to application layers. Under this mechanism, each application or protocol has its own policy. All nodes supporting a certain application and enforcing its policy form a trusted application centric network. Since an application may depend on other applications, our policy enforcing framework creates a trusted multi-tier network. The member nodes in such a network must enforce the policies associated with these applications as well. For instance, a peer-to-peer file sharing application may depend on an on-demand routing protocol. In this case, the mechanism creates a two-tier trusted file sharing network. It first establishes a trusted routing tier, and hence a trusted network for routing, comprising of all nodes that enforce the routing policy. On top of this tier, it then creates a file sharing tier, enforcing the file sharing policy. Two nodes may communicate through an application if and only if they enforce the same application tier policy and all the underlying tier policies.

To address trusted identity problem for Mobile Ad hoc Networks, we developed a **locality driven key management architecture** [120]. The method exploits locality of trust in MANETs implied in the application centric MANETs. In this method, a certificate authority is established only within a neighborhood in which all nodes are performing a common task or executing a common application. The certificate authority is composed of a group of nodes, which collectively perform key authentication using threshold cryptography. Different certificate authorities maintain trust relationships, called trust chains for cross-CA authentication.

The solution is suitable for MANETs for several reasons. First, locality makes certificates more trustworthy in that in a local community a CA has better chance to interact with other principals. Moreover, it reduces communication overhead between principals with their CA because of shorter local distance of message delivery. Thirdly,

threshold cryptography greatly improves fault tolerance and provides high availability.

1.6 Contributors to Dissertation

In this dissertation, I used material from three papers that Cristian Borcea (Assistant Professor of New Jersey Institute of Technology) co-authored. Cristian Borcea contributed to the design of the protected ad hoc network, the policy enforcement mechanism and the case study that was used to illustrate the application of the protected ad hoc networks. Josiane Nzouonta (PH.D student of New Jersey Institute of Technology) implemented the NS2 [20] tool to generate the vehicular scenarios used for evaluation of the policy enforcement mechanism. Jiejun Kong (PH.D student of University of California, Los Angeles) provided source code of threshold cryptography algorithm, based on which the locality driven key management architecture was implemented.

1.7 Dissertation Roadmap

The dissertation is organized as follows. Chapter 2 describes Satem, a TPM based trusted computing system. In Chapter 3 and 4, we present the distributed mechanism to build protected ad hoc networks and the policy enforcing mechanism. The locality driven key management architecture is discussed in Chapter 5. Finally, we conclude the dissertation in Chapter 6.

Chapter 2

Satem: Service-aware Trusted Execution Monitor

This chapter presents the design of Satem, a trusted service-aware execution monitor that guarantees the trustworthiness of the MANET service or application code execution across the entire transaction. We start with the problem statement of ensuring trusted application code execution in MANETs in Section 2.1. Then, the Satem architecture is presented in Section 2.2 followed by the commitment protocol that facilitates establishment of trust on the application in Section 2.3. Next, we discuss the prototype implementation of Satem in Section 2.4 and the performance evaluation in Section 2.5. We demonstrate how Satem solves the security threats in Section 2.6. Finally, we discuss its limitations in Section 2.7 and summarize the chapter in Section 2.8.

2.1 The Problem Statement

In distributed computing, an application A on a machine X calls an application B on another machine Y to conduct a series of transactions. The applications A and B can be different, e.g. A being the client and B being the server program, or the same, e.g. both being a peer-to-peer application. In either case, B provides a computing service to A. Hence, we refer to machine Y on which B is run as the service provider and machine X on which A is run as the service requester. [1]

Because the service is remote (i.e., the application is computed on the remote service provider), the service requester is concerned about whether it is truly the claimed service. Traditionally, this problem is solved by authenticating the service provider. For example, in the Internet, a user of a web application can authenticate the web server

[1] In the reminder of the chapter, we use application and service interchangeably.

through its digital certificate. However, with frequent incidents of security breaches, knowing what machine runs the application is no longer sufficient to convince the user that she is interacting with the right application, because the application software may be tampered with. This problem is more severe in MANETs. For one thing, a MANET is often constructed by anonymous network nodes, which makes identity based authentication helpless. For another thing, even if some of the nodes are "trusted", due to the lack of physical protection, the chance of having these trusted nodes captured and compromised by malicious attackers is much higher and can not be neglected.

Recently, several methods [77, 106, 42, 6, 72, 100] based on software attestation [55, 25] have been proposed to ensure code genuineness and integrity on untrusted hosts. However, they are insufficient to achieve trusted application code execution for the entire lifetime of an application transaction for two reasons. First, although these methods can ensure the trustworthiness of code at the time of attestation, they either do not guarantee that the trustworthiness will persist afterwards(i.e., during the transaction) [100, 54], or fail to provide such a guarantee prior to invoking the application [103]. Second, the existing methods lack the capability of precisely measuring and protecting the integrity of the application code base, which consists of all the programs loaded by the application at runtime. This makes it difficult for trust evaluation since a change detected by the attestation may be irrelevant of the application code base and thereby has no impact on its trustworthiness.

To motivate the need for a new approach, we present three security threats faced by users of MANET applications, which are difficult or even impossible to address by the existing attestation based methods. To illustrate the security threats, we consider a vehicular ad hoc network application run by a number of cars with mobile computing devices on a highway, which queries and disseminates traffic and safety information. To ensure correctness of the information, such an application usually requires binding messages a car sends to its unique identity [71, 88]. This raises the concern of privacy [61, 88] since the combination of the positions and the identity of the car may turn the application into a tracking system. Therefore, it is desired that the application running on each car should only use the identity and position information in a legitimate

way and not disclose them to other unauthorized entities. Since the computer is owned by each user, we assume that the attacker can have physical access to and superuser privilege on at least one computer in the network (e.g. her own computer) and is free to install untrusted code on it. Her goal is to trick other cars into communicating with the untrusted application on her car in order to acquire their positions and identities.

Attack 1: Service Spoofing

The attacker does the following:

1. Runs her own evil application, e.g., `/tmp/evanet`, on a UDP port 222.

2. Runs the legitimate application, e.g., `/bin/vanet`.

After step 1, if `/tmp/evanet` is not in the scope of attestation, no record is taken. After step 2, the `/bin/vanet` is attested if it is in the scope of attestation. At this moment, however, the evil application runs on port 222 rather than the legitimate `vanet` service because `vanet` failed to bind to the port.

The attestation cannot reveal the problem if it does not cover `/tmp/evanet`. To solve the problem by including `/tmp/evanet` in the attestation scope implies including all files in the system, because there is no way to predict which file will be used to attack the service. Attesting the entire file system is impractical. Not only does it increase the cost, but also makes it difficult for the requester to assess the attestation result. For example, knowing that both `/tmp/evanet` and `/bin/vanet` were executed does not help the requester to understand which is the application she intends to connect to.

An alternative to counter this attack is to also attest the runtime process status. This solution, nevertheless, further complicates the attestation because the process status constantly changes. Furthermore, the process status is not standard, which makes it difficult for the requesters to verify its integrity.

Attack 2: Service Tampering

The attacker can also modify the code base of the application by changing its binary. For

instance, modern applications usually rely on some shared libraries. Therefore, a determined hacker can force `vanet` to link to an evil shared library (e.g., `/lib/libc.so.6`) without being detected by the attestation. She does the following:

1. Runs another program `/bin/vanet2` that also uses `libc.so.6`.
2. Installs an evil shared library with the same name in `/tmp`, `/tmp/libc.so.6`.
3. Sets `LD_LIBRARY_PATH=/tmp:/lib/`.
4. Runs `/bin/vanet`.

The attack is based on the assumption that `/tmp` is not watched by the attester. Also, as explained in Attack 1, it is impractical to attest all files. After step 1, the `libc.so.6` as well as `/bin/vanet2` are attested correctly. Since `libc.so.6` is a common library, it is easy to find another program like `/bin/vanet2` that also uses it. After step 4, `vanet` is attested, but linked to the evil library, though the attestation results contain the correct `vanet`, `vanet2`, and `libc.so.6`. The attack may be detected by attesting and publishing the environment variables. However, similar to attesting the process status, this is not acceptable in general.

Attack 3: Post-Request Attack

Compared with the difficulty of assessing trustworthiness of the application as shown in attacks 1 and 2, an even more complicated problem comes from the impossibility to guarantee a priori that the application will run only trusted code after the initial trust is established. In other words, the trustworthiness of the application is valid, at best, at the time of attestation, not even the time of verification. From then on until the next attestation, the application is completely vulnerable to tampering.

Assume that after verifying the attestation result, some car decides to trust the attacker's car and send its position and identity in a message. We also assume that every time the application `vanet` receives a message, it invokes another application `/bin/aggregator` for data aggregation. The attack takes place as follows:

1. The car sends its identity and position along with the message to the application run on the attacker's car.

2. Before receiving the message, the attacker replaces `/bin/aggregator` with a malicious software,`/bin/eaggregator`, which in addition to desired data aggregation saves the car's identity and position to a plain text file.

Since the malicious `eaggregator` has not been called by the time of the pre-request attestation, it is not revealed by the attestation report. It is invoked after the identity and position is sent. Although the attestation can be done immediately upon execution, the victim will not know this until it requests the next report.

In summary, although existing attestation based approaches, such as [100], address attacks 1 and 2, they are subject to high false positive rates due to lack of service-awareness. The attack 3 cannot be handled by any existing approaches because they are unable to guarantee trust across application transactions before the transactions start.

To tackle the above problems, we proposed Satem, a service-aware trusted execution monitor, to achieve trusted service code execution across service transactions. Satem exploits the Trusted Platform Module (TPM) [106] to bootstrap trust on the execution monitor, the kernel agent on the service provider. The trusted execution monitor then provides a priori guarantee on trusted service code execution across transactions through service commitment protocol. During this protocol, executed before every transaction, the service requester requests and verifies a commitment from the service provider that promises trusted code execution. Subsequently, the execution monitor enforces this commitment for the duration of the subsequent transactions.

The primary benefit of Satem is that it provides a priori guarantee to a service requester that only trusted service code will be executed across the upcoming service transactions. Due to its service-awareness, Satem only protects code executed by the service and reduces both the attestation overhead and false positives. Another benefit that makes Satem suitable for MANET applications is its certificate-based trust

Figure 2.1: The Satem Architecture

evaluation method. In this method, the service requester evaluates the service trustworthiness through one certificate check without managing a huge database of authentic code hashes. We introduced a short-life certificate based authentication scheme, which allows nodes to authenticate certificates without constant connectivity to the Internet certificate authorities.

2.2 The Satem Architecture

In this section, we present the architecture of Satem, which comprises of components on both the service provider and the requester sides. As Figure. 2.1 shows, the service provider components include a TPM (Trusted Platform Module), a trusted execution monitor, and a commitment for each protected service. On the service requester side, Satem includes a trust evaluator and a trust policy. In our model, we assume that the attacker is unable to perform direct hardware attacks. For example, she cannot write to or read from the TPM, network card, CPU registers or physical memory without going through the OS kernel. In particular, we assume that the attacker cannot perform Direct Memory Access (DMA) based attacks. Other than that, we consider that the attackers can get super-user privileges and modify any software, including the OS kernel, at any time.

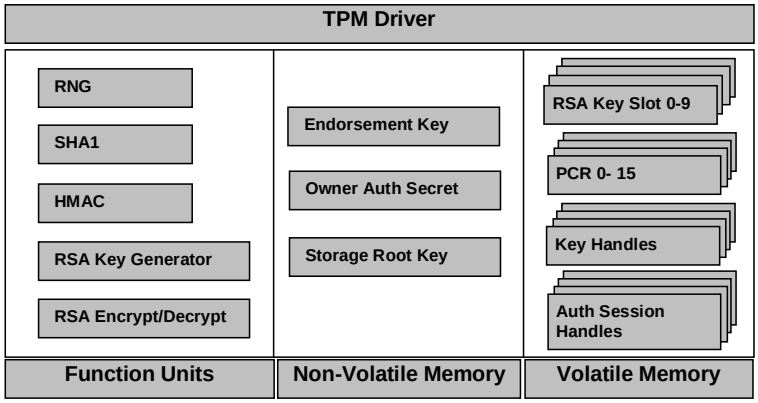

Figure 2.2: TPM Overview

2.2.1 TPM: The Root of Trust

The Trusted Platform Module (TPM) specified by Trusted Computing Group (TCG) is the root of trust of Satem. We briefly introduce the IBM implementation with a focus on the features used by Satem. More details can be found in [106, 98].

The IBM TPM integrates cryptographic function units and private volatile and non-volatile memory on a single chip, which is resistant to tamper and probe. The function units include a random number generator (RNG), a SHA-1 and an associated HMAC calculator, a key generator, which is capable of generating up to 2048 bit RSA key on the chip, and a RSA engine for hardware signing, encrypting and decrypting.

The non-volatile memory is used to store root secrets including the unique manufacturer endorsement key, an owner supplied authentication secret and a root key for security storage. The implementation of TPM guarantees that these secrets can never leave the chip. A user may use them, e.g., to decrypt a message using the endorsement key, only through a restricted APIs provided by the TPM.

The volatile memory provides a secure storage for run-time secrets and software integrity measurements. User keys can be loaded into the key slots on demand and referenced via key handles. Each key is associated with an authorization secret and can be protected by another key, which is called parent key. To load a key, a user must pass authorization, which means providing the right authorization secret of the key and

its parent keys. Successful authorization returns a session handle, saved in the Auth Session Handles table, to the user. When a key is not in use, it can be wrapped with user specified authorization secret and purged out of the chip. One of the most important functions of TPM is to capture system states and report them in a tamper-evident manner. This is achieved by using the 16 platform configuration registers (PCRs).

2.2.2 Bootstrap Trust using TPM

Satem uses trusted boot and secure attestation functions of TPM to bootstrap trust on the underlying platform and the Satem trusted execution monitor.

At the boot time, the service provider runs through a trusted boot procedure, in which each component in the boot sequence attests the next one before handing over the control. The TPM helps establish trust on the OS kernel and the Satem execution monitor, which is the root of trust in all service transactions. The attestation result is saved in a PCR register (PCR_0), which is an internal configuration register of TPM. The only way to change the content of a PCR is through the function `TPM_Extend`, which computes a `SHA1` hash over the PCR's current content and the new object to attest. This prevents the PCR content from being reset (i.e., the attacker can change the attestation value, but it cannot set it to an arbitrary predetermined one). In our case, the TPM invokes this function to attest the BIOS image and transfers control to the BIOS after that. Consequently, the BIOS calls `TPM_Extend` over the OS loader (e.g., LILO), and the latter does the same over the OS kernel image, denoted as OSK. As a result, after the OS kernel is loaded,

$$PCR_0 = SHA1(SHA1(SHA1(0|BIOS)|\\ LILO)|\\ OSK)$$

assuming $PCR_0 = 0$ initially.

The content of selected PCRs is reported via the `TPM_Quote` API, which signs the content with a TPM internal key. To counter replay attacks,this API also includes a 20 byte random *nonce* as a parameter in the signed report. We assume that the TPM is unbreakable and, therefore, we consider that the attestation it conducts and the report

it produces cannot be tampered with [2].

2.2.3 Commitments

Satem defines one system commitment for the service provider platform and one service commitment for each service that wants to use Satem (referred to as protected service). The system commitment describes all the kernel code (e.g. modules) the service provider may load before next reboot. Each protected service has an associated commitment that describes all the code the service may execute in its entire lifetime. Both system and service commitments are composed of the integrity descriptions of all the software code files in the format of a tuple <file name, SHA1 hash value>. A snippet of a possible Apache web service commitment is presented in Table 2.1.

```
software name   =  Apache
version number  =  2.0.50
file name       =  httpd
SHA1 value      =  7e7923bb0b7a0e74d2e...
file name       =  libaprutil-0.so.0
SHA1 value      =  d888e5f9916761cca24...
...
```

Table 2.1: A Satem Commitment Example

It is the service provider's responsibility to prepare the commitments of its protected services. There are two ways to determine it. One is via testing (i.e., tracing the service for every possible request to find out what code it executes). In theory, this method may be incomplete since it is hard to ensure that the test exhausts all branches of executions. A better way is to trust the service software code producers to provide this information [54]. Satem adopts and extends this approach. In Satem, a commitment of service S (denoted as $C(S)$) is a certificate signed by a certificate authority (CA) trusted by service requesters of service S. This commitment is generated as follows:

1. *Request code certificates.* The service provider requests each vendor to generate

[2]The current TPM specifications use SHA1, which has been found breakable [112]. We expect future releases of TPM to be upgraded with a stronger one-way hash function such as SHA256.

a self-signed code certificate in the same format as the commitment for its code.

2. *Sign the commitment.* The requester forwards all the code certificates and the commitment to the CA. The CA needs to verify the signatures of all code certificates and compare the code hashes in the commitment against the certificates. The CA signs the commitment if and only if it verifies all code certificates and code hashes in the commitment.

$C(S)$ only guarantees to the requester that the code described in $C(S)$ is what its vendors released. The requester has to verify against its local trust policy that the vendors and their code are trusted. To the service requester, decoupling code trustworthiness from genuineness not only simplifies trust management, but also provides the flexibility to use any trust policy.

The CA plays a central role in trust establishment. Its job is simplified, however, by decoupling code genuineness from code trustworthiness. All a Satem CA has to do is to verify authenticity of software code certificates. Unlike issuing identity certificates, no manual and time consuming background investigation is necessary. Furthermore, the need for certificate revocation is minimal because the certificate only vouches for the fact that the vendor certifies the code's unique digest. The only possible scenario for revocation is when the attacker compromises the signing key of the software vendor and generates bogus integrity description for the vendor's software. Although such certificate authority service does not exist today, any reputed security organization (e.g., CMU CERT [4] or SANS [21]) can take on this role.

2.2.4 Trusted Execution Monitor

The monitor resides in the OS kernel of the service provider and has two main goals: (1) provide a guarantee to the service requester that only trusted code will be executed by the application during the transaction, and (2) enforce fail-stop protection of the application during transactions.

The monitor is loaded and attested along with the OS kernel. Once being loaded, it immediately enforces the system commitment. It enforces the service commitment

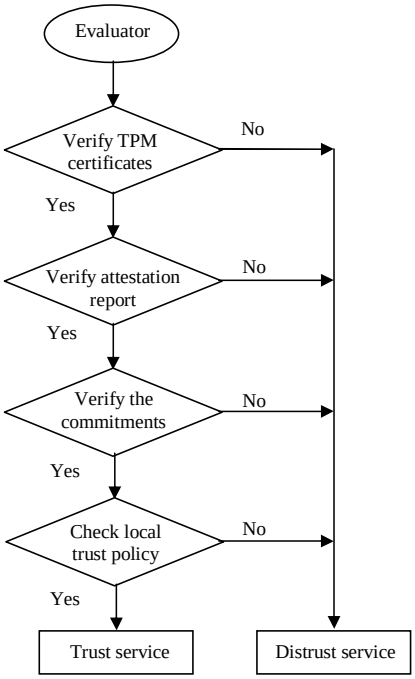

Figure 2.3: Satem Trust Evaluation

of each protected service only when the service is started. The monitor enforces these commitments in the same way. First, the protected service and the system will not be allowed to load any code that is not defined in the commitments. Second, the monitor forbids the service and the system to load any code that is defined in the commitments but tampered with. An extensive description of the methods used to enforce the commitments is presented in Section 2.4.

2.2.5 Evaluator and Trust Policy

In order to establish a trusted transaction with a service, the requester asks the service provider to deliver an attestation report of the service platform OS and the service and system commitments. The evaluator determines if the service execution described by the report and the commitments can be trusted. Figure. 2.3 illustrates the trust evaluation process. The evaluator first checks the authenticity and the integrity of the report and the commitments. For the report, this requires the authentication of the TPM public key (TPM's private key is used to sign the report). TCG defines a series

of certificates (signed by the TPM CA) for users to authenticate TPM and its public keys. After verifying the genuineness of the TPM and TPM keys, the evaluator checks the attestation report. For the commitments, the evaluator needs to authenticate the public key certificate of the CA that signs the commitment.

In general, authenticating the public key is non-trivial in ad hoc networks due to the lack of constant connection to the Internet and PKI. A node may still be able to authenticate a certificate if it locally holds the public key of the signing certificate authority or a valid certificate chain to it. However, it is unable to validate in real-time the certificate since it has no access to the CA's certificate revocation list. The issue can be significantly alleviated given the special nature of the problem we aim to solve. As discussed in [30], although nodes do not have persistent Internet connectivity, they can still get online from time to time. For example, a user may be off-line on an inter-city train most of the time, but get online when the train enters a train station. Furthermore, a Satem commitment only states that a code file has a certain corresponding `SHA1` digest. This fact is invariant under any circumstance. Lastly, since the ad hoc network is formed for a specific task and only lasts for a relatively short period of time, the likelihood of revoking a certificate is negligible.

Based on the above observations, we introduce a *short-life certificate* to authenticate the commitment. When being connected to the Internet, each node obtains a regular long-life commitment certificate C_L, a short-life commitment certificate C_S, and the authority's certificate C_A. When losing Internet connectivity, it can still use the C_A to authenticate C_S of other nodes. Since this certificate is only good for a short period of time, there is no need to be concerned about revocation. After C_S expires, the node needs to regain Internet access to renew it using its C_L. The CA verifies the C_L using PKI and grants the renewal request without re-authenticating it from scratch.

Verifying the report and the commitments proves the genuineness of the code that can be executed by the service platform OS kernel and the service. The requester has to verify the commitments against its local trust policy to make the trust decision. Although creating an appropriate trust policy is an interesting research topic, it is beyond the scope of this work. A promising approach is to have the service software

certified using methodologies proposed by Voas [110]. Here, we simply assume that the policy exists on each service requester that wants to use Satem. If the evaluator verifies the report successfully, the user will trust the execution monitor to enforce the commitment. Consequently, if the evaluator verifies the commitment, the user is convinced that the service will only execute trusted code.

2.3 The Service Commitment Protocol

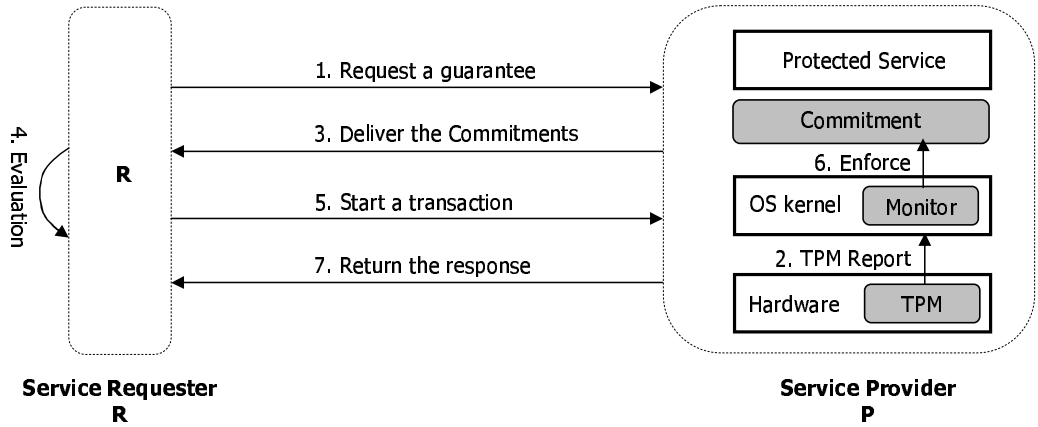

Figure 2.4: The Steps of the Satem Service Commitment Protocol

The requester R establishes trust on the service S on the service provider P through the protocol illustrated in Fig. 2.4. This protocol assumes that the attestation results obtained at the boot and OS kernel loading time have been saved in the TPM (as described in the previous section).

1. R sends a request (TST) demanding P to provide a guarantee of trusted execution of S,

   $$TST =< SID, nonce, PK_R >$$

   where SID is the service identity (in a TCP/IP network, this is the ip address and port number), *nonce* a random number, and PK_R its public key.

2. Upon receiving TST, P generates a key k, which is saved in the kernel memory controlled by the monitor. Then, it calls the TPM to generate a report *Rep* of

the content of PCR_0,

$$Rep =< PCR_0, p >$$

where PCR_0 is the content of PCR_0 of the TPM. p is a parameter fed into the TPM for report generation. It is defined as

$$p = SHA1(nonce|SHA1(C(S))|SHA1(C(T)) \\ |SHA1(PK_R)|SHA1(k))$$

where $C(S)$ is the service commitment and $C(T)$ is the system commitment.

3. P sends Rep to R for evaluation. In addition, it encrypts k with R's public key PK_R (denoted as $(k)_R$) and sends it together with the commitments $C(S)$ and $C(T)$ to R.

4. R must verify Rep, $C(T)$ and $C(S)$ against the local trust policy before starting the transaction. R first verifies the authenticity and integrity of Rep, $C(T)$ and $C(S)$. In addition to verifying the TPM signature, it decrypts $(k)_R$ with the private key SK_R, and computes p' using $C(S)$ and $C(T)$ received at step 3, its own copy of $nonce$, and PK_R. Rep is verified if and only if $p = p'$.

5. When everything has been verified, R sends S the request $Req(S)$ to start the transaction. R and S use k to encrypt the transaction.

6. The monitor enforces $C(T)$ and $C(S)$, which guarantees that S will only execute trusted code to process Req.

7. Finally, S may generate and send back a corresponding response Res.

The trust decision on the service S is made at step 4. From Rep, the user learns that the service platform has been booted into a trusted Satem kernel. Knowing $C(T)$ ensures the user that the kernel including the monitor has been protected and thereby can be trusted. Knowing $C(S)$ convinces the user that the monitor is trusted to enforce $C(S)$. The enforcement begins from the beginning of the service S since the monitor loads $C(S)$ when S starts. The p of Rep also proves to the user that k, PK_R, $C(T)$, and

$C(S)$ received with *Rep* are the same with those used by the monitor, which defeats spoofing attacks.

From step 5 on, both R and P use k to encrypt the transaction traffic. This prevents the attacker from hijacking the transaction by launching a man-in-the-middle attack. The attacker may try to steal k by intercepting TST at step 1 and replacing PK_R with its own PK_A. Then, it intercepts the returned k at step 4 (encrypted with PK_A), decrypts it and re-encrypts it with PK_R. This attack will be detected by R at step 4 due to the inclusion of $SHA1(PK_R)$ in the parameter p. On the other hand, the attacker can impersonate P. For this attack to succeed, the attacker's machine must be Satem-enabled. Otherwise, the requester will refuse to trust it at step 5. However, when Satem is enabled, it guarantees that the service will only execute trusted code no matter who owns the service provider platform.

2.4 Prototype Implementation

In order to verify the design concepts and understand the performance of Satem, we have implemented a prototype under the Linux 2.6.12 kernel. The most important part of the prototype is the trusted execution monitor. The prototype includes also TPM control functions as well as the evaluator on the client side. We have not implemented the attestation functions in BIOS and LILO, which have already been implemented in the Enforcer project [6].

2.4.1 Satem Monitor and Commitment Enforcement

The focus of the Satem monitor is to provide a fail-stop protection mechanisms to enforce service commitments. Our implementation has less than 1000 lines of C code. The complexity, however, comes from the fact that the monitor code is integrated into the OS kernel almost everywhere by inserting checkpoints to kernel calls, such as `do_execve` and `sys_open`, to intercept new code execution invoked by protected service processes. These modifications are added by patching the original Linux kernel of the service provider platform. When a service needs protection, the monitor associates a

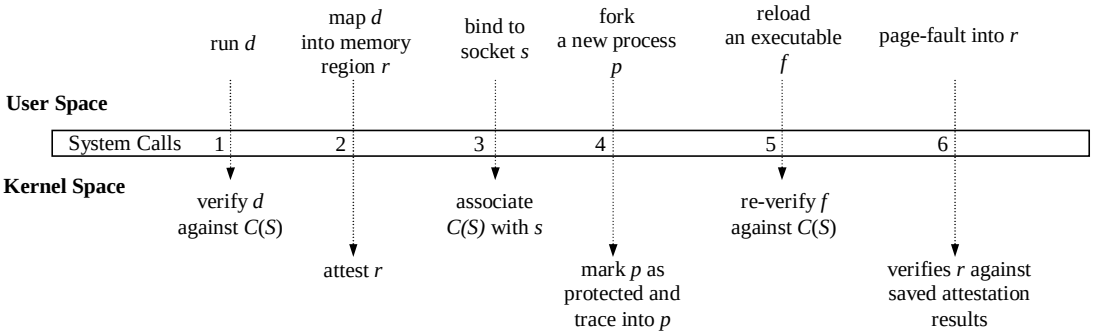

Figure 2.5: Satem Commitment Enforcement Work Flow

protection flag with all processes executed by this service, memory regions mapped by these processes, and code files opened by them. By having this flag, we achieve service-awareness by limiting the scope of Satem within the protected service instead of performing attestation and other Satem actions on irrelevant programs. More details about our protection mechanism will be presented throughout this section.

The commitment of a protected service is loaded into the kernel memory. It is implemented as a table, and Satem uses the name of the service code file as the key to look up the corresponding hash value. We use the Linux kernel `crypto` API to implement SHA1 functions. Figure. 2.5 shows, from left to right, how the monitor enforces the service commitment when a service is launched and executed. The up-down arrows represent the interception of kernel calls, in which the monitor has the checkpoints. We assume that the services are based on TCP/IP.

Loading

When a service S is started, its daemon program d is executed; this results in an invocation of `do_execve()` that traps into the kernel. The monitor intercepts the call in the kernel (arrow 1 in the figure) and does the following:

1. Compares d with a preset list of protected services, PS, to see if $d \in PS$. Service providers that want to use Satem must provide such a list. In brief, it defines the mapping between d and $C(S)$. For instance, the following PS defines two

protected services: a peer-to-peer file sharing application associated with a commitment `cf` and a routing application associated with a commitment `cr`.

`</usr/mute/bin/textMute cf>`

`</usr/bin/aodvd cr>`

The monitor does not verify whether PS tells the truth about what commitment is associated with the service. In this way, an attacker may try to associate a bogus commitment with the service in order to run untrusted code. From a requester's perspective, however, this is not a problem because the evaluator on the requester side will refuse to trust the commitment during the service commitment protocol. If the commitment is trusted, the monitor will detect immediately any attempt to execute untrusted code either by d or by the service.

2. If $d \in PS$, the monitor marks the current process as protected. Then, it reads the commitment, $C(S)$, attests d, and verifies whether d is defined and has the same `SHA1` value as in $C(S)$.

3. When an interpreter, i, is loaded (e.g., `ld.so` for binaries, `perl` for perl scripts), the monitor recognizes that it is loaded with a protected service and thereby attests i in the same way as d.

4. The kernel maps memory for d and i using `do_mmap` (arrow 2 in the figure). Since the mapping is called by a protected service, the monitor marks the memory region r as protected and partitions the entire region into a list of small segments such that $r =< sg_0, sg_1, sg_2, ...sg_{n-1} >$. Each sg_i corresponds to a page. The monitor then reads each page to fill the segments sg_i one after another and computes SHA1(sg_i). The attestation results are saved in kernel internal memory. After attesting each sg_i, the kernel does not attempt to keep the content of the page in memory.

The monitor saves the attestation results of protected memory regions, commitments, and the secret key k (defined in Section 2.3) in system memory rather than in the tamper-resistant TPM. This is caused by the fact that TPM does not have sufficient

space for the variable-sized results. Our prototype addresses this problem by letting the monitor define the area and forbidding access to it from any non-Satem user-space applications. We assume that the OS kernel correctly protects kernel memory such that the only channel to access this area from user space is through direct memory mapping mechanism, such as `/dev/kmem` and `/dev/mem`.

Linking

When d is fully loaded, the kernel transfers the control to its interpreter to load the shared libraries. Similar to d, a library l is mapped, not read, into memory and attested.

Binding

Once loaded, the service program needs to bind a network socket. The `sys_bind` system call traps into the kernel (arrow 3 in the figure). The monitor first checks whether the current process is protected. If so, it links the socket to $C(S)$; in this way, the service platform can deliver the right commitment to requesters.

Cloning

The current process may create new child processes. In this case, the monitor intercepts the fork system call (arrow 4 in the figure) and checks if the current process is protected. If so, it marks the new child process as protected and links it to $C(S)$. Then, the monitor will track the child process in the same way with its parent.

Code Changing

The user may modify the kernel by loading new kernel modules. When a kernel module is loaded, the trusted agent intercepts `sys_init_module` and verifies its integrity against the system commitment without checking the calling process's protection flag. Another way to modify the kernel is to reboot into a different one. In this case, the change is captured by the TPM in the trusted boot procedure and revealed to the requester when the attestation report is verified. We assume that the OS kernel isolates processes correctly (i.e., it forbids one user process from accessing memory of another process

without authorization). Therefore, once the code is loaded into memory, the attacker cannot modify it without compromising the kernel.

The attacker, however, can modify arbitrary code files. The monitor does not attempt to catch changes in code files. Instead, it detects and blocks any attempt of a protected user space process to invoke the compromised code. For instance, reloading a modified d (arrow 5 in the figure) will be blocked in `do_execve` call.

A less obvious attack is to directly modify mapped binary code chunks on the disk. Consequently, the tampered code chunk is loaded into memory by `filemap_nopage` when a page fault occurs. Attestation at the granularity of files cannot detect this problem. A mapping can exist even without keeping the underlying file open, which makes it impossible to catch changes just by watching system calls, such as `sys_open()`, `sys_read()` and `sys_write()` as in [100]. Satem can defeat such attacks by using the previously saved hash values for the protected memory regions. Each time a memory fault causes a real read of a missing page, the monitor recomputes the SHA1 value of the page to be read and compares it with the saved value (arrow 6 in the figure). If the values are different, the monitor concludes that the page was tampered with, on the disk, after being mapped.

Monitoring Interpreted Programs

Interpretation of scripts or virtual machine based executables (e.g., perl scripts or java servlets) are more difficult to monitor because their execution takes place in a black box from the monitor point of view. To solve this problem, we monitor all the files opened and read by protected processes (e.g., in `sys_read` call) and verify each of them against the commitment.

2.4.2 Lazy Attestation

Attestation of large code files and mapped memory regions is costly. This can cause significant overhead to the service provider system since the service may repeatedly invoke the same code. Since not all code segments in a binary will be loaded for execution, Satem uses lazy attestation to minimize the overhead by avoiding attesting

code segments which are not used. When the binary is executed for the first time (via `do_execve()`), Satem attests the file as a whole. In addition, it partitions the binary file into page-sized chunks and also attests the loaded chunks, such as those containing the ELF headers. The results are cached, and when the file is executed subsequently, Satem does not attest again the file if the commitment has not been changed. Instead, it verifies each of the loaded chunks against the cached attestation results. Satem will attest the entire file only if the execution loads a chunk which has not been loaded before. Similarly, when a code segment is mapped, Satem uses the previously saved attestation results rather than re-attesting it.

2.4.3 TPM Functions and Satem Trust Evaluator

Our implementation of TPM Functions is based on the IBM TPM driver. Satem uses TPM in a very light way. It only uses TPM for attestation and result reporting (i.e., the `TPM_Extend` and `TPM_Quote` functions). The Satem trust evaluator is a simple user space application. It performs RSA signature verification. The trust policy was set to "allow any" in the prototype. We have not implemented the commitment certificate yet, but this is trivial if the certificate authority is known to the evaluator.

2.5 Evaluation

To evaluate the performance of Satem, we measured the overhead in terms of monitoring scope, transaction processing delay, commitment delivery, as well as the overhead imposed by Satem on application execution and kernel calls.

2.5.1 Methodology

We created an 802.11g ad hoc network consisting of three laptops (IBM T43 with a 1.7Ghz Pentium M CPU, 512M RAM, and Atheros wireless card). In order to test multi-hop communication, the network was configured with a line-like logic topology (i.e., the direct link between laptop 1 and 3 was disabled, but they could still communicate with each other through laptop 2 as a router). This was achieved by enabling MAC filtering

Application	Satem Monitoring Scope (number of files)
File Sharing	47
Routing	41

Table 2.2: Scope of Satem Monitoring

Commitment Latency	1.3(seconds)

Table 2.3: Satem Commitment Delivery Latency

using `iptables` [19] on each laptop. Two applications were used for experiments: a peer-to-peer file sharing service and an ad hoc routing service. The former was implemented using Mute [12] and the later using AODV-UU [2].

2.5.2 Scope of Monitoring

Satem does not have to monitor all files on the service provider due to its service-awareness. In this way, it can significantly reduce the monitoring overhead. For our experiments, Table 2.2 shows the size of the monitoring scope in terms of number of files. For instance, to protect the file sharing application, about 50 files are monitored at runtime: 36 kernel files, 10 library files and 1 executable. Compared with IBM TCG Linux [100], for instance, which attests more than 250 files at run time, the scope of Satem attestation is significantly reduced.

2.5.3 Commitment Delivery Latency

We measured the service provider's cost for commitment delivery as the latency of getting the commitment ready for delivery. As shown in Table 2.3, it is costly to deliver the commitment. This also demonstrates the efficiency of Satem compared to other attestation-based methods because this is just one-time cost. After trust is established, Satem protects the service from being tampered with.

2.5.4 Transaction Processing Delay

To users of protected services, the performance is measured by the time to finish a service transaction. To test the file sharing service, we ran Mute on both laptops and

Application	Performance Overhead (%)
File Sharing	5.71
Routing	1.06

Table 2.4: Satem Application Transaction Processing Delay

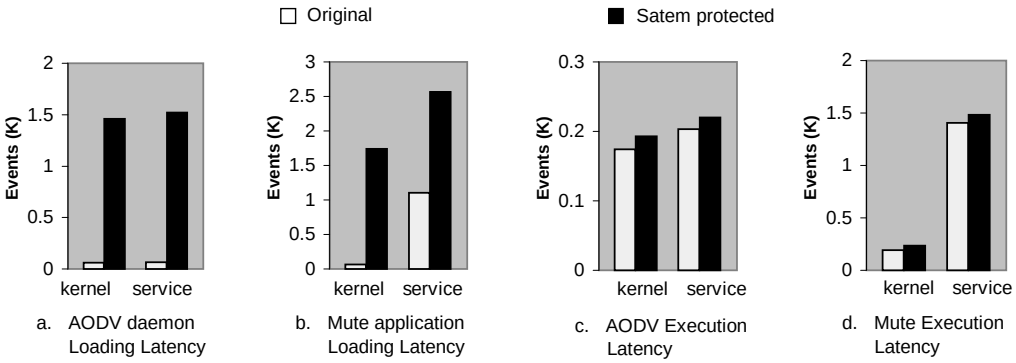

Figure 2.6: Satem Cost of Service Execution

used one laptop to download files from the other. We measured the performance impact in terms of decreased download speed. To minimize the impact of network transmission latency, we used small randomly generated files of 1k bytes. We indirectly tested the performance of the AODV routing service by having laptop 1 constantly ping laptop 3. In the meantime, we periodically deleted the established routes on laptop 1 to force AODV routing to take action. We measured the performance impact in terms of increased ping round trip latency.

As Table. 2.4 shows, the overhead in transactions with a Satem-protected service is small. The reasons are two folded. First, the performance impact on the services imposed by Satem is mainly due to the attestation of the service code, which only takes place once at the time the code is loaded. Once the service is up, it may have already loaded all the code needed to service the requests. Second, in case of code reloading, our lazy attestation mechanism greatly reduces the attestation cost.

2.5.5 Cost of Application Execution

To measure the CPU cost, we turned on `oprofile` [17], which uses the Pentium M CPU hardware performance counter to count `CPU_CLK_UNHALTED` events. For file sharing

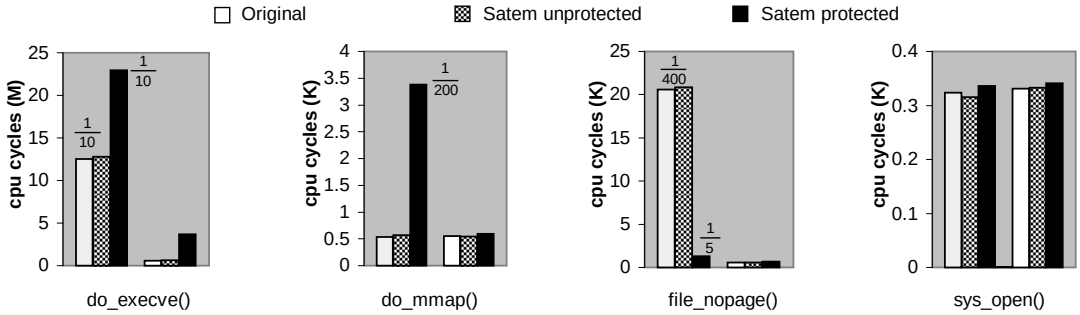

Figure 2.7: Satem Overhead in Kernel Calls

application, we measured the cost that laptop 1 took to download three mp3 songs (roughly 4M each) from laptop 2. For routing application, we ran a simple profiler program, which periodically invalidated the established routes for ten times on laptop 1. The results are shown in Figure. 2.6. In these graphs, the left columns represent the application cost in kernel, and the right columns represent its total cost (kernel and user-level).

Clearly, Satem incurs significant overhead in initial loading of both AODV and Mute to the kernel and to the applications as a whole (Figure. 2.6.a and b). This is because Satem attests each code file being loaded and each mapped memory region of the application. However, since this is a one-time initialization cost, it does not affect the performance of ongoing application transactions. As Figure. 2.6.c and d show, Satem does not incur significant overhead to the applications due to its lazy attestation mechanism.

2.5.6 Overhead in Kernel Calls

To evaluate Satem-enabled kernel calls, we directly read the timestamp counter (TSC). Figure. 2.7 shows the overhead in four kernel calls: `do_execve`, `do_mmap`, `sys_open`, and `filemap_nopage` in terms of extra CPU cycles needed to complete these calls [3]. We measured the cost of these functions in Mute file sharing transactions. All functions are

[3]The cost of the calls vary dramatically. In order to compare them in one figure, we used a reduced scale to plot the following figures: first call to `do_execve` at 1/10, first call to `do_mmap` in *Satem protected* at 1/100, first call to `filemap_nopage` in original and *Satem unprotected* at 1/400, and first call to `filemap_nopage` in *Satem protected* at 1/40.

measured in three cases: in the original Linux 2.6.12 kernel, in the Satem-augmented kernel in which the Mute application is not protected (called *Satem unprotected*), and in the Satem-augmented kernel in which the Mute service is protected (called *Satem protected*). For each function, we measured its overhead for the first call (the left three columns in the figures) and the overhead for subsequent calls (the right three columns in the figures).

The graphs show that the impact of Satem on the first time function call is significant. For `do_execve` and `do_mmap`, the cost is dramatically increased because of the per-page attestation for mapped code files. For `filemap_nopage`, by contrast, the cost is significantly reduced. This is because Satem needs to load every mapped page into the page cache when it attests a protected memory region. As a result, when a page fault occurs, it is very likely that the page is still in the cache. Furthermore, the cost of the subsequent calls in *Satem protected* is greatly reduced. This is also because of the lazy attestation mechanism. Finally, if a function in the Satem-augmented kernel is called from an unprotected service, its performance is comparable to that in the original kernel. This means that Satem does not impact other unprotected programs.

2.6 Case Studies

In this section, we use the three case studies from Section 2.1 to briefly exemplify how our proposed Satem addresses attacks.

Attack 1: Service Spoofing

Satem does not interfere through the attack. At the end of the attack, `/tmp/evanet` is loaded up and listens on UDP port 222. However, when a user requests the commitment on the service, Satem fails to retrieve the commitment since it understands that `/tmp/evanet` is bound to the service port, but this program is not in the PS list. Consequently, the user will cancel the connection because she is unable to evaluate the trustworthiness of the service.

Attack 2: Service Tampering

Satem loads the service commitment immediately after the service is loaded (e.g., in this case after `/bin/vanet` is executed). Due to the service-awareness, Satem tracks the execution of the protected service and attests everything it loads, including `/tmp/libc.so.6`, while ignoring `/lib/tls/libc.so.6` because this library is not loaded by the protected service. If `/tmp/libc.so.6` is defined in the commitment, Satem allows it to be loaded, but the user will be aware of it when evaluating the commitment. Otherwise, if `/tmp/libc.so.6` is not defined in the commitment, Satem prevents it from being loaded.

Attack 3: Post-request Attack

Once the commitment has been evaluated and trust has been granted by the user, Satem will block any code from being loaded by the service, which includes `/bin/eaggregator`.

2.7 Limitations

A challenging task faced by Satem is to ensure trusted handling of user's data. In addition to guarantee trusted service code execution, other mechanisms are needed to achieve this goal. First, the protected service may cooperate with other processes via inter-process communication (IPC). Satem can be extended to cover the code base of the IPC processes by including it in the service commitment. Moreover, the service provider may further call another service on a remote platform. Addressing this problem requires extending Satem to a multi-tiered architecture, in which Satem is enabled on each platform on the connection chain starting from the service provider. Lastly, the protected service may cache user's data in memory or even save it to disk. The problem is further complicated by the fact that the data can be transformed in service transactions. The transformed data may carry the same information as the original one and thereby must also be protected. A possible solution is to track data transformation by use of data lifetime [44] and integrate it into execution monitoring.

Satem is designed to ensure that a protected service can not load untrusted code from the disk. An attacker can exploit, however, buffer overflow attacks to cause the protected service to run arbitrary code without changing its disk image. Satem is

unable to tackle this type of attack. It only mitigates the problem in two aspects. First, Satem may reveal the code which has known buffer overflow vulnerabilities by attesting it to the user. Hence, the user can avoid trusting the vulnerable code. Second, in the case of a successful buffer overflow attack, the attacker runs her own code on the service stack without being caught by Satem. But due to the limited size of the stack, the attacker's code typically has to call other local programs on the service provider to make the attack meaningful. Satem restricts the attacker's capability of launching arbitrary local code (i.e., any code launched by the protected service must be defined in the commitment).

Satem kernel code is not modularized. The main hurdle in face of modularization is the mapping protection mechanism, which involves deep kernel internal calls. This problem can be avoided by building Satem on top of a virtual machine monitor (VMM) [56, 111, 54, 50]. In such a case, the VMM (instead of OS kernel) can control the memory access and perform integrity checks on memory regions. Another advantage of using VMM is to better protect Satem by governing the access to the memory area holding Satem data without limiting use of `/dev/mem` and `/dev/kmem`.

Transactions are not encrypted in the current implementation of the service commitment protocol. This can be solved straightforwardly using `netfilter` and the kernel `crypto` API.

2.8 Summary

This chapter presented Satem, a novel service-aware trusted execution monitor that guarantees trusted application code execution. Users establish trust with the applications through a service commitment protocol executed before starting any new transaction. Satem exploits the TCG-specified TPM as the root of trust to build trusted components in the OS kernel, which consequently enforce the service commitment. Satem achieves service-awareness by limiting the scope of monitoring to protected applications instead of performing attestation and protection on all programs. We have

implemented a prototype under Linux and evaluated it using two MANET applications. The experimental results demonstrate that Satem incurs low overhead to both the applications and the underlying platform. Furthermore, Satem does not impact the performance of unprotected applications.

Satem tackles the problem of trusted applications. In another words, it ensures a user that the application she calls on another node does not and will not run malicious code to attack her. But this does not completely eliminate the user's concern because once she joins the network to run the application she will be exposed to network attacks, such as probes, flood and denial-of-service. As a result, she demands a mechanism to create a protected network that can shield network attacks from reaching her node and other trusted nodes. We will present a solution based on Satem in the next chapter.

Chapter 3

Protected MANET

This chapter presents a Satem-based mechanism to create protected MANETs, in which common network access control policies can be enforced in a distributed manner. We start with the problem description of protecting MANETs in Section 3.1. Then, the architecture is described in Section 3.2 followed by the detailed discussion on the protocols for nodes to join the protected network in Section 3.3. We discuss the implementation of method, evaluate its performance in Section 3.4 and demonstrate the use of our method through a case study in Section 3.5. Finally, we discuss its limitations in Section 3.6 and summarize the chapter in Section 3.7.

3.1 The Problem Statement

A protected network is one that is shielded from unauthorized traffic. A practical method to create a protected infrastructure based network is to enforce network access control policy on a firewall or router. However, this is not possible in MANETs since malicious nodes can simply roam into the wireless transmission range of another node, establish a direct wireless link, and start launching attacks.

As an example, let us consider a group of users who run a peer-to-peer file sharing application across a MANET composed of their smart phones. Through a direct wireless link or ad hoc routing, a malicious user can attempt to exploit a vulnerability in this application to compromise other nodes. Since there are no prior trust relationships between the nodes, it is impossible to identify the untrusted nodes and protect against them. Furthermore, even if these nodes are known and the attacks launched from them are detected, these attacks can still reach the trusted nodes causing depletion of resources on smart phones (e.g., battery). To make things even worse, given the

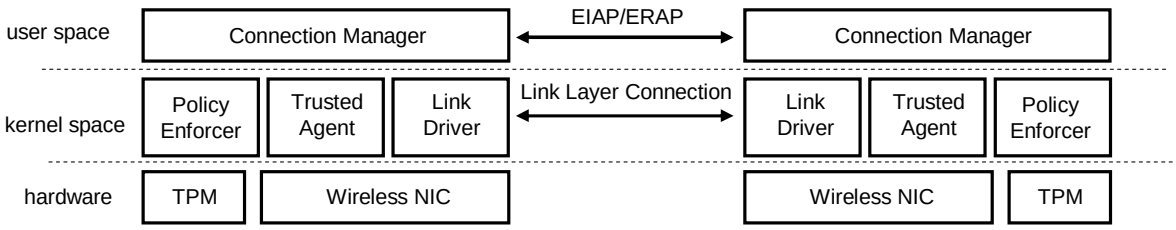

Figure 3.1: The Common Security Architecture of Protected Ad hoc Networks

relatively low network capacity, a single attacker can flood the entire network and make it unusable.

We developed a distributed mechanism that allows trusted nodes to create protected networks in MANETs. A protected network is created to run a specific application and enforce a common network access control policy associated with that application. To become a member in the protected network, a node has to demonstrate its trustworthiness by proving its ability to enforce the network policy. Attacks from untrusted nodes are impossible because these nodes are not allowed to establish wireless links with member nodes. Attacks from member nodes are stopped at the originators by the network policy. The trusted execution of all programs involved in policy enforcement is guaranteed by a kernel agent implemented on top of Satem.

3.2 Security Architecture

The essential idea of our mechanism is to allow only nodes that can be trusted to enforce a common network access control policy to be part of the protected ad hoc network, regardless of whether the node is owned by a trusted entity or not. To accomplish this goal, each node supports a modified version of Satem (i.e., the *trusted agent* is modified for policy enforcement) augmented with a *connection manager* and a *policy enforcer* as illustrated in Figure 3.1. The connection manager establishes the link layer connection and shares the policy with other nodes. The policy is a set of access control rules, and the enforcer is a packet filter, such as Linux netfilter, that can enforce the rules.

We adapted Satem to ensure trusted execution of all code involved in the policy

enforcement. First, according to Satem, the OS kernel (including the trusted agent, the policy enforcer, and the link layer driver) is attested by the system commitment. Second, we define a *link commitment*, which includes the connection manager in its protection scope. Third, we modify Satem's service-oriented commitment protocol and integrate it into the link layer access control protocols. This part of the design will be detailed in Section 3.3.

If a MANET is deployed by an authority, this authority is responsible for defining and updating the policy. To ensure its authenticity, the authority signs the policy and preloads its public key on each node. If the MANET is created spontaneously by a set of nodes (referred to as the creators of the network hereafter), the creators negotiate the policy using existing links. A typical scenario is that the node that initiates the network, drafts the policy and proposes it to the others. The policy can be signed using threshold based methods, such as [124, 28, 70, 40, 120]. When the policy is defined, all creators drop the existing wireless links with each other. These peers then rejoin the network by running the secure protocols described in Section 3.3.

In spontaneously-created MANET, there is no assumption about the correctness or trustworthiness of the policy. All the creators may be malicious, and they form the network just to attract innocent nodes. Therefore, a node may need to evaluate the policy independently based on its own need and capability of enforcing the policy. Since all nodes intending to join the network have the goal of executing a common application, they are motivated to accept a reasonable common access control policy to benefit from the application and protect themselves from attacks.

Each node may already have a local policy to protect itself from malicious nodes. There may be a conflict between the local policy and the network policy. Policy rule conflict discovery and resolution has been extensively studied and many methods are available [24, 74]. In this work, we assume the enforcer is able to use one of these methods.

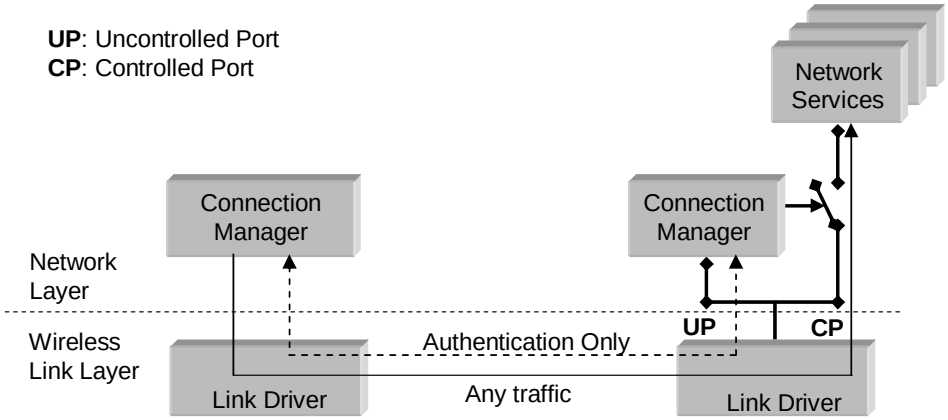

Figure 3.2: 802.11 Dual Port Access Control

3.3 Protocols for Joining Protected Networks

Our approach requires that the link layer have the following properties:

P1: *dual port access control*;

P2: *secure link association*;

The concept of dual port access control is defined in IEEE 802.1x [10], which is supported by IEEE 802.11 (e.g., Windows XP and Linux provide such support). The link layer of a network node has two access ports, uncontrolled and controlled, as shown in Figure 3.2. The controlled port has full access to the link layer, while the uncontrolled port is used only for authentication. Any node in the transmission range can connect to the uncontrolled port in order to authenticate itself. Only after a successful authentication and authorization by the peer node, the initiator can connect to the controlled port.

Once a connection is established, the link layer must ensure that it cannot be hijacked. This is addressed by secure link association. In practice, **P2** is accomplished using link layer encryption (i.e., all frames transmitted over the link are encrypted). Consequently, a node with a single wireless card can be part of only one protected MANET at a time.

We define two protocols that allow a node to join a protected network: *Enforcement Initial Activation Protocol* (EIAP) and *Enforcement Re-Activation Protocol* (ERAP).

Both protocols perform three tasks: (1) verifying the mutual trustworthiness of nodes, (2) sharing the link layer key, and (3) distributing the network policy. EIAP is used for a node to join a network for the first time, while ERAP is used to re-join a network; their main difference is the verification method.

Assuming that a new node N_e wants to join a protected ad hoc network AN by initiating a link layer connection request to a member node of AN, N_m, the high-level view of our security protocols is as follows:

1. N_e calls its connection manager to establish a link layer connection to N_m. The connection manager of N_m grants the connection only to its uncontrolled port.

2. The connection manager of N_e calls the trusted agent to attest the capability and trustworthiness of enforcing AN's policy.

3. The connection manager of N_m verifies the attestation, attests itself and pushes AN's policy P and the link layer key k to N_e. It then grants N_e full access to its controlled port.

4. After verifying N_m's attestation, the connection manager of N_e invokes the policy enforcer to enforce P and finalizes the connection with the link layer key k. Note that the enforcer must be already loaded in the OS kernel.

In the rest of the section, we present the two protocols in detail.

3.3.1 Enforcement Initial Activation Protocol (EIAP)

When a node N_e that has never connected to any node of AN attempts to connect to a member of AN, N_m (we say N_e connects to AN via N_m), N_m activates the enforcement of AN's policy P on N_e through the EIAP protocol. The protocol is illustrated in Figure 3.3.

1. The connection manager of N_e initiates a wireless connection request $CREQ$ to N_m.

2. The connection manager of N_m grants access to its uncontrolled port and replies with a random number, $nonce_m$.

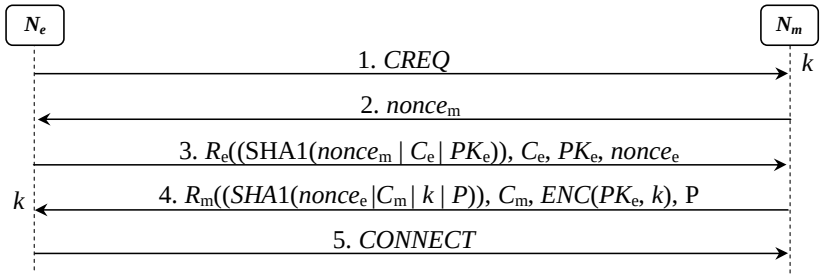

Figure 3.3: The Enforcement Initial Activation Protocol (EIAP)

3. The connection manager of N_e attests itself and delivers the commitments to N_m. To attest itself, the connection manager of N_e calls the trusted agent to generate a TPM report (R_e) of the kernel attestation results with $SHA1(nonce_m|C_e|PK_e)$ as the parameter (| means concatenation). C_e represents the link commitment and the system commitment. PK_e is the public key generated by the connection manager. The TPM signs both the parameter and the attestation results. N_e then generates another random number $nonce_e$ and sends R_e, C_e, $nonce_e$ and PK_e to N_m

4. The connection manager of N_m activates the enforcement of P on N_e. The connection manager of N_m validates R_e to make sure it is generated by a valid TPM. Then, it validates the parameter by recomputing the hash using its stored $nonce_m$ and the newly received C_e and PK_e. Next, it generates a TPM report in the same way as N_e with a parameter $SHA1(nonce_e|C_m|k|P)$, where C_m represents the link and system commitments of N_m, k is the link layer session key, and P is the network access control policy. Finally, the connection manager encrypts the link layer session key k with PK_e, $ENC(PK_e, k)$ and sends it with R_m, C_m, and P to N_e. Then, N_m grants N_e full access to its controlled port.

5. N_e establishes a full connection with N_m. N_e validates R_m the same way as N_m did. Then, it evaluates P before accepting it. Once P is accepted, it is pushed to the policy enforcer, which starts enforcing it immediately. Finally, N_e decrypts k using the corresponding private key SK_e, enables link layer encryption, and

obtains full connectivity to N_m's controlled port.

The EIAP protocol enables the two nodes to mutually verify trustworthiness in enforcing the policy. This is necessary because the link to be established is bi-directional and both nodes need to protect themselves from each other.

Security Analysis. Let us consider a local attacker on N_e (the analysis holds if the attacker is on N_m). We assume that the attacker cannot break the TPM or launch hardware based attacks, and in particular, cannot use direct memory access (DMA). We further assume that the attacker is unable to bypass the node operating system to gain access to system resources, such as memory, CPU, network card, and disk. Other than these restrictions, the attacker can have full control of the software system, including superuser privileges.

Disable enforcement of P. The most direct attack is to disable the enforcement of P after obtaining the connectivity. The attacker can do so by disabling the policy enforcer. This requires removing the policy enforcer's kernel module. The trusted agent intercepts the removal request, clears k, and tears down the link before removing the module. Thus, the attacker has to first disable the trusted agent, which requires rebooting the system.

Modify P. The attacker may attempt to modify the current policy P at runtime. The trusted agent secures the memory space holding P such that only the connection manager have write permissions to it. Additionally, the connection manager is protected by the trusted agent. The attacker may try to run a malicious connection manager. This is allowed only if it is described in the link commitment, which means the link commitment is also untrusted. N_m will receive the commitment at step 3 and refuse to trust the malicious connection manager.

Steal k. The attacker may attempt to steal the session key k on the node. The key is secured by the trusted agent in memory and accessible only to the connection manager and the link driver [1]. The protocol ensures secure distribution of the key. On one hand, the key owner will not distribute the key to any untrusted node (step 4). On

[1]Some drivers provide simple user space utilities for users to read the link layer session key. In our method, these utilities will fail.

the other hand, a node that joins the network will not accept the key from a member node unless the member node has been verified to be trusted (step 5). Consequently, an untrusted node cannot create a key and fool others to accept it.

Hijack k. The attacker may try to steal k in distribution. The public-private key pair is dynamically generated, and the private key SK_e is saved in the connection manager's memory and never disclosed to any other processes. The trusted agent protects the key from being disclosed to any process other than the connection manager. Hence, the attacker is unable to acquire SK_e to decrypt k intercepted at step 4.

Play man-in-the-middle. The attacker may attempt to play a man-in-the-middle attack by replacing PK_e with her own PK_a in order to decrypt k with the private key SK_a she owns. Although PK_e is not authenticated in the protocol, it is attested in the report R_e. The TPM report certifies that the public key PK_e belongs to the system attested by the TPM. Therefore, unless the attacker's system is fully trusted, which makes it impossible to launch attacks, N_m will detect this and refuse to distribute k. The attacker may want to replay a valid TPM report and exploit it to gain trust from N_m. The protocol foils the attacker by including $nonce_e$ and $nonce_m$ in the attestation reports.

3.3.2 Enforcement Re-Activation Protocol (ERAP)

EIAP addresses the scenario where an external node N_e connects to AN via $N_m \in AN$ for the first time. In MANETs, the network topology and the connectivity between nodes may change constantly. For instance, N_e may roam out of the wireless transmission range of N_m, and thereby, it will lose the previously established connection. Subsequently, it may approach another node $N_h \in AN$ and re-establish connection to AN via N_h. This brings up two issues, which make the use of EIAP unsuitable for connection re-establishment.

First, the network policy may be updated to accommodate certain changes in the network, which causes the local copies on N_h and N_e to be inconsistent. We need to ensure that only the most recent version of P is enforced. This problem can be solved by assigning a version number v to P, which is incremented every time P is updated.

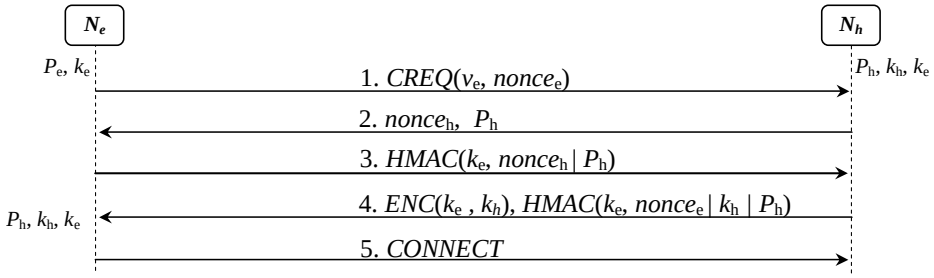

Figure 3.4: The Enforcement Re-Activation Protocol (ERAP)

Second, the nodes enforcing old policies should be disconnected from the network. This can be done by having the nodes enforcing the new policy update their link layer session key. As a result, when the node rejoins the network, it has to request the new key to re-establish the connectivity.

EIAP can solve the above problems, but it is too costly due to the need to generate the TPM report and transmit large data via a partial link. Therefore, it is desirable to optimize the protocol for the link re-establishment scenario. We observe that both N_e and N_h must have been verified before they were allowed to connect to AN for the first time. Owning an old key implies that it has been protected by the trusted agent. Otherwise, if any program defined in the commitments had been compromised since last connection, the trusted agent would wipe out the key. Therefore, we can let each node hold the past V keys and use them to compute a keyed message authentication code (HMAC) to prove its trustworthiness. Figure 3.4 illustrates the Enforcement Re-Activation Protocol (ERAP) assuming N_h has the latest policy.

1. N_e sends a request to N_h to establish a full link. N_e includes its policy version v_e and a random number $nonce_e$ in the request. N_e cannot lie about its v_e since it will be verified later by N_h.

2. N_h grants access to its uncontrolled port. N_h replies with another random number $nonce_h$. It also compares its policy version v_h with v_e and includes the latest policy P_h in the reply if $v_h > v_e$.

3. N_e authenticates to N_h. If N_e authenticates P_h, it learns that its policy needs to

be updated. It computes $HMAC(k_e, nonce_h|P_h)$ and sends it back to N_h.

4. N_h verifies N_e and distributes k_h to N_e. Verifying $HMAC(k_e, nonce_h|P_h)$ proves that N_e holds k_e and P_h. N_h then encrypts k_h with k_e ($ENC(k_e, k_h)$) and sends the encrypted key with $HMAC(k_e, nonce_e|k_h|P_h)$.

5. N_e verifies N_h and establishes a full link with N_h. Verifying $HMAC(k_e, nonce_e|k_h|P_h)$ convinces N_e that N_h holds k_e and enforces P_h. N_e overwrites P_e with P_h received at step 2. N_e then decrypts $ENC(k_e, k_h)$ and obtains k_h. It can now enable link layer encryption and establish full connectivity with N_h.

In case of $v_h < v_e$, the roles of N_e and N_h are just swapped. N_h sends v_h instead of P_h at step 2. N_e sends P_e at step 3 with $HMAC(k_h, nonce_h|P_e)$. The rest is similar.

Security Analysis. As discussed in EIAP, the attacker is unable to break the connection manager, the policy enforcer, or the trusted agent. As a result, she cannot obtain the link layer session key without enforcing the policy. Additionally, she cannot modify the policy or steal the link layer session keys on the machine either. So, the new key distributed at step 4 is safe.

The number of old keys kept on each node, V, is a design parameter. If a node has missed too many policy updates, it cannot re-connect to the network through ERAP since none of the nodes enforcing the latest policy holds the old keys any more. In this case, it has to join as a new node through EIAP.

3.4 Prototype Implementation and Evaluation

To show the feasibility of our mechanism, we implemented and evaluated a prototype that works over IEEE 802.11-based networks. The EIAP and ERAP protocols were implemented in the connection manager as extensions of IEEE 802.1x. To implement the connection manager, we modified `xsupplicant` [15], an open source 802.1x client and `hostapd` [9], an open source 802.1x server. The two connection managers conduct the protocol negotiation via the `EAPOL` protocol [23]. We used the built-in netfilter as the enforcer.

Scenario	Latency (in seconds)
802.11 WEP	1.2
EIAP	3.1
ERAP	1.9

Table 3.1: 802.11 Link Establishment Latency in Protected Ad hoc Network

Scenario	Download Speed (KB/second)
Open	235.23
WEP	230.57
Our Solution	229.42

Table 3.2: Performance of Data Communication over Protected Ad hoc Network

To simplify the implementation, we used 802.11 WEP [1] to encrypt the link. Extensive research studies showed that WEP is insufficient to guarantee secure association [52, 82, 26]. Other types of stronger encryption, such as WPA [1], can be used to replace WEP. For instance, integration with WPA is straightforward, but more configuration work in `xsupplicant` and `hostapd` is needed.

Our method incurs a certain latency for link layer connection establishment and data communication. To measure them, we used two nodes that create an IEEE 802.11b ad hoc network for the application described in Section 3.5. The source node N_S is an IBM R40 laptop with a 1.3Ghz Pentium M CPU, 512M RAM, an Intel IPW2100 wireless card, and an Atmel TPM. The destination node N_D is an IBM T43 laptop with a 1.7Ghz Pentium M CPU, 512M RAM, and an Atheros wireless card.

The link establishment latency is shown in Table 3.1. Compared to the link layer connection establishment in the standard 802.11b with 104 bit WEP authentication, EIAP incurs a high overhead in the initial link establishment. This is mainly due to the cost of collecting attestation reports and the negotiations over `EAPOL`. As expected, the overhead of ERAP is significantly reduced because it does not perform the costly trust verification. We believe that these results are acceptable, especially because the high overhead imposed by EIAP is just a one-time cost. After that, the typical overhead is reduced through the use of ERAP for reconnections.

The overhead of joining is insignificant for the overall wireless communication performance because connection establishment happens only once in a while even in a volatile network. The cost that dominates the overall network performance is the latency of data communication. To quantify this cost, we measured the download speed of Mute in three networks: standard open 802.11b, standard 802.11b with WEP, and a protected network that enforces the policy presented in Figure 3.5. Since we measured the cost when the link was fully established, this cost is relatively fixed per packet (i.e., policy enforcement). Therefore, using large files increases the accuracy of the cost estimation. In the test, we let node N_S download 256M files from N_D.

As Table 3.2 shows, our method incurs small performance degradation compared to both the open 802.11b (2.47%) and WEP-based 802.11b (0.5%). This result is due to the fact that our method does not incur any costs besides WEP encryption and packet filtering, which are very light-weighted. Another reason is the simplicity of the policy being enforced.

3.5 Case Study

To illustrate our mechanism, we consider a simple ad hoc network created for a peer-to-peer file sharing application, namely Mute [12]. For instance, a group of students on-campus can use their 802.11-enabled smart phones to create such a network. The students do not know each other and want to protect themselves from being attacked by malicious peers. Since the network is formed for a specific application (i.e., Mute), the network access control policy must deny any other connection request from different applications. Additionally, even though Mute connections are accepted, the policy must protect the nodes against attacks that target Mute, such as flooding.

Figure 3.5 shows an example of a policy for this application expressed in pseudo netfilter rules. We assumed that Mute runs on TCP port 5000. Furthermore, to allow multi-hop communication, the network runs the AODV [91] routing protocol on UDP port 654. Each computer uses interface `wifi0` to join the ad hoc network.

The rules $\mathcal{R}2 - 6$ allow only Mute and AODV traffic to be sent from each node.

```
IN
ℛ1   Mark=1 TCP from any to wifi0 !Local_IP:5000

OUT
ℛ2   Allow TCP SESSION from wifi0 to any:5000
ℛ3   Allow TCP SYN from wifi0 to any:5000 limit 3/s
ℛ4   Allow UDP from wifi0 to any:654 limit 10/s
ℛ5   Allow TCP from wifi0 to any:5000 Mark==1
ℛ6   Drop TCP or UDP from wifi0 to any

FORWARD
ℛ7   Drop from any to wifi0
```

Figure 3.5: The Policy for Protected File Sharing Network

Hence, attacks to other services (e.g., default services like `netbios`) are impossible. The attacker could try to exploit one of the allowed services: Mute and AODV. A direct attack is to flood Mute, but this is significantly limited by $\mathcal{R}3$, which allows a node to initiate at most three TCP connections per second. Once a session has been established, $\mathcal{R}2$ allows packets to be sent at any rate. Similar to $\mathcal{R}3$, $\mathcal{R}4$ protects AODV from being flooded. All these attacks are stopped at their originators without having any impact on the target node or the network. This is impossible through receiver side protection.

$\mathcal{R}1$ and $\mathcal{R}5$ enforce hop-by-hop packet forwarding for Mute traffic. Per $\mathcal{R}1$, the node marks an incoming (pre-routing) Mute packet of which it is not the destination. This means that another node uses it as a router. Per $\mathcal{R}5$, the marked packets are allowed to be forwarded. In a more complicated scenario, some protected nodes may have multiple wireless network cards and be in multiple networks. The attacker may leverage this fact to launch attacks from an unprotected network to nodes in the protected network. $\mathcal{R}7$ stops this attack by forbidding packet forwarding across different networks.

Policies, such as the one described in this section, can be enforced by built-in kernel filters (e.g., netfilter). These filters can easily be extended by adding modules and thereby can support more complicated policies. For example, if the application is vulnerable to a buffer overflow attack and the attack signature is known, one can implement an extended module to check the specific signature in the packet and stop the attack at the originator. Consequently, our method is flexible and extensible.

3.6 Limitations

Runtime intrusion. The security of our method largely depends on the security of the underlying trusted system, which only prevents untrusted code from being loaded from the disk. Therefore, our approach is unable to tackle runtime intrusion exploiting code vulnerabilities, such as buffer overflows. As we mentioned in Section 2.7, the method only mitigates the problem by revealing the vulnerable code in the Satem report and by restricting the attackers from running arbitrary local code.

Attacking the uncontrolled port. Neither EIAP nor ERAP prevents the attacker from sending link layer frames to the uncontrolled port of its one-hop away neighbor. The attacker can leverage this weakness to flood its neighbor nodes. We address this weakness from two perspectives. First, flooding the uncontrolled link layer port is by far less effective than flooding the network layer. This is because in the former case, the attacker can only target a small number of nodes in her one-hop vicinity. In the latter case, she can target any node in the network and exploit widely distributed denial-of-service slaves to dramatically amplify the damage. Hence, our method addresses the main and most severe threat. Second, this weakness can be effectively addressed by host based countermeasures [32]. For instance, limiting the rate of accepting connection requests can foil resource depletion attacks.

Attacking the connection manager. The attacker may flood the connection manager of the protected node with overwhelming connection requests. To mitigate the problem, the protected node can limit the rate of handling the connection requests to bound the resources spent on EIAP and ERAP processing. In addition, EIAP and ERAP require the connection requester to do the attestation first at step 3. Since attestation is much more costly than verification, the difficulty of the attack is increased. The attacker has to either use a high-end computer or control a large number of low-end computers.

Kernel update. During the time the node maintains a link with the network, loading new kernel modules is limited by the system commitment. For instance, if the node obtains a new driver that is not defined in the system commitment, it cannot load

the driver until it reboots the system.

3.7 Summary

This chapter presented a mechanism for creating protected ad hoc networks. The creation of such networks is triggered by users who want to run a common application. Our mechanism does not allow untrusted nodes to establish wireless links with nodes in the protected networks. Furthermore, it enforces a common network access control policy in the networks; this policy is associated with the application running in the networks. Attacks from member nodes are suppressed locally by the common network policy. To ensure trusted enforcement of the policy, we augmented every node with a trusted kernel agent based on the TPM. We evaluated the method through a prototype based on an IEEE 802.11 ad hoc network. The results demonstrate that our method imposes little impact on network communication.

The protected network ensures trusted communication at low level since it only prevents each network member from being attacked at network layer. It lacks knowledge about the application protocols and is unable to handle the complex interdependence between them. Therefore, this method is not sufficient to ensure secure and proper collaboration between network nodes through the application. In the next chapter, we will present a policy enforcement mechanism to address that problem.

Chapter 4

Trusted Policy Enforcement

In this chapter, we further extend the idea of enforcing application centric network policy for MANETs to all application layers by designing and implementing a trusted policy enforcement framework on top of Satem. To begin with, we motivate the research with the need of the method of enforcing application specific policies in MANETs in Section 4.1. Then, we introduce the trusted multi-tier network followed by the methods to create it in Section 4.2 and 4.3. Next, we discuss the prototype implementation of the policy enforcement mechanism and evaluate its performance in Section 4.4. Finally, we discuss the limitations in Section 4.5 and summarize the chapter in Section 4.6.

4.1 The Problem Statement

In addition to protecting network nodes from being attacked, a key to the success of MANET applications is a mechanism assuring secure communication and proper collaboration among all participant entities at the application level. To achieve this goal, communication policies that govern the interactions between entities must be defined and enforced. Although significant research work and breakthroughs have been done in the area of security policies, they mainly focus on providing sufficient expressive power to represent policies [35, 34]. The challenge that remains unsolved for MANET applications is how to enforce such policies. We illustrate the problem through three MANET applications and show the difficulty of applying existing solutions to them.

4.1.1 Example 1: Secure Routing

Consider a group of nodes supporting AODV routing protocol. AODV is known to be vulnerable to wormhole attacks [60], in which an attacker exploits a fast tunnel to

attract all network traffic through it. One way to defeat this attack is to implement Packet Leashes [60]. For example, a geographical leash can ensure that the destination node is within a certain distance from the source node. It is implemented as follows:

> *The source node r checks for each AODV reply from the destination node s, $d_{max} > ||p(s) - p(r)|| + 2(t_r - t_s) \times v + e$, where d_{max} is the max distance that the destination node s is allowed from the source node r, $p(s)$ is the position of s at t_s, the time of sending the AODV packet, $p(r)$ is the position of r at t_r, the time of receiving the AODV packet, v is the maximum relative moving speed of the two nodes, and e is the acceptable error. Replies that do not pass the check are deemed as from wormholes and rejected.*

We can directly translate the above leash into a routing policy $\mathcal{P}_R$. However, the implementation of the above leash or enforcement of the policy $\mathcal{P}_R$ requires that node r and s be loosely synchronized and r can authenticate s. In general, this is non-trivial in MANET due to the lack of a central time server. In case of anonymous environment, this becomes more difficult since the two nodes cannot trust each other. The node has to rely on round-trip delay to estimate the time needed for an AODV message to reach the other. However, this method will accumulate large errors with number of hops and distance between the two nodes increasing. Therefore, the best place to detect the wormhole is on the node that is close to either end of the tunnel. The further away the node is, the less precise the estimate becomes, and the higher false positives and negatives the method incurs. But this is infeasible since we do not know which node is close to the tunnel and whether this node can be trusted to check the leash.

4.1.2 Example 2: Unselfish Sharing

Consider cars on a highway forming a vehicular network to obtain traffic information ahead of them [46]. Each node simultaneously posts queries, answer queries, receives responses, and forwards queries for others. To benefit all cars in the network, it is vital to ensure that enough cars respond to and relay the queries posted by others. Similar concerns exist in other applications, such as a P2P file sharing network where sufficient

file providers are desired. To achieve these goals, each node must abide by a policy $\mathcal{P}_F$, like the following, before joining the network:

> *Every mobile node has to serve or relay at least 1 request from others after posting 3 queries to the network.*

Clearly, the only way to enforce the policy is to do it on every node in the network. Due to the anonymous nature, any identity based policy enforcement method, such as Peace [69], does not apply.

4.1.3 Example 3: Fair Game

Consider a group of smart phones using a MANET to play a game. They are separated into n teams and each of them chooses to join one of the teams at the beginning of the game. To ensure that each node can only take one role in the game, the following game policy $\mathcal{P}_G$ is defined:

> *Each gaming node is free to join any of the n teams. But once it joins one, it can not join another team without first withdrawing from the current team.*

The above policy is similar to Chinese Wall Policy [37]. Enforcing such a policy for Internet based applications has been addressed in literatures, such as [81]. The existing methods rely on the capability of differentiating one node from another. However, due to Sybil attack [49], this is difficult in MANET.

It is difficult for existing methods to enforce any individual policy aforementioned. To make it more challenging, these policies can be related. For example, secure routing may be prerequisite to secure the file sharing and gaming applications. So, enforcing the file sharing or gaming policy requires that the underlying routing policy have already been enforced. On the other hand, a node may run the file sharing application side by side with the gaming application. Enforcing the policy for one should not interfere with the other. Since nodes can run these applications in any combinations, it is critical to enforce their associated policies flexibly and organically.

4.2 Overview of Trusted Multi-tier Networks

In this section, we first formally define the trusted multi-tier networks. Then, we illustrate the idea by a two-tier network example. Finally, we describe how to create such a network.

4.2.1 Definition and Policy Enforcement

For some application S, we define the trusted policy enforcing tier $\mathcal{T}_0$, as follows:

$$\mathcal{T}_0 =< N, S, P >$$

where N are the set of nodes communicating through S, and P is the policy defined for S. To facilitate description, we use "." to represent "member of" relation, i.e., $\mathcal{T}_0.N$ means the set of nodes in the tier $\mathcal{T}_0$.

Assume S calls a set of h independent protocols and each protocol is associated with a policy, the nodes running these applications and enforcing their associated policies form $\mathcal{T}_0$'s underlying tiers $\mathcal{T}_1$, $\mathcal{T}_2$, ... $\mathcal{T}_h$. Similarly, each of these protocols may also depend on other protocols and therefore, may have its own underlying tiers. Assume that there are totally m direct and indirect underlying tiers of $\mathcal{T}_0$, these $m+1$ tiers form the trusted multi-tier network of S defined as follows:

$$\mathcal{N} =< \bigcap_{i=0}^{m} \mathcal{T}_i.N, \ \bigcup_{i=0}^{m} \mathcal{T}_i.S, \ \bigcup_{i=0}^{m} \mathcal{T}_i.P >$$

Each policy is enforced at its associated trusted tier independently. Each trusted tier $\mathcal{T}_i$ ensures both compliance and authenticity of the messages in $\mathcal{T}_i.S$ as follows:

1. **Compliance.** For each member node of $\mathcal{T}_i$ to send a message in $\mathcal{T}_i.S$, it must be permitted by $\mathcal{T}_i.P$.

2. **Authenticity.** For a message of $\mathcal{T}_i.S$ to be accepted by a member node of $\mathcal{T}_i$, it must be sent by another member node.

Compliance ensures that all member nodes abide by $\mathcal{T}_i.P$ in communicating with each other through $\mathcal{T}_i.S$. This is accomplished because only nodes that are trusted to

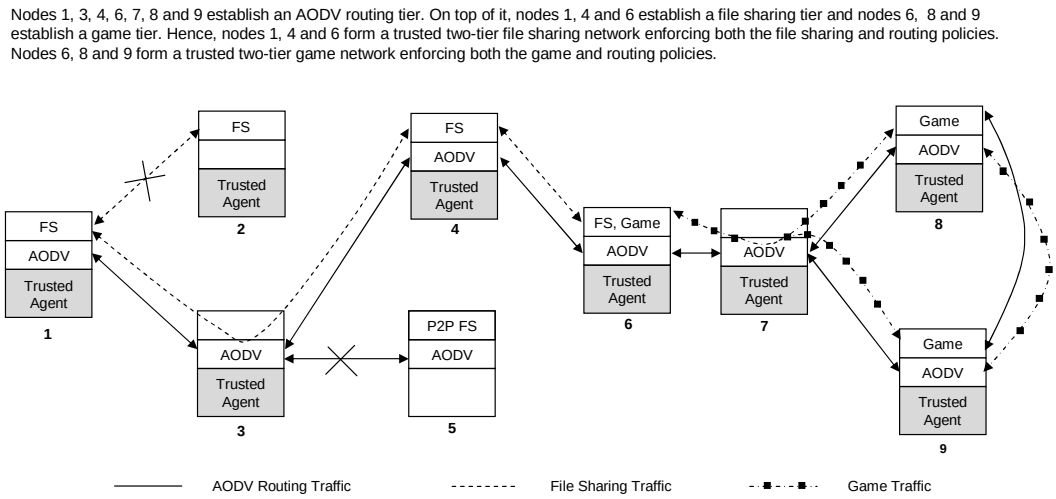

Figure 4.1: Policy Enforcement in Multi-tier Networks

enforce P can join the trusted tier. Once the trust is established, the node's underlying trusted computing system ensures that it will not be compromised. Otherwise, the node will lose its membership of the trusted tier. We will discuss how these are accomplished in the next section. Authenticity prevents a non-member node from creating and injecting messages to the trusted tier. To achieve this, the enforcer on the node attaches a Message Authentication Code (MAC) to each message X of S it sends out. The trusted tier key k_T is used to compute the MAC code e.g., $M_T(X) = HMAC(k_T, X)$. k_T is created when $\mathcal{T}$ is established and shared by all member nodes in $\mathcal{T}$. We will discuss more on the trusted tier key in the following sections.

Example: Two trusted two-tier networks

In order to understand the main idea of our solution, let us first consider the example presented in Fig. 4.1. This example shows a group of nodes using a MANET to run a file sharing and a game application, denoted by F and G. Both applications rely on AODV routing denoted by R. To address the issues described in the previous section, the nodes can build two trusted two-tier networks: (1) a file sharing network $\mathcal{N}_F$ consisting of a file sharing tier $\mathcal{T}_F$ and a routing tier $\mathcal{T}_R$; and (2) a game network $\mathcal{N}_G$ consisting of a game tier $\mathcal{T}_G$ and the same routing tier $\mathcal{T}_R$. A node can join more than one multi-tier networks at the same time (e.g., node 6 in this example). Nodes in each tier must

enforce the tier policy, P_R for $\mathcal{T}_R$, P_F for $\mathcal{T}_F$, and P_G for $\mathcal{T}_G$. Formally, $\mathcal{N}_F$ and $\mathcal{N}_G$ are defined as follows:

$$\mathcal{T}_R =< \{1,3,4,6,7,8,9\}, R, P_R >$$

$$\mathcal{T}_F =< \{1,4,6\}, F, P_F >$$

$$\mathcal{T}_G =< \{6,8,9\}, F, P_G >$$

$$\mathcal{N}_F =< \{1,4,6\}, \{R,F\}, \{P_R, P_F\} >$$

$$\mathcal{N}_G =< \{6,8,9\}, \{R,G\}, \{P_R, P_G\} >$$

Enforcing P_R, P_F, and P_G is no longer a problem in $\mathcal{N}_F$ and $\mathcal{N}_G$ because they are enforced on every member node of them. For P_R, if there is a wormhole in the networks, the node closest to the wormhole will check the leash and detect the existence of the wormhole. Enforcing P_F and P_G is also straightforward, since the history of the node posting queries, serving requests and registering its identity is available on this node.

For two nodes to communicate, they have to be in the same multi-tier network. For example, in Fig. 4.1, node 1 cannot share files with 3 because node 3 does not enforce P_F and is not a member of the trusted two-tier file sharing network. Neither can files be shared between node 1 and 2 as node 2 does not join the underlying trusted routing tier. On the other hand, the two nodes do not have to be neighbors, as the higher tier application traffic can be routed by the trusted lower tier in a multi-hop fashion. For example, node 1 and 4 can share files securely by routing through node 3. Node 3 is trusted to enforce P_R even though it is not trusted to enforce P_F. Nodes in any trusted multi-tier network must have the trusted agent. Otherwise, they cannot join any trusted tier, such as node 5.

4.2.2 Creating a Trusted Multi-tier Network

Building a trusted multi-tier network involves establishing all the trusted tiers it is composed of in a bottom-up fashion. For example, to build the file sharing multi-tier network $\mathcal{N}_F$ in Figure 4.1, the trusted AODV tier $\mathcal{T}_R$ is first established followed by the trusted file sharing tier $\mathcal{T}_F$. Fig. 4.2 illustrates this procedure.

Building the trusted 2-tier file sharing network demands creating the AODV tier τ_R followed by the file sharing tier τ_F. To create the AODV routing tier τ_R, node 1 initiates τ_R and invites its neighbor nodes (2 and 3) to join the tier. At step 2, node 2 joins τ_R and further invites its neighbors (node 4 and 5). Finally node 6 joins the tier at step 3. The file sharing τ_F is then built similarly on top of the AODV tier.

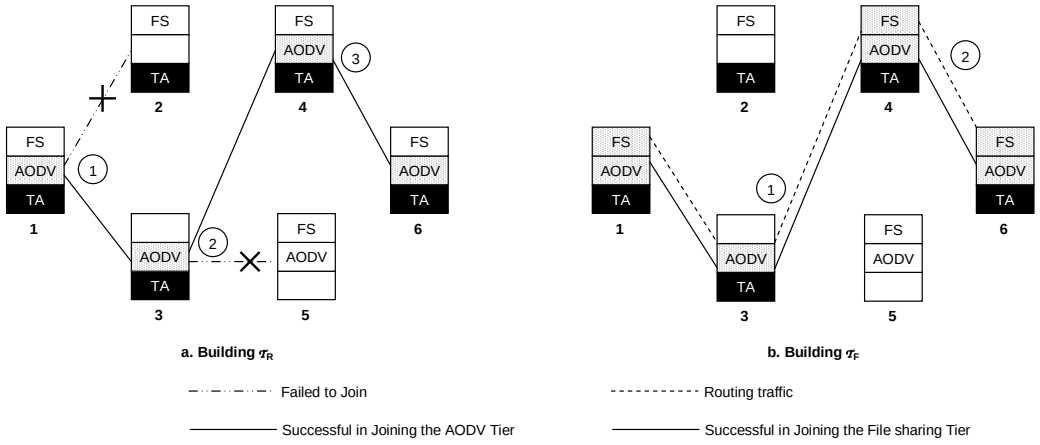

Figure 4.2: Creation of Trusted Multi-Tier Network

A tier is created step-by-step. First, a node begins to enforce the tier policy. It creates the tier key, which is used to authenticate in-tier communications as discussed earlier. By doing so, it becomes the first member of the tier, called *originator* of the tier, e.g. node 1 in Fig. 4.2. The originator then broadcasts an invitation to its neighbors, e.g. node 2 and 3, to join the newly created tier. Assume node 2 and 3 choose to join this tier. Since node 3 enforces P_R, it succeeds in joining the tier and receives the tier key from node 1, but node 2 fails because it does not enforce P_R. Next, node 3 extends the tier one step further by inviting nodes 4 and 5. Similarly, node 4 joins and continues the process to include node 6 in the tier. The tier originator controls the size of the tier by setting a TTL parameter in the invitation message. Each node decrements the TTL after joining the tier and stops forwarding the invitation message once the TTL becomes zero. The joining procedure is defined in JOIN protocol, which will be discussed in details in next section.

Once the routing tier is built, the upper-layer file sharing tier can be built in a similar way. The difference is that the broadcast is in a multi-hop manner. That said, a member node of the file sharing tier broadcasts the invitation to its neighbors. If a neighbor node decides to join the tier, it re-broadcasts the invitation in the same way as

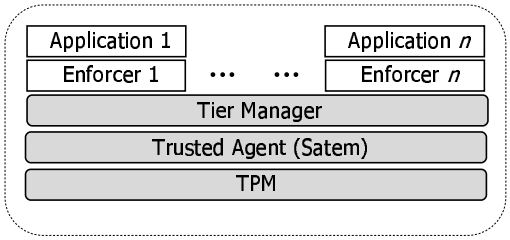

Figure 4.3: Node Architecture of the Trusted Multi-tier Network

in the routing tier creation. Even if the neighbor node does not join the tier, it forwards the invitation message to its neighbors and acts as a router for further communication between the sender and other potential members of the new tier.

The policies are enforced by each node in the multi-tier trusted network rather than by a trusted central authority. Therefore, it is critical to verify a node's trustworthiness of enforcing every tier policy. This is accomplished when the node joins the network through two protocols: JOIN or MERGE, which will be discussed in details in the next section.

4.3 Node Architecture and Protocols

In this section, we introduce the node architecture of our method. As shown in Fig. 4.3, it consists of a trusted agent (Satem), a tier manager and a number of enforcers, each of which enforces a tier policy. We then discuss in details the two protocols: JOIN and MERGE, followed by the analysis of their correctness.

4.3.1 Trusted Agent

We leveraged Satem [117] to build the trust agent, which guarantees trusted policy enforcement. Similar to the Protected Ad hoc Networks discussed in Chapter 3, Satem ensures that the underlying platform and the policy enforcing software components, including the tier manager and all the enforcers, will only execute trusted code. This is fulfilled by defining a *system commitment*, which describes all the code files the kernel and the tier manager may load in runtime, and an *enforcer commitment* for each

enforcer the system runs, which describes all the code files the enforcer executes.

The trusted agent enforces the system commitment at boot time and the enforcer commitment upon being started, such that the kernel and the enforcer are forbidden to load any code files that are either undefined in the commitment or tampered with. Therefore, if the requester verifies that the kernel, the agent, and the commitments are trusted, it is convinced that (1) the enforcer has executed only trusted code up to the time of attestation; and (2) the enforcer will continue to do so in the following phases due to the protection provided by the trusted agent.

4.3.2 Tier Manager and Enforcer

The tier manager is an application that allows the node to create, join and merge into a tier. When the user decides to create a new tier, she calls the tier manager to create the tier key and start the tier creation procedure. Then, the tier manager communicates with the tier managers on other nodes through the JOIN or MERGE protocol. The node may join multiple tiers and thereby run multiple enforcers. An enforcer is any software that can enforce the tier policy. In the simplest form, the tier application itself has built-in capabilities of enforcing certain policies and can be the enforcer. Both the tier manager and the enforcer must be trusted. They are defined in the system and enforcer commitments, respectively, and protected by Satem.

Before creating or joining a tier, the user first registers the tier enforcer with the tier manager. As explained later in JOIN protocol, this enables the tier manage to deliver the correct enforcer commitment. Moreover, at the end of the JOIN and MERGE protocol, the tier manager receives the tier key. Then, the tier manager can deliver the key to the right enforcer that has been attested by the trusted agent.

4.3.3 Joining a Tier

The JOIN protocol is for a node to join a trusted tier. For a node to join a trusted tier for the first time, it has to be verified by an existing tier member for its trustworthiness of enforcing the tier policy. At the same time, the node must also verify the trustworthiness of the member node because a policy enforcing node only communicates with other

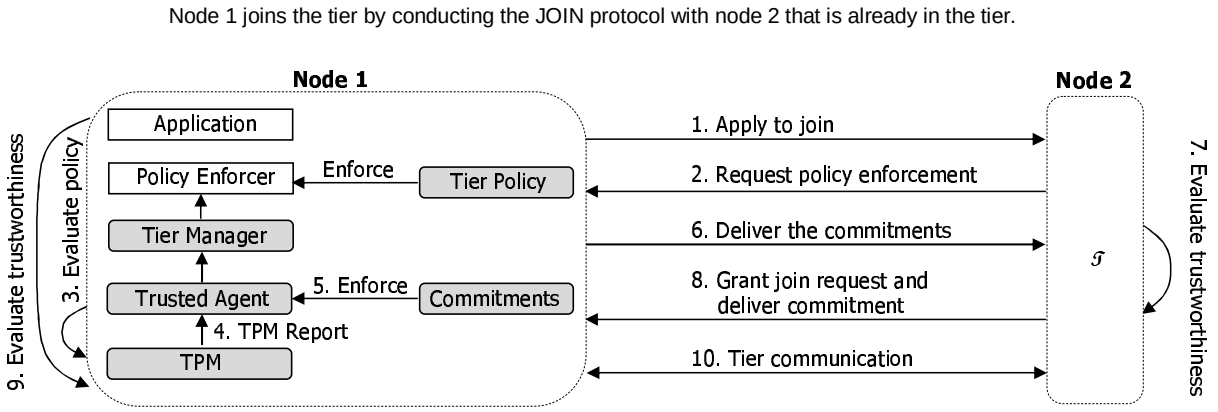

Figure 4.4: JOIN Protocol of Trusted Multi-tier Network

policy enforcing nodes. Figure 4.4 illustrates the JOIN protocol. In the figure, we assume that Node 2 is already a member of a trusted tier and Node 1 wants to join this trusted tier. The JOIN protocol works as follows:

1. Node 1 sends a join request to Node 2 by specifying the application identity (e.g., the IP address and port number).

2. Upon receiving the join request, Node 2 replies with a request for a guarantee of trusted enforcement of the tier policy.

3. Node 1 evaluates the policy and decides whether it can be enforced.

4. If Node 1 accepts the policy, it calls the trusted agent to generate a Satem report, including (1) its system commitment, (2) the enforcer commitment (i.e. the service commitment defined for the enforcer), and (3) the attestation of booting.

5. Node 1 enforces both the system commitment and enforcer commitment, which guarantees that Node 1 can only load trusted code to enforce policy P.

6. Node 1 sends the Satem report to Node 2 for evaluation.

7. Node 2 first authenticates and verifies the integrity of the commitments and the attestation report. Then, it verifies the system commitment, the enforcer commitment, and the boot attestation in the Satem report against the local trust

policy before accepting Node 1 to the tier.

8. When everything has been verified, Node 2 sends the trusted tier key along with its own Satem report to Node 1.

9. Node 1 verifies the report the same as Node 2 did at step 7. When it is verified, Node 1 sets the tier key and enforces the tier policy. As discussed in Section 4.2.1, the node uses the tier key to ensure authenticity of messages exchanged with other nodes in the same tier.

4.3.4 Merging Tiers

The MERGE protocol is for two separate tiers that enforce the same policy to merge into one unified tier. Every node has an equal opportunity to establish a trusted tier. To prevent multiple nodes from establishing the same trusted tier at the same time, the originator may first query its neighborhood for the existing tier. However, this method does not work if two nodes are not reachable from each other. As a result, they will create two trusted tiers running the same application and enforcing the same policy, but holding different trusted tier keys. When later connectivity becomes available between them, they will not be able to communicate with each other. In this case, the two trusted tiers can be merged by unifying the two tier keys into one common key. This procedure starts when a node (Node 1) in a tier (tier A) learns the existence of another tier (tier B) nearby. For instance, it may receive a message from Node 2 that it cannot authenticate. This may indicate that Node 2 is running the same application but having a different key. To verify whether tier B is enforcing the same policy, Node 1 exchanges its policy with Node 2. If the policies are the same, Node 1 starts the MERGE protocol to unify the key. Fig.4.5 shows the overview of the MERGE protocol.

1. Node 1 negotiates with Node 2 the new key to be used by the merged trusted tier. They compute a hash of their own keys and select the key with greater hash value as the new trusted tier key.

The old tier and the new tier enforce the same policy but were created separately and have different tier keys. Node 1 in the old tier joins the new tier by conducting the JOIN protocol with Node 2. Then Node 1 calls for other nodes (e.g. Node 3) in the old tier to merge into the new tier through the MERGE protocol.

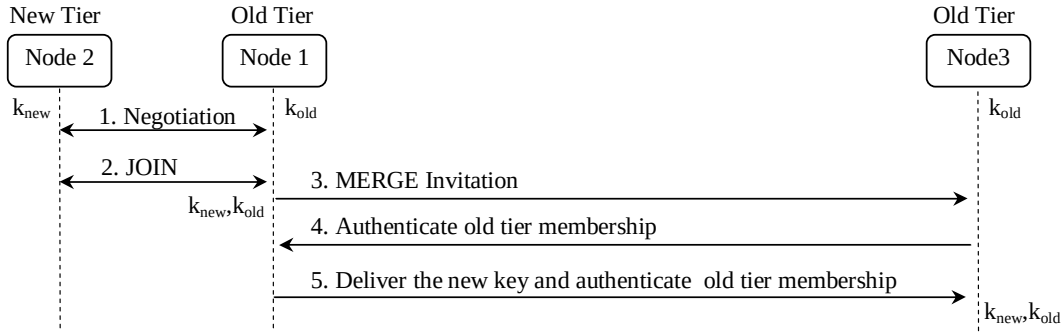

Figure 4.5: Merge Protocol of Trusted Multi-tier Network

2. Assume the key of Node 2, k_{new} is chosen, then Node 1 initiatives a JOIN process with Node 2. Similar to establishing a new tier, both nodes mutually verify that they are enforcing the same policy. Furthermore, after receiving the key, Node 1 must verify it against the hash received in the previous step. Then, Node 1 will still keep the old key, k_{old}, for a certain period of time.

3. After joining the new tier, Node 1 broadcasts a MERGE message to its neighbor nodes in its old tier, which includes a random nonce. Similar to the tier creation procedure, the broadcast is in a multi-hop manner. For instance, the MERGE message of the file sharing tier can be delivered to other nodes in the old tier through a series of AODV router nodes that are not in either of the new and old file sharing tier.

4. Upon receiving a MERGE message, a Node 3 in the old tier responds with a message authentication code (MAC) over the nonce using its tier key, k_{old}, to Node 1. Along with the MAC, Node 3 also sends another nonce. This answer serves as a request to merge into the new tier.

5. Node 1 checks the MAC using the saved old key, k_{old}, which verifies Node 3's membership of the old tier. It then computes a new MAC over the nonce sent by Node 3, encrypts its new tier key, k_{new} with the old key, and sends them back to

Node 3.

6. Node 3 checks the MAC code, which verifies that Node 1 was in the same old tier. It then decrypts the new key and merges into the new tier.

4.3.5 Protocols Analysis

Trusted tier key distribution. A key task of the trusted tier establishment is to generate the trusted tier key, which will be shared by all tier members. In the JOIN protocol, the member node delivers the tier key to the requesting node after verifying its trustworthiness at step 8. It must be protected from being stolen in transmission by another unauthorized node or on the member node by unauthorized processes. The method to achieve this is similar to that used in the Protected Ad hoc Network as discussed in Chapter 3.3. In details, the tier manager on the requesting node dynamically generates a public-private key pair when being started. The private key is protected by Satem in memory and is only accessible to the tier manager. Satem binds the public key with the Satem report. Thus, the member node is assured that the public key is owned by a Satem protected node. The member node can securely deliver the tier key by encrypting it using this public key, because only the requesting node has the corresponding the private key.

Once the tier key is decrypted, it is passed to the enforcer. The enforcer is not allowed to disclose it to any other program or save it to disk. Violating enforcers will either be attested in the enforcer commitment or forbidden to run. In either case, the member node will refuse to deliver the tier key. Satem protects the key: if any part of policy enforcement software stack is compromised, it wipes out the key.

Due to the protection of the tier key, owning the key implies that the node has been verified for trusted policy enforcement and is still protected by Satem. This property helps simplify the trust verification process in the MERGE protocol. At steps 3-6, two nodes leverage a common old key to verify the trustworthiness without the need of Satem reports. Since computing the MAC code is much cheaper than the Satem report, the performance is greatly improved.

It is of little importance who generated the key. What really matters is that the key must be produced by a node that is trusted to enforce the policy. As we have discussed, the JOIN and MERGE protocols ensure that (1) a policy enforcing node will only accept a key from another policy enforcing node, and (2) a policy enforcing node will share the key only with other policy enforcing nodes.

On-demand merging. The MERGE protocol does not guarantee two trusted tiers to be merged completely in one run. In fact, merging is driven by the need of facilitating communication. As a result, it only aims to merge the nodes that interact with each other. In practice, a TTL can be used in MERGE messages to limit the scope of merging in the same way as trusted tier establishment. Nodes beyond the coverage of the TTL may later merge into the trusted tier when they need to interact with nodes in the trusted tier. In this way, merging is carried out on-demand step by step.

4.4 Prototype and Evaluation

We implemented the policy enforcing mechanism prototype under the Linux 2.6.12 kernel. It consists of the Satem based trusted agent and the tier manager. To evaluate the performance, we also implemented enforcers for two MANET applications: AODV (user-level daemon) [2] for ad hoc routing, and Mute [12] for P2P file sharing. Since the implementation of Satem was fully discussed in Section 2, in this section, we will focus on the policy enforcer and tier manager.

4.4.1 Policies and Enforcers

We defined the policy for Mute, P_F, and the policy for AODV, P_R, as illustrated in Fig. 4.6. These policies address the security issues for the routing and file sharing described in Section 4.1 respectively. In P_R, when node i receives an AODV reply message `RREP(o,d)` to a previous query originating at `o` for destination `d`, it must compute and check the geographical leash. The route is accepted if and only if the leash is valid. In the evaluation, the max distance $dmax$ and runtime distance between the two nodes $d(i, d)$ are constant and pre-defined. td and ti are the local time on node

AODV Policy (P_R) on Node i implements the secure routing policy for AODV tier, which requires that each node check the packet leash.

```
Incoming:
   R1.  RREP(o,d):
        if (dmax  > d(i,d) + 2*(ti-td)*v + e)
           ACCEPT
        else
           DROP
```

Mute Policy (P_F)is the unselfish sharing policy for Mute tier, which ensures that nodes in the file sharing network serve at least one file download request from other nodes after posting 3 requests.

```
Initial: Credit=3,
Incoming:
      R2.  Request(f);
           if(Own(f))
             Credit--;

Outgoing:
      R3.  if(Serve(f))
             Credit += 3;

      R4.  if(Request(f)){
             if(Credit > 0){
                Credit--;
                ACCEPT;
             } else
                DROP;
           }
```

Figure 4.6: The Example Policies of the File Sharing+AODV Two-tier Network

d and i respectively. In P_F, each node is given 3 credits in the beginning. The credits are deducted by 1 every time the node rejects a request for a file it owns or requests a file from other nodes, and added by 3 every times the node serves a request. The node must maintain positive credits to be able to request new files from others.

The enforcers can be any software that understands and enforces the policies. In general, users need to provide the right enforcers for the applications. For evaluation purpose, we implemented simplified enforcers for the above Mute and AODV policies by modifying the applications' source code and hard coding enforcement of the policies. Hence, the applications are also the enforcers of themselves and must be trusted. This is

done by including them in the enforcer commitments and protecting them using Satem.

4.4.2 Tier Manager

The tier manager is a service application. It implements the tier creation, JOIN and MERGE protocols. In addition, it provides the registration service for the user to register tier enforcers. During registration, the user provides the server port that the enforcer listens on. Satem maintains the mapping between the application port and its commitment. Hence, the tier manager can use the port number to retrieve the commitment of the enforcer during JOIN protocol. Moreover, when the tier manager receives the tier key at the end of JOIN or MERGE protocol, it delivers the key to the registered enforcer that has been verified.

4.4.3 Experimental Evaluation

To evaluate the performance of our mechanism, we ran prototype experiments. In the experimental evaluation, we created a 2-tier trusted ad hoc network over 3 laptops and measured the overhead incurred by our policy enforcing mechanism in application execution and communication.

Our method impacts system performance by adding latency to (1) the kernel call execution due to enforcing the Satem commitment, (2) joining the trusted network due to verifying trustworthiness during JOIN and MERGE protocols, and (3) the network communication due to enforcing the policy and computing and verifying the MAC for application messages.

Methodology

We created an 802.11g ad hoc network consisting of three laptops (IBM T43 with a 1.7Ghz Pentium M CPU, 512M RAM, and Atheros wireless card). In order to test multi-hop communication, the network was configured with a line-like logic topology (i.e., the direct link between laptop 1 and 3 was disabled, but they could still communicate with

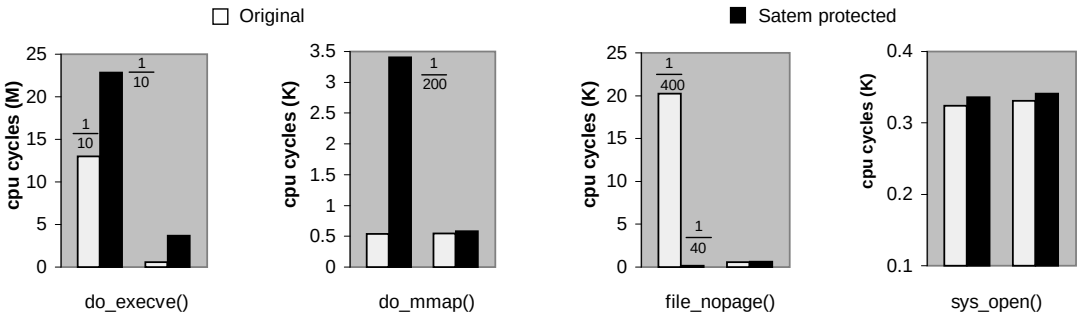

Figure 4.7: Overhead of Satem Commitments Enforcement in Kernel Calls

each other through laptop 2 as a router). This was achieved by enabling MAC filtering using `iptables` [19] on each laptop.

Each laptop ran two applications: AODV (user-level daemon) [2] for ad hoc routing and Mute [12] for P2P file sharing. We used P_R and P_F defined in Fig. 4.6 and created a two-tier trusted network consisting of a file sharing tier enforcing P_F and an underlying routing tier enforcing P_R. We used a simplified method to implement the Mute and AODV enforcers by directly modifying the applications' source code and adding the policy enforcement functionality. As a result, the two applications themselves are trusted and attested by the enforcer commitments.

Results

Kernel call cost. The overhead in kernel calls is incurred by the Satem trusted agent, which enforces the system and enforcer commitments. We measured it in terms of extra CPU cycles needed to complete these calls. Fig. 4.7 shows the overhead in four kernel calls: `do_execve`, `do_mmap`, `sys_open`, and `filemap_nopage`. We measured the cost of these functions in enforcement of P_F in Mute enforcer. All functions were measured in two cases: in the original Linux 2.6.12 kernel and in the Satem kernel. For each function, we measured its overhead for the first call (the left two columns in the figures) and the overhead for subsequent calls (the right two columns in the figures).

Scenario	Latency (in seconds)
Join	2.54
Merge	0.38

Table 4.1: Tier Joining and Merging Delay

The graphs [1] show that Satem impacts the first time call of the kernel functions differently. For `do_execve` and `do_mmap`, the by-page attestation for mapped code files dramatically increases the overhead, since Satem preloads every mapped page into the page cache. However, the preloaded pages benefit `filemap_nopage` because the pages are very likely to be in the cache when a page fault occurs. Furthermore, the lazy attestation mechanism significantly reduces the costs of the subsequent calls in the Satem kernel. Cost of the subsequent `do_execve` calls is still large compared with the original kernel, but the impact on the system is limited since it is only one-time cost.

Joining latency.

We measured the joining latency as the delay between the time a node starts joining or merging into a tier and the time it is accepted. As shown in Table 4.1, both JOIN and MERGE protocols incur significant latency. The high overhead is mitigated from two perspectives. First, JOIN is only used for nodes to join a new tier for the first time. Second, if a node loses wireless connectivity to others just for a short period of time, it is unlikely that the trust tier to which the node belongs will merge with other tiers. Therefore, once the node regains the wireless connectivity, it can simply reconnect to the tier without running either JOIN or MERGE protocols. With that said, even MERGE protocol is rarely needed.

The latency of JOIN protocol is largely due to the time the TPM takes to generate signatures, which varies dramatically by the TPM models and vendors. The TPM we used in the test were manufactured by National Semiconductors. TPMs with higher performance are already available [3]. JOIN latency between machines equipped with

[1] The cost of the functions of the first and subsequent invocations may be dramatically different. In order to compare them in one figure, we use a reduced scale to plot the following figures: first call to `do_execve` at 1/10, first call to `do_mmap` in the Satem kernel at 1/200, first call to `filemap_nopage` in original kernel at 1/400, and first call to `filemap_nopage` in the Satem kernel at 1/40.

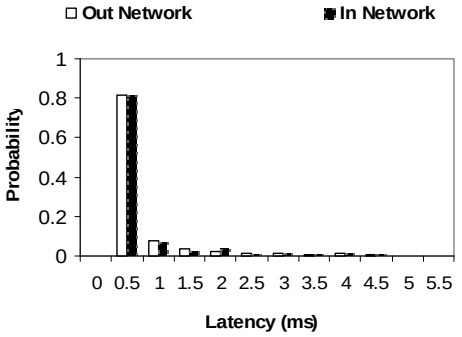

Figure 4.8: Probability Distribution of Ping Latency with and without Policy Enforcement for Routing (AODV)

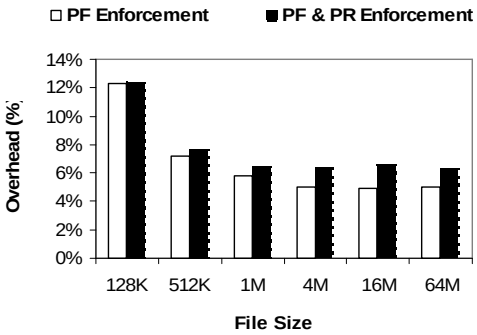

Figure 4.9: Policy Enforcement Overhead in Mute and Mute+AODV

these TPMs is expected to be much lower.

Network communication delay.

We measured the routing latency indirectly. We ran the AODV daemons on all three laptops and pinged laptop 3 from laptop 1. Then, we measured the round trip time (RTT), which includes both packet transmission latency and the routing cost. Since all ICMP packets are at the same size and enforcement of P_R has no impact on the ICMP protocol, the cost of transmitting each packet is the same. The routing cost varies with the change of the network state. In one extreme, the routing cost is high when laptop 1 has to build a full route from scratch since all 3 laptops have no routes to the others. The overhead of enforcing P_R is also high in this case because enforcement is performed on all nodes on the route. In the other extreme, the routing cost is zero once the route is established and remains valid; laptop 1 can keep sending packets via the route. Therefore, the enforcer is not invoked and the enforcement overhead also becomes zero. To test the different cases, we randomly invalidated the existing routes on each laptop by blocking and unblocking the wireless network interfaces. The enforcement overhead is determined by the probability distribution of these scenarios.

Fig. 4.8 compares the probability distributions of ping latency in the trusted network with P_R being enforced (called in-network) and in the original network with P_R not being enforced (called out-network). Clearly, the latency in the two cases are nearly identical, indicating little impact of enforcing P_R on the routing latency. In both scenarios, the majority (over 85%) of routing efforts incurred low delay (less than 1ms).

This was because our route invalidation was infrequent and most of the time, laptop 1 had a valid route to laptop 3 and the enforcer was not invoked. Hence, the mean overhead is low (less than 5%).

To measure the overhead of enforcing P_F, we compared the downloading speed of Mute application in three cases (1) ran Mute in the original network with no policy enforcement, (2) ran Mute in the 1-tier network with only $\mathcal{T}_F$ tier enforcing P_F, (3) ran Mute in the 2-tier network with $\mathcal{T}_F$ and $\mathcal{T}_R$ tiers enforcing P_F and P_R respectively. We measured the download latency in case (1) as the baseline and measured the percentage of increased delay in case (2) and (3). The results are summarized in Fig. 4.9.

Compared with no enforcement, the enforcement overhead in both cases decreases with the size of the file being transferred increasing. This is because the overall costs consist of the initial cost of enforcing the policy in handling file download requests, plus the ongoing cost of computing or verifying the MAC code for each Mute message being sent out or received. With the file size increasing, network transmission and routing costs become more significant, making the policy enforcement cost relatively small. However, the policy enforcement overhead does not vanish. Instead, it levels off at around 6%. The overhead is mainly due to the commitment enforcement by the trusted agent and the MAC generation and verification by the tier enforcers. The cost of enforcing the policy set $\{P_R, P_F\}$ demonstrates similar pattern with high level of overhead due to the extra cost of enforcing P_R. One important difference is that the cost of enforcing P_R increases with the length of the route between the two nodes, while the cost of enforcing P_F does not. This is due to the fact that P_R is enforced by all nodes on the route, while P_F is only enforced by the two endpoints. We will show in the simulation that this difference causes the overhead of routing to increase dramatically in complex dynamic networks.

4.4.4 Evaluation Through Simulations

To understand the performance overhead of network creation and policy enforcement in large MANETs, we also ran simulations in various mobility scenarios using NS-2 [20].

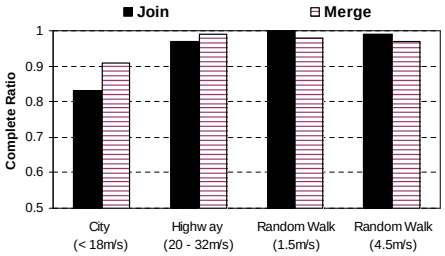

Figure 4.10: JOIN and MERGE completion ratio in Trusted Multi-tier Network

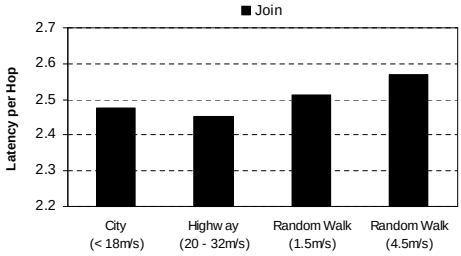

Figure 4.11: JOIN Latency per Hop in Trusted Multi-tier Network

Methodology

The simulation includes three types of mobility models: highway vehicular network, city vehicular network and a network with nodes moving randomly at walking speeds. We leveraged our vehicular simulation tool [87] to generate the highway and city networks. The highway scenarios simulated is a 10 mile segment of New Jersey Turnpike with 200 nodes (cars) moving at a speed from 45 miles per hour (20 m/s) to 72 miles per hour (32 m/s). The city scenarios is a 1.2x1 square miles region of Los Angeles with 100 nodes moving up to 40 miles per hour (18 m/s). We also modified Carnegie Mellon University *setdest* [38] utility to generate random waypoint mobility models with 100 nodes in a 1x1 square miles region, moving at walking speeds. In our simulations, we set the node density to be around 6-8 neighbors per node to avoid connectivity failure due to sparse networks or too much contention in over-crowded networks. In all scenarios, we ran the simulations for 300 seconds.

Since the cost of cryptographic operations associated with JOIN, MERGE, and enforcement cannot be ignored but NS-2 does not account for execution time, we added latencies for these operations. Specifically, these additional latencies were modeled as normal random variables with standard deviation equal to 10% of its mean. We set the mean latencies as follows: 1150ms for JOIN, 180ms for MERGE, and 0.15ms for enforcement. These numbers were obtained from the previous experimental results.

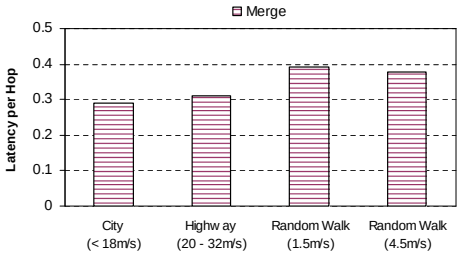

Figure 4.12: MERGE Latency per Hop in Trusted Multi-tier Network

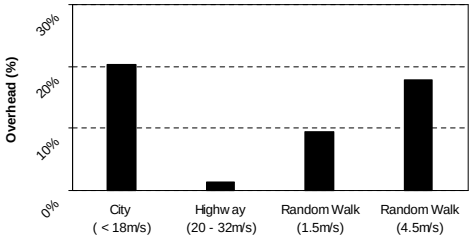

Figure 4.13: AODV Policy Enforcement Overhead in Ping RTT

Results

Cost of network creation. We measured the cost of network creation in terms of both successful ratio of JOIN and MERGE operations and the latency it takes for a node to join the network. The completion ratio of JOIN (or MERGE) is defined as the total number of nodes that successfully join (or merge into) the network against the number of nodes that apply to join (or merge into) the network. To test MERGE, we first set all nodes in the network to be the members of the old tier. We randomly selected one node and updated its membership to become the first node of the new tier. Then, this node automatically started the MERGE process with other nodes.

As illustrated in Fig. 4.10, in most cases we achieved a completion of over 80% for both JOIN and MERGE. The ratio is lowest for the city scenario. This is mainly because nodes exit the region when they reach the boundary of the map. This problem does not exist in the random walking scenarios and is less of a problem in the highway scenario since most cars stay in lanes without exiting the highway.

The latency for a node to join the network is defined as follows:

$$joining\ latency = \frac{t_i - t_0}{dist_i} \tag{4.1}$$

where we denote t_i as the time node i joins the network, t_0 as the time the originator initiates the network, and $dist_i$ as the number of steps the join invitation message has traversed before reaching node i. In another words, we measured the latency per hop. The reason to do so is because obviously the more number of hops a node is away from the network originator, the longer it takes for it to join the tier. We did not count

nodes that failed to join the network since the latency for these nodes was infinite.

Fig. 4.11 and 4.12 show that both vehicular networks incurred less latency in joining and merging into the trusted network. This result can be explained by the broadcast storm problem in tier creation. In the random waypoint scenarios, the number of broadcast messages increases exponentially with the network expanding neighborhood by neighborhood, which leads to many packet losses due to contention. By contrast, in the vehicular network scenarios, messages have to propagate along the roads unless the sending cars are at the road intersections. This means that most of the time the number of broadcast messages does not grow as they propagate through the networks. Therefore, latency is low in this case.

Enforcement overhead in AODV. We measured the enforcement overhead in AODV in the same way as we did in the previous subsection. We randomly selected the source and destination nodes and let the sources repetitively ping the destinations. We measured the per-hop RTT by dividing the round-trip time by the number of hops traversed. We denote BRTT as the basic per-hop RTT measured when P_R was not enforced and ERTT when P_R was enforced. We computed the overhead as

$$overhead = \frac{ERTT - BRTT}{BRTT} \times 100\% \tag{4.2}$$

As Fig. 4.13 reveals, the overhead is higher than in the simple prototype experiments (Fig. 4.8), but they still remain under 20% in all cases. The main reason for the overall increase is that the network is highly dynamic and the established routes do not last long. Frequent broken routes trigger route repairs, which involve policy enforcement. In the prototype experiments, the route re-establishment was far less frequent.

The worst case is the city scenario because it is the most dynamic network. The overhead is small in the highway scenario. This is because the relative positions between most nodes do not change compared to other networks though each node itself moves at highest speed.

4.5 Limitations

In the current prototype, we implemented the enforcer by hard coding the policy enforcing function in the application source code. This is inflexible since changing the policy may require modifying the application. A better solution is to implement a standalone enforcer as the transparent application proxy. In this way, the application request is redirected to its local enforcer, which communicates with the application on the remote node. One way to achieve this is to establish the mapping between the application and its enforcer when the enforcer registers with the tier manager. To do so, the user provides the tier manager with the TCP or UDP port number on which the application S listens, p_S, and the port number on which the enforcer listens, p_E. The manager then maintains the mapping between the application port, p_S, and the enforcer port, p_E, in the kernel by using Linux built-in kernel hooks `NF_IP_LOCAL_OUT` and `NF_IP_PRE_ROUTING` [19] as follows:

1. `NF_IP_LOCAL_OUT`

   When the local node n_l sends a message to S on a remote node n_r, the kernel maps destination port $n_r : p_S$ to $n_l : p_E$. This causes the message to be redirected to the local enforcer. E computes and attaches the MAC code for the application message.

2. `NF_IP_PRE_ROUTING`

   When the local node n_l receives a message for S from a remote node n_r, the kernel maps destination port p_S to p_E, which causes the message to be redirected to the local enforcer. The local enforcer first verifies that the attached MAC code is correct. Otherwise, it drops the message. Next, it strips off the MAC code and forwards the message to the application.

4.6 Summary

This chapter presented a mechanism for MANETs to enforce application communication policies. Under this mechanism, nodes supporting the same set of applications and

enforcing the same policies construct a *trusted multi-tier application centric network*. Each tier of the network runs one application and enforces its associated policy. The application of the upper tier depends on the applications of the lower tiers to communicate. Only trusted nodes are allowed to join the network. Moreover, communication between them is regulated by the policies at every tier. To ensure trusted policy enforcement, we augment each node with a trusted kernel agent based on the TCG TPM. We evaluated the method through a prototype based on an IEEE 802.11 ad hoc network and through network simulations. The results demonstrate the feasibility of the proposed method as well as its low overhead.

In the past three chapters, we have presented the methods to ensure trusted applications and trusted communication between them for MANETs. Orthogonal to them is the problem of ensuring trusted identity. In the next chapter, we will present a locality driven key management architecture to solve this problem.

Chapter 5

Locality Driven Key Management Architecture

In this chapter, we present a locality driven key management architecture that achieves robust key authentication. We start with the problem of authenticating public keys in MANETs in Section 5.1. Then, we present the architecture in Section 5.2 and discuss the protocols to manage trust in Section 5.3. Next, we evaluate the correctness and costs of the our solution in Section 5.4. Finally, we discuss the limitations in Section 5.5 and summarize the chapter in Section 5.6.

5.1 The Problem Statement

In distributed systems, the dominant strong method of authenticating a node is to authenticate the node's public key, which requires ubiquitous capability of verifying the binding between a public key and the owner principal. In the Internet, the mainstream solution is to have a third-party centrally trusted entity, called Certificate Authority (CA), vouch for the authenticity of the binding by issuing digital certificates, which in essence is a statement of the binding digitally signed by the CA. In practice, CAs and digital certificates are organized and maintained by Public Key Infrastructure (PKI) [33, 7].

It is still questionable if PKI can be implemented in MANET because PKI requires well-protected CAs and constant connectivity between users and CAs. However, in MANETs, all nodes are exposed to hacking to the same extent and no one can be assumed to be significantly more secure than the others. Moreover, devices may roam around, run out of power or just stop functioning, which lead to volatile connectivity among them and CAs.

Research proposals have been seen in [124, 123, 70] etc to address the two issues by

distributing the CA's functionality across a set of network nodes and by using threshold signature [105] to achieve tolerance up to the threshold number of faulty nodes. These methods are only suitable for small MANET with a single CA, partially due to the inherent high communication cost. A more fundamental reason is that it is difficult if not infeasible for a CA to get familiar with all the other principals in a large MANET, whereas the trustworthiness of certificates a CA signs mainly depends on how much it knows about the principals.

Another approach is based on the concept of "web of trust", first appearing in PGP [126]. In these methods, each principal is its own CA [40, 29] and keeps a certificate directory. To authenticate a certificate signed by another principal, a principal has to find a certificate path between them. Although these methods avoid the problem of maintaining a key infrastructure at high cost, they are faced with difficulty of finding such a path without incurring a lot of broadcasting costs or forcing each principal to save a large number of certificates in its local directory.

We believe that the solution to the key management issue in MANET demands an application view of MANET. Opposite to the traditional routing-driven view of MANET as a monolithic network, we envision a MANET from the application angle as a group of interacting networks, each of which is formed to fulfill a specific task. Different MANETs may communicate with each other to get help for another task. The application view discloses the locality of MANET. We argue that locality helps build key management because each MANET is composed of principals for the same task, which have close interaction with each other and are likely to be in the same neighborhood. The concept of locality has already been exploited explicitly or implicitly to help build security. For instance, in [31] two network nodes set up a secure channel via location-limited channels. [70] also enhances the trustworthiness of a certificate using locality in that a certificate is generated by one-hop neighbors in most cases.

We developed a new key management architecture driven by the application centric nature of MANET and the concept of locality of trust. In this architecture, certificate authorities are established only within a neighborhood using threshold cryptography. Different certificate authorities maintain trust relationships, called trust chains

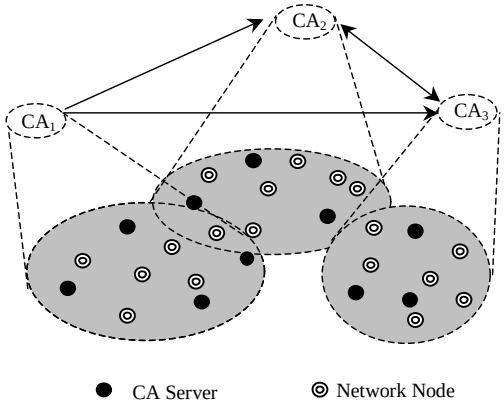

Figure 5.1: Conceptual Key Management Architecture

for cross-CA authentication. The architecture has a number of benefits. First, locality makes certificates more trustworthy, in that in a local community a CA has better chance to interact with other principals. Moreover, it reduces the communication overhead between principals with their CA because of shorter local distance of message delivery. Secondly, the architecture inherits fault-tolerance and high availability from threshold cryptography. Thirdly, the trust chain enables the architecture to authenticate foreign certificates in a low-cost and timely fashion without forcing each individual principal to keep a huge certificate directory like in other PGP-like approaches [126]. Compared to other systems, like [40], our solution provides an answer with full certainty (not just a probability one) but at lower communication cost by eliminating the need of broadcasting trust changes to irrelevant principals.

5.2 Key Management Architecture

Our key management architecture is composed of a group of certificate authorities (CAs), each of which provides public key authentication service to its own community. Among them, there may exist trust relationships. CAs rely on the trust relationships with others to authenticate "foreign" certificates issued by other CAs. The architecture is illustrated in Figure 5.1. We will explain in details how CAs and the trust relationships among them are established and maintained at conceptual level in this section

and discuss the protocols for trust management in details in the next section.

5.2.1 Certificate Authority

Adjacent nodes jointly establish and maintain a certificate authority for key authentication. We say that two nodes are adjacent if they have short routes to each other. We call these nodes CA servers to differentiate them from the other nodes in the neighborhood. The CA is constructed using standard threshold cryptography. To be self-contained, we include the high level description of the threshold mechanism. More information about threshold digital signature can be found in [105, 73].

1. *CA Bootstrap.* We assume each node has the capability of generating public/private key pairs, producing and verifying digital signatures. We further assume there exists a special node d in each neighborhood that is trusted by all nodes. In threshold cryptography, d is also called *dealer* and its own public key PK_d is well-known to all nodes. The CA is initialized by the dealer d generating a CA key public private key pair denoted as (PK, SK) and polling the community for N nodes who like to serve as the servers constituting the CA. The criteria that d picks up the N nodes are system dependent and are beyond the scope of our discussion, e.g., d may select servers with most computing resources or better network connectivities etc.

   When N servers are chosen, the d sets the threshold (denoted as K) and generates the partial shares of SK: $SK_1, SK_2...SK_N$ for each server. All these partial shares are signed with d's private key SK_d. That is, any set of K servers can generate a valid CA signature by combining partial signature from each server but any fewer than K servers cannot. We require $K > \frac{S}{2}$ such that honest servers form a majority. Due to the locality of CA, the CA servers should have good connectivity to each other. We assume that at any time at least K honest servers are available and for any message delivered at least K honest servers will receive it. Then d distributes the partial shares to the other servers, destroys the CA private key SK and broadcasts the composition of the CA to the community.

2. *CA Certificate Generation.* The CA provides both certificate generation and verification services to the nodes in the community. Any node x in the community can request a certificate from the CA, which put in the simplest way, is the public key of the node denoted as PK_x signed with SK of the CA. The procedure starts with x submitting its PK_x to one of the N servers, for instance, S_p. Since S_p acts as the proxy of x, we call it *proxy server* in this work. Then, S_p calls other servers to sign PK_x using their partial keys. Each server independently decides if PK_x should be trusted as the public key of x. If not, it signs a predefined denial message instead of PK_x. The partial signatures are gathered by S_p for full signature generation. As long as K servers concur to trust the PK_x, the valid certificate is generated. On the other hand, the certificate can be directly verified using CA's public key PK, since the signature is literally the same as that signed with SK.

   Our approach does not impose any restriction on how a CA server evaluates the trustworthiness of a principal's public key. Instead, the trustworthiness of a public key is fully determined by policies and rules, which can vary from CA to CA or even server to server within the same CA. CA servers are free to choose any policies at their own discretion. For instance, a CA server may be convinced of the authenticity of a node's public key just because the network address in the certificate request matches the source address of the request sender. A more prudent server may do more by checking if there are any conflicting requests for the same address. However, whatever policy CA servers choose, our architecture guarantees that no single or even up to $K - 1$ malicious servers can subvert the authentication.

3. *CA Certificate Revocation.* Basically, the revocation process is similar to standard PKI with an exception that signature is generated in a threshold manner. The CA keeps a copy of all the revocations it has signed in its certificate revocation list (CRL). When a node y later wants to verify the validity of a certificate, it may send the request to the CA. Similar to processing certificate generation request,

each CA server checks its local CRL. If y is found in its CRL, it will partially sign a predefined denial message. Finally, combining the partial signatures the CA can generate an signed reply in line with the decisions of at least K servers. When the CA revokes a certificate, it may optionally notify new revocations to the community. Since the community is local and the revocation is relatively rare, the communication cost is not prohibitive.

5.2.2 Trust Chain

So far, CAs manage key authentication for nodes in their local neighborhoods. When two nodes from different neighborhoods need to authenticate each other, they need a trust chain. In this section, we discuss the details of how a trust chain is established and maintained and how certificates can be authenticated using it.

Trust Chain Definition

In the context of key management, we have a narrow definition of trust between two principals: a principal A trusts another principal B if and only if (1) A can authenticate B and (2) A believes in the authenticity of any valid certificates signed by B. In addition to trust relation, a principal may *distrust* or be *unfamiliar* with another principal. We say that A distrusts B if only if A can authenticate B BUT does not believe in the authenticity of any valid certificate signed by B. Unfamiliar covers all the rest, which means A has no idea about B, neither its identity nor its behavior. We assume that trust is transitive while both distrust and unfamiliar are not. If A trusts B on its own, i.e. based on direct interactions with B, we call this direct trust, denoted as $\rightarrow$ to differentiate it from the trust gained by transitivity. Opposite to direct trust is indirect or recommendation trust. We say A indirectly trusts B if there exists a trust chain from principal P_0 to P_n, denoted as $P_0 \Rightarrow P_n$, defined recursively as follows:

(1) $P_{n-1} \rightarrow P_n$ and

(2) $P_0 \Rightarrow P_{n-1}$

Neither $\rightarrow$ nor $\Rightarrow$ is symmetric, i.e $A \rightarrow B$ does not imply $B \rightarrow A$. Formally, the trust chain can be represented as a directed graph $G = (V, E)$ where V is the set of vertices representing all CAs and E the set of edges representing the relations between two CAs. Based on the type of the relation, we call it trust, distrust or unfamiliar edge. Let C_1 and C_2 be two CAs, and V_1 and V_2 be the vertices representing them in the graph. The trust chain from C_1 to C_2 can be represented as a directed path from V_1 to V_2 that is composed of only trust edges. If we take a CA as a root, all the chains it has compose a trust tree.

Dynamic Trust Chain Maintenance

When the system is bootstrapped, only direct trust exists. A CA need to build trust chains by exchanging trust information with other CAs that it directly trusts. Furthermore, when changes of trust take place somewhere, a CA must be able to update its chains to reflect the new trust relationships on-the-fly. We define a protocol enabling each CA to build the trust tree locally. The idea is inspired by routing protocols such as OSPF [14] in which each router only keeps local information but computes global routing table by exchanging information only with its neighbors. Similarly, each CA keeps a record of its directly trusted CAs, and establishes indirect trust with other CAs by exchanging records with them. The details are below:

Data Structures Each vertex V_i maintains two tables: I_i and relation table R_i where

$I_i = \{V_j | V_j \rightarrow V_i\}$ and
$R_i = \{(V_j, V_g, c) | V_j, V_g \in V \text{ and } c \text{ is an integer}\}$

In I_i table, the CA C_i keeps track of the truster CAs that trust itself. When C_i becomes trusted by another CA, i.e. receiving a certificate from the CA, it updates its I_i table to include the certificate issuer. The table R_i describes the trust relations that C_i has with other CAs. Conceptually,V_g serves the role similar to gateways in routing:

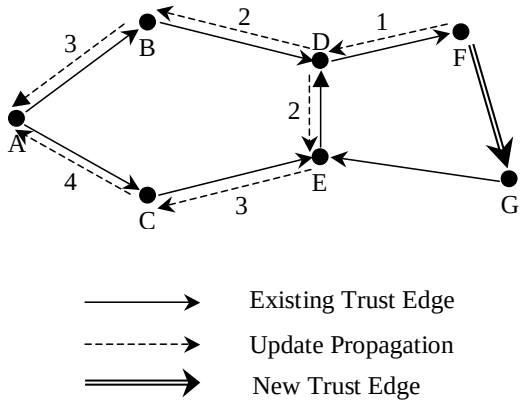

Figure 5.2: Trust Chain Update between Certificate Authorities

it means that the reason the relation between V_i and V_j exists is due to the fact that V_g has the relation with V_j, i.e., $V_g \Rightarrow V_j$, and $V_i \rightarrow V_g$. c is the weight of the relation and $-MAX \leq c \leq MAX$, where MAX is a positive integer. By assigning c to different values, R table can capture all kinds of relations the CA has with others, for instance, positive for trust, negative for distrust and 0 for unfamiliar. The bigger $|c|$, the more trustworthy the relation.

Trust Propagation. Whenever there is a change of inter-CA trust relations including both establishment of new direct trust and revocation of existing trust, all related CAs need to be notified.

1. *Relation Establishment and Revocation*

   When a CA server C_i (represented by V_i in the graph) decides to trust C_j, V_i assigns an initial weight $c(c > 0)$ to the relation and adds the entry (V_j, V_i, c) to R_i. Then, it sends an notification message [ESTABLISH:(V_j, V_i, c)] to V_j and waits for replies. When V_j receives the message, it replies with its own current trust table R_j signed with the CA key. When V_i receives the reply, it checks its R_i to find out if it needs to be updated because of R_j. The R_i table will be updated if and only if there exist l,m and n such that for (V_l, V_m, c_i) in R_i and (V_l, V_n, c_j) in R_j, and the following inequality holds:

$|c_i| < (|c_j| - w)$, where $w(w \geq 0)$ is the loss of trust transitivity.

Here we assume that the trustworthiness of a relation decreases as it becomes more and more indirect. When $c_i = 0$, the inequality represents changes of relation from unfamiliar to familiar(trust or distrust). When $c_i \neq 0$, the inequality indicates that the trustworthiness of the existing relation is enhanced. In both cases, (V_l, V_m, c_i) will be replaced with $(V_l, V_j, -(|c_j| - w))$ if $c_j < 0$ or $(V_l, V_j, c_j - w)$ otherwise, which results in a new table R_i'. Then, the CA sends an update message [UPDATE:R_i'] to each $V_i \in I_i$. Similarly, when C_i decides to distrust C_j, it adds an entry (V_j, V_i, c) to R_i and directly sends an update to each $V_i \in I_i$. Sending notification to V_j is an option but not required, since distrusting V_j implies no expectation on V_j to follow the protocol any more.

2. *Relation Update*

   The scenario is triggered by the message [UPDATE:R_l] from V_l. Upon receiving the message, a CA C_k will first check if this message is an out-dated one. This is done via assigning version number to table R and we will explain in details in next section. If this is a new update message, C_k repeats the same procedure as the above to update its own R_k. Finally, if a change is made and a new table R_k' is generated, C_k passes [UPDATE:R_k'] to each V_k in its I_k set.

An example is shown in Figure 5.2 depicting how a trust chain is maintained by updating each CA's local trust table in a lockstep manner. In this example, A, B, C, D, E, F and G are CAs. When F begins to trust G, it triggers chain update along the reverse path of the trust chain step by step. For instance, D gets the update message first and then propagates to both B and E, which further notify A and C.

A CA verifies if the message comes from a directly trusted CA before taking further actions at each step by running public key authentication. The only exception is when a CA receives ESTABLISH message. This is because the receiver does not need to trust the sender (recall that trust is asymmetric) or may not have the sender's public key.

Cross CA Authentication

Trust chains enable cross CA authentication. When x wants to verify if PK_y belongs to another node y, it verifies the validity of y's certificate. If y is signed by a CA x trusts, e.g. CA_i, the key authentication is successful. However, if y just roams into x's neighborhood from another neighborhood and only has one certificate signed by CA_j, x may still be able to authenticate y if there is a trust chain from CA_i to CA_j. When CA_i finds an entry like $(j, l, w)w > 0$ in its R_i, it can either reply with a message of "successful authentication" or continue sending verification request to C_l, which recursively repeats the same procedure until a response signed by C_j is returned. The former case trades off assurance for low cost, in that the result is based on previous records and does not take into account most recent revocation. By contrast, in the latter case, the result is directly based on the certificate issuer's latest decision but incurs more communication and computing costs. Another way to enhance assurance is to let each CA also notify all CAs in its I table when a CA revokes a certificate. Each CA receiving the revocation simply saves it to its local CRL and continues to forward the message in the same way. As a result, each CA has the CRL of all other CAs it trusts and can answer the verification request on behalf of them.

On the other hand, if y comes into a new community and wants to authenticate node z, it may choose to either phone home or trust the new local CA to provide certificate services. The former option stops working when y loses network connectivity to its original community, which is not unusual in MANET. The latter solution does not have this problem but requires new trust to be established, which entirely relies on y's own judgment.

5.3 Protocols for Trust Management

It is straightforward to implement the trust chain protocol defined in Section 5.2 when each vertex is a single network node. However, in our trust chain a vertex represents a local CA, which actually consists of a group of CA servers. Communication protocols are needed to ensure that a group of servers can behave like a single server. The typical

solution to this problem in fixed wired networks is to pick up a server as a master and let it represent the team. However, this approach will not work in our system because no one can be trusted absolutely. Furthermore, the improvement of security in CA lies in its threshold scheme, in which no single server (or more generally, less than quorum servers) can compromise the whole system. But having a master server introduces a single point of failure. To solve this problem, we define two protocols: CA Table Update protocol (CTU) and CA Head Election protocol (CHE), both of which work in an autonomous fashion. That is, each of the servers keeps a copy of all the data, including R and I tables, and makes trust decisions independently based on its local data. Combining these decisions together, the CA servers can make a uniform decision, which bears the CA signature.

In the rest of the section, we will explain in details how the two protocols work. Assume we have a CA $C_i = \{C_{i1}, C_{i2}, ...C_{iN}\}$ with N servers, K as the threshold, and the local copies of R_i, and I_i of each CA server $C_{ik}(1 \leq k \leq N)$ are denoted as R_{ik}, and I_{ik}, respectively. Furthermore, a server C_{ik} will not update its local copy of R_{ik} unless it receives a new table $R_i{}'$ signed by C_i. This guarantees that all updates to R_i table are certified by at least K CA servers.

5.3.1 CA Table Update Protocol

Based on the conceptual protocol above, a CA updates its table when it establishes a new trust relation with another CA or receives an UPDATE message from a CA it directly trusts. We will discuss them one by one. To deal with replay attack or outdated data, we let each C_{ik} of C_i keep a version number of its local copy of R_{ik} denoted as v_{ik}.

1. *Establish a new trust relation*

   (a) *When the proxy CA server, denoted as C_{ip}, generates a new signed trust decision, it makes a copy of its R_{ip} and works on the copy to generate a new table R'_{ip} according to the new trust. Then, it multicasts to all other CA*

servers a table update request [UPDATE:R'_{ip}] with version $v'_{ip} = v_{ip} + 1$, and waits for replies.

(b) *Upon receiving the new trust decision, a server C_{ik} verifies if it is signed with the CA key and should be trusted. Then, it does the same as C_{ip} to create a new table R'_{ik} and compares R'_{ik} including the version with R'_{ip}. If they are matched, C_{ik} signs the new table with its partial key and returns it to C_{ip}.*

(c) *When C_{ip} has at least K correct and unanimous responses, R'_i with the version $= v'_i$, it generates the full signature for it. Then, it overwrites its own copy of the table R_{ip} with R'_i and multicasts the signed new table to other CA servers. If its I_{ip} is not empty, it also sends the newly signed table R'_i to all CAs in I_{ip}.*

(d) *When C_{ik} receives the new table R'_i, it verifies the signature and overwrites its own R_{ik} with the new one.*

2. *Respond to UPDATE message*

The protocol works in the same way as the previous one except the first two steps:

(a) *When a proxy C_{ip} receives an update message [UPDATE:R_l] from another directly trusted CA l, it multicasts the message to other CA servers followed by a table update request [UPDATE:R'_{ip}] as well as [UPDATE:R_l].*

(b) *When a server C_{ik} receives the update message, it verifies if the CA, C_i, trusts the CA, C_l, directly. If C_i trusts C_l, C_{ik} checks R_l against its local R_{ik} in the same way as described in the conceptual protocol above. If C_{ik} needs to update its R_{ik}, it generates a new table R'_{ik} and compares it with R'_{ip}. C_{ik} signs R'_{ip} if R'_{ik} with R'_{ip} are matched in the same way as C_{ip}.*

In both cases, the message signed with the CA key also serves as a confirmation: a server will not generate more partial signatures if it is waiting for another confirmation. Since in MANET there is no guarantee of message delivery, the server will time out if no confirmation is received. The protocol works even if there are faulty or compromised servers as long as they are less than $N - K$. To get a valid signature, a proxy server, no matter it is honest or dishonest, needs to collect K correctly partially signed and unanimous R. Since we assume that at least there are $K > \frac{N}{2}$ honest servers that cache the correct R table locally, the protocol guarantees that new tables are generated correctly. On the other hand, because there are at most $(N - K) < \frac{N}{2}$ servers that can have incorrect R or be compromised, it is impossible for malicious users to generate a wrong signed table R.

5.3.2 CA Head Election Protocol

When dealing with multiple update requests at the same time, the CA Table Update (CTU) protocol may generate a different table R_i with the same version even though the protocol requests each server to generate one partial signature at a time. This happens only when compromised CA servers cooperate to violate the protocol. For instance, if C_{ip1} receives [UPDATE:R_l] and C_{ip2} receives [UPDATE:R_m] at the same time and both send the table update request to other CA servers, there is no way to control the sequence these requests are received. It is possible that $\frac{K}{2}$ honest servers get [UPDATE:R_l] and the other half get [UPDATE:R_m]. In general, the two different UPDATEs result in different R_i' but if all servers only sign one table at a time, the two different tables will be signed in sequence and thus, with different version number. However, compromised servers can simply sign both requests such that both C_{ip1} and C_{ip2} may have enough partial signatures to construct their R_i' with the same version number. This will cause confusion to other CA's when they need to update their trust table since the version number is used to guarantee that only the latest table from the trusted CA's will be considered.

To solve the above problem, we have to restrict that only one request is allowed to be sent at a time. To simplify the protocol, we require that only one server in one CA,

called Head, is permitted to send request. To avoid trusting any single server naively, the server is elected by the group and rotated periodically and/or on demand. There are no means of preventing a Head from violating the protocol. However, this can be easily detected by honest servers, which receive multiple requests at the same time. The protocol is simple and works as follows:

1. *Each server is assigned an ID known to all other servers. Everyone in the CA keeps track of the current head ID, called HeadID and uses the same algorithm, called GetNextHead to get the ID of the new Head. It is easy to find such an algorithm, i.e, if all ID's are within [0..N − 1], (HeadID + 1) mod N works. When some server C_{ip} calls an election, it multicasts the request to all other servers.*

2. *When a server C_{ik} receives the request, it calls GetNextHead to generate the ID and compares it with the request. If the generated ID matches the request, the server C_{ik} signs the request with its partial key and sends it back to C_{ip}.*

3. *Similar to table update, C_{ip} generates the fully signed notice for the new head, multicasts it to all other CA servers, which will update their head accordingly.*

In this protocol, there is no need to restrict a CA to only generate one partial signature for the new head at a time. The reason is that even if multiple requests arrive at the same time, a CA server will only sign the one that matches the result given by *GetNextHead*.

5.3.3 Message Delivery Fault Tolerance

Messages can get lost at network layer (routing failure) and application layer (a CA server dropping requests). The former problem has been extensively studied in MANET routing algorithms [91, 90] and is beyond the scope of this work. We simply assume that for each message sent, at least K honest servers in a CA will receive it. We tackle

the latter case. As we know, a single compromised server can not forge a valid signature and the only harm it may do is Denial-of-Service(DoS) attack by dropping it silently. However, this is detectable if it takes place inside a CA, because for each request, a reply is expected by the requester and dropping a request will finally result in discovery of the deviation from the protocol. The only threat is to inter-CA messages including ESTABLISH and UPDATE. In this case, both sender and receiver can be compromised.

There are two options to address the lost of ESTABLISH and UPDATE messages. First, since lost of these messages only results in other CA's having out-of-date trust relations, we let each CA run a mandatory periodic update notifications to everyone in its I table by picking up both sender and the receiver randomly. In this solution, the risk is bounded and finally an update notification can be received by an honest server in other CA's. Second, since out of $N-K+1$ servers there is at least one honest server, we can guarantee successful message delivery if there are $N-K+1$ senders and $N-K+1$ receivers. To do this, each CA picks up $N-K+1$ servers as senders. Every time when a CA sends message to another, each server in the sender CA determines if it is one of the $N-K+1$ servers. If it is one of these servers, the CA server randomly selects $N-K+1$ servers of the receiver CA and sends the message to them. To make it more efficient, each server is assigned a priority and sends the message after a different delay. If the server receives a confirmation before the timer expires, it simply cancels the timer. For instance, when sending an ESTABLISH message to another CA, a server sets up the timer. When the server receives an UPDATE message from the CA, it cancels the timer. We take both approaches in our solution without implementing the timer.

5.4 Evaluation

We evaluated our locality driven key management architecture in a hybrid method of both prototype implementation and simulation. First we implemented a prototype based on openssl [16] libcrypto library on Linux platform in C to evaluate the real computation cost of the threshold scheme. Second, we used NS2 [20] to simulate the protocols to evaluate its effectiveness and communication overhead in various MANET scenarios.

N	K	Proxy generating full signature (ms)	CA server generating partial signature (ms)
3	2	13.95	25.81
5	3	13.96	28.25
5	4	15.89	13.85

Table 5.1: Costs of Certificate Generation on Dell Latitude CPi Laptop(Pentium II 366M Hz, 256M SDRAM)

N	K	Proxy generating full signature (ms)	CA server generating partial signature (ms)	Total delay (ms)
3	2	396.70	333.98	717.87
5	3	421.85	203.29	959.88
5	4	427.60	212.93	996.25

Table 5.2: Costs of Certificate Generation on HP/Compaq iPAQ H3700 (Intel StrongARM 206M Hz, 64M SDRAM)

5.4.1 Prototype Implementation

The prototype implemented the basic signature generation function of CAs. It consists of 3000 lines of C code and is compiled on Linux 2.4 for both Pentium (Dell Latitude CPi Laptop with Pentium II 366 MHz processor and 256M SDRAM) and ARM(HP/Compaq iPAQ H3700 series with 206 MHz Intel StrongARM SA-1110 32-bit RISC processor and 64M SDRAM) architectures. Computation costs in terms of CPU time in different scenarios were evaluated on both architectures. Communication delay was calculated in the 802.11b network composed of several iPAQs. The results are shown in Table 5.1 and 5.2. The calculation was based on RSA 1024 bit public/private key.

From table 5.1 and 5.2, the overall cost is higher on the low-end PDAs than on more powerful and resourceful full-fledged computers. The computation cost from the CA server's perspective is independent of how the CA is composed because the computation load remains the same. Furthermore, the impact of increasing K on the proxy is nearly indiscernible. As observed by J. Kong et al. in [70], this is because the overhead of combining more partial signatures can be ignored when compared to the cost of key

generation. However, the total delay becomes bigger with the number of servers in the CA and the threshold increasing. This is actually due to the increased network communication costs and the waiting time to collect all partial signatures.

5.4.2 Simulation

We measured communication cost and tested effectiveness of our architecture in different MANET scenarios through NS-2 simulation. It consists of 1500 lines of C++ code and a few hundred lines of TCL code.

Measurements

We evaluated the locality driven key management architecture from effectiveness and communication cost aspects. A certificate service is effective if it can react to changes of trust in the system in time. In our inter-CA trust chain, a change of trust will be propagated along each trust chain and CA's on the chain will update their trust tables according to the new change. We define the *vulnerable window* of a CA (denoted by V) as the time between an event of trust change happens and the time when the CA finishes updating its R table for the event. In a vulnerable window, a CA makes decision on trusting or distrusting other CAs based on its previous records, rather than the most recent events. The smaller V, the more effective the system is. It is especially significant for a CA to deal with certificate revocation, because small V ensures low false positive rate resulting from granting trust to the CAs that should have been distrusted if the trust updates had been received.

Another important perspective of the system performance is communication cost. The threshold cryptography enhances the security at the cost of the messages since generation of a signature gets at least the threshold K number of nodes involved. This is even aggravated by the fact that no single node can be absolutely trusted. Therefore, to ensure correct message delivery, redundancy is brought in. We calculated the communication cost by the number of messages triggered by a single event. However, this number varies based on different logic topology of the trust relationships. To avoid being further complicated by these runtime variants, we only calculated the message

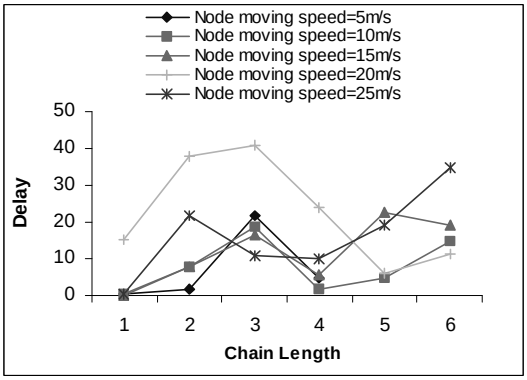

Figure 5.3: Certificate Authority Vulnerable Window(V) vs Node Speed

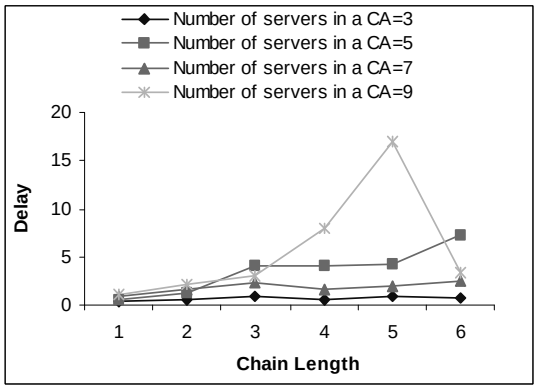

Figure 5.4: Certificate Authority Vulnerable Window(V) vs Number of Servers(N) of CA

cost for a single trust chain, denoted by M. The message cost of multiple chains is upper-bounded by (number of the chains) $\times M$.

Scenarios, Parameters and Results

We modified Carnegie Mellon University *setdest* [38] utility to generate random waypoint mobility models with different node moving speed. The speed varied from 5, 10, 15, 20, 25m/s etc. The simulations were run with 150-300 nodes spread in areas ranging from 300x300, 500x500 and 800x800 m^2. The results are shown in Figure 5.3, 5.4, 5.5, 5.6 and 5.7. In the current simulation implementation, we simply used multiple unicasts to simulate a multicast. Therefore, from the aspect of underlying communication cost,

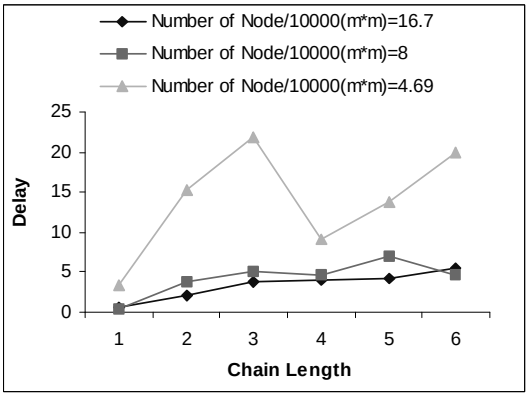

Figure 5.5: Certificate Authority Vulnerable Window(V) vs Network Density

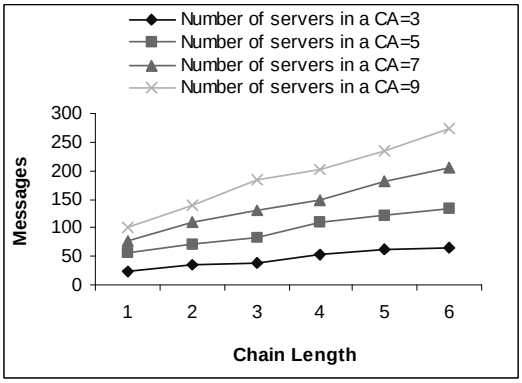

Figure 5.6: Message Cost(M) of Trust Chain Management vs Number of Servers(N) of CA

the simulation results can be considered as an upper bound.

As shown in Figure 5.3, the vulnerable window V of the system varies with the nodes' moving speed but there is no evidence of a clear relation between them. The same observation holds for V vs CA composition relation as demonstrated in Fig 5.4. After analyzing the trace logs of the simulations, we found that routing between CA's dominated the trust propagation delay along the chain, which was relatively random compared to CA internal routing where usually routes were shorter. Fig 5.5 further supports this by showing V increases as density of the network (represented by the number of nodes per 10000 square meters(m^2)) decreases. This is because there are

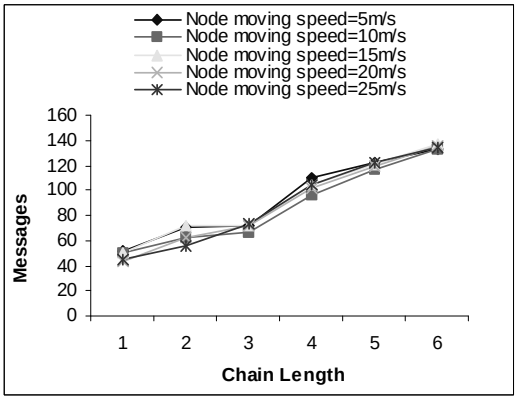

Figure 5.7: Message Cost(M) of Trust Chain Management vs Node Speed

less routes available between nodes in sparse networks.

Unlike the vulnerable window size V, the message cost is in direct proportion to the number of servers(N) and threshold K of CA as expected as shown in Figure 5.6. Figure 5.7 shows that network density and volatility do not impact the message cost significantly. This is because we did not rely on retransmission to fail over routing problem or network partition, due to the high cost. Instead, we used routine and event based updates to repair broken chains, in which, if an update notification fails to reach other CA's, it is simply dropped and the system waits for either next routine or event-triggered update.

5.5 Limitations

The main weakness of our architecture is the commonly trusted dealer in each community. This can be risky in general since it introduces a single point of failure. Bootstrap can also be accomplished without the help of the dealer as discussed in [73]. Our key architecture did not implement this method in order to favor performance and simplicity. We mitigated this problem from two aspects. First, it is not unusual that such a dealer does exist, e.g., the chair of a workshop or the provider of some MANET services. Second, the trust is only needed at the bootstrap phase and can be revoked after CAs are fully setup.

A more secure solution to the dealer problem is to leverage Satem in the CA bootstrap process. To implement this method, we first need to develop a trusted CA dealer service that securely generates the CA private key and destroys it immediately after distributing the partial shares. The service is defined in a Satem commitment. Before a CA server accepts the partial share from the dealer, it establishes trust on the dealer through the Satem service commitment protocol. As we discussed in Chapter 2, this ensures the CA server that the dealer will dispose the private key properly.

CA servers of a community may fail or leave the community. Due to the threshold scheme, losing less than $N - K$ servers will not cause the CA to stop functioning. However, if more than $N - K$ servers leave the community, the CA is broken. We have not addressed this problem in the current approach. A possible method to tackle this problem is to require each CA server to find a successor before leaving and securely transfer its partial secret to the successor. Then the successor notifies other CA servers and generates a new CA certificate describing the new composition.

Certain optimizations can also be incorporated into the current implementation. The most important one is to implement the delay timer as mentioned in Section 4. Besides, one can use real multicast to send ESTABLISH and UPDATE messages instead of using multiple unicasts. With these optimizations in place, the overhead of our architecture can be further reduced.

Trust chains are only used for key authentication in this work due to our narrow definition of trust, i.e., the authenticity of nodes' public keys. Their use can be expanded by adding more semantics to the trust definition. For instance, when trust is based on nodes' behavior, the CAs actually become trust monitors and any node can verify if another node is decent by use of the trust chains.

5.6 Summary

In this chapter, we presented a locality driven key management architecture for MANET. The design is motivated by the application centric nature of MANET and based on threshold cryptography to achieve high fault tolerance against network partition and

malicious nodes. On top of it, we designed distributed trust protocols to help set up trust relations on-the-fly. We implemented a prototype for both Intel based laptops and ARM based iPAQ PDAs. We also ran NS-2 simulations for different scenarios. The results support our design goals of high fault tolerance, in-time services and efficiency.

Chapter 6

Conclusion and Future Direction

In this dissertation, we investigated how to ensure trust in mobile ad hoc networks by exploiting the application centric nature of MANETs and augmenting network nodes with low cost hardware based trusted computing system. The main questions that this dissertation has tried to answer are:

1. *How to ensure a MANET application user on one node that the application running on another node can be trusted?*
2. *How to ensure fair and secure communication between multiple network nodes?*
3. *How to ensure that a network node is what it claims to be?*

We presented the design, implementation, and evaluation of Satem, a service-aware trusted execution monitor, for ensuring trusted code execution. Users establish trust with the applications running on other nodes through a service commitment protocol executed before starting any new application transaction. During this protocol, Satem provides the users a service commitment, which describes all the code files that the service may execute in all circumstances, such as executables, libraries, etc. Satem exploits the TCG-specified TPM as the root of trust to build trusted components including the execution monitor in the OS kernel, which consequently enforces the service commitment. Satem achieves service awareness by limiting the scope of monitoring to protected applications instead of performing attestation and protection on all programs. We have implemented a prototype under Linux and evaluated it using two MANET applications. The experimental results demonstrate that Satem incurs low overhead to both the applications and the underlying platform without impacting the performance of unprotected applications.

On top of Satem, we developed two methods to ensure trusted communication between network nodes: one is a method that creates protected MANETs to shield network member nodes from network attacks, and the other is an application layer network communication policy enforcement mechanism to ensure secure and cooperative communication between network participants. Both methods are driven by the application centric nature of MANETs, i.e., the creation of MANETs is triggered by users who want to run some common applications, but differs in the layer and the granularity of control. The former method does not allow untrusted nodes to establish wireless links with nodes in the protected networks. Furthermore, it enforces a common network access control policy at the network layer; this policy is associated with the application running in the network. Attacks from member nodes are suppressed locally by the common network access control policy. To ensure trusted enforcement of the policy, we augmented every node with a Satem-based trusted agent. We evaluated the method through a prototype based on an IEEE 802.11 ad hoc network. The results demonstrate that our method imposes little impact on network communication.

The latter method moves the trust to the application layer and is capable of controlling the behavior of individual applications by enforcing application context aware policies. Under the latter method, nodes supporting the same set of applications and enforcing the same policies construct a trusted multi-tier application centric network. Each tier of the network runs one application and enforces its associated policy. The application of the upper tier depends on the applications of the lower tiers to communicate. Only trusted nodes are allowed to join the network. Moreover, communication between them is regulated by the policies at every tier. Similar to the method of creating protected ad hoc networks, trusted policy enforcement is guaranteed by our Satem-based trusted kernel agent. We evaluated the method through a prototype based on an IEEE 802.11 ad hoc network and through network simulations. The results demonstrate the feasibility of the proposed method as well as its low overhead.

Orthogonal to the behavior based trust, such as trusted application and trusted communication, we presented a locality driven key management architecture to ensure trusted identity. The design leveraged the application centric nature of MANET to

enhance trustworthiness and performance of authentication. Moreover, the method is based on threshold cryptography to achieve high fault tolerance against network partition and malicious nodes. Trust relationships are managed dynamically by use of our distributed trust protocols. We evaluated the method through prototyping on both Intel based laptops and ARM based iPAQ PDAs and NS-2 simulations. The results support our design goals of high fault tolerance, in-time services and efficiency.

The main conclusion of this dissertation is that the emerging low cost trusted hardware combined with the application centric nature of MANETs can be exploited to provide solutions to the problems of lack of trust in MANETs, which would otherwise be impossible.

6.1 Future Direction

Looking forward, we believe that application centric view with assistance of low cost trusted hardware is conducive to improve trust in MANETs. One potential application is to implement a distributed credit system, which maintains the credit record of a specific node for an application like TrafficView [46] while preserving the node's privacy. The existing trust system can be leveraged to assign a credit score to each network member. However, these methods suffer two problems. First, it takes long time to accumulate credit history while MANET applications are usually transitory. As a result, the history of the node executing the same applications must be carried over. Second, due to the lack of central credit authority, there is nowhere to save the credit history except on the node itself. A potential solution to this problem is to implement a trusted system on each node. Similar to the Satem, this trusted system must guarantee that the credit updating system (that is also running on the node) is trusted. Moreover, it must perverse integrity and privacy of the credit records it receive. We are investigating an approach based on the configuration verification feature of TPM.

Another potential topic is a Digital Right Management (DRM) system. With the popularity of high-definition media and the accelerating deployment of IPTV and peer-to-peer video sharing, we expect DRM to become a much more severe problem due to

the high economic interest. The existing solutions are based on dedicated client devices, such as IPTV residential gateways. This approach has two drawbacks. First, the client devices are closed-box. Therefore, the solution developed on top of it is not applicable to open-box devices like laptops and palm computers. Second, these client devices are far from tamper-proof. We are investigating a TPM based solution, which guarantees that the media would only be played by authorized trusted media players and can not be saved illegally.

References

[1] 802.11 protocols. http://www.ieee802.org/11/.

[2] AODV Implementation. http://core.it.uu.se/core/index.php/AODV-UU.

[3] Atmel tpm. http://www.atmel.com/.

[4] Carnegie mellon cert coordination center. http://www.cert.org.

[5] Dictionary Definition of Trust. *Merriam-Webster Online Dictionary*. http://www.merriam-webster.com/dictionary/trust.

[6] Enforcer project. http://enforcer.sourceforge.net.

[7] Entrust Certificate Service. http://www.entrust.com.

[8] Facebook. http://www.facebook.com.

[9] Hostapd Project. http://hostap.epitest.fi/.

[10] IEEE 802.1X Port-based Network Access Control. In *IEEE Standard 802.1X, 2001 Edition*.

[11] Mobile Ad hoc Networks. http://en.wikipedia.org/wiki/Mobile_ad-hoc_network.

[12] Mute p2p file sharing application. http://mute-net.sourceforge.net/.

[13] MySpace - a place for friends. http://www.myspace.com.

[14] Open Shortest Path First (OSPF). http://www.ietf.org/rfc/rfc1247.txt.

[15] Open1x project. http://open1x.sourceforge.net.

[16] OpenSSL Project. http://www.openssl.org.

[17] Oprofile project. http://oprofile.sourceforge.net.

[18] The HiperLan project. http://en.wikipedia.org/wiki/HIPERLAN.

[19] The netfilter/iptables Project. http://www.netfilter.org.

[20] The Network Simulator - NS2. http://www.isi.edu/nsnam/ns.

[21] The SANS Institute. http://www.sans.org.

[22] WiFi Project. http://www.wi-fi.org/.

[23] Extensible Authentication Protocol Over Lan. In *IEEE Standard EAPOL*, 2000.

[24] Ehab Al-Shaer and Hazem Hamed. Discovery of policy anomalies in distributed firewalls. In *Proceedings of Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, 2004.

[25] W Arbaugh, D Farber, and J Smith. A secure and reliable bootstrap architecture. In *Proceedings of IEEE Symposium on Security and Privacy*, 1997.

[26] William A. Arbaugh, Narendar Shankar, and Y.C. Justin Wan. Your 802.11 wireless network has no clothes. *Wireless Communications, IEEE*, 9(1):44–51, 2002.

[27] N Aschenbruck, M Frank, P Martini, and J Tolle. Human mobility in manet disaster area simulation - a realistic approach. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, pages 668–675, 2004.

[28] N. Asokan and Philip Ginzboorg. Key agreement in ad-hoc networks. In *Nordsec'99 Workshop*, 1999.

[29] T. Aura and S. Mäki. Towards a survivable security architecture for ad-hoc networks. In *Security Protocols, 9th International Workshop, Cambridge, UK, April 2001*, volume 2467 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin. Springer-Verlag Berlin Heidelberg 2002.

[30] Walid Bagga, Stefano Crosta, Pietro Michiardi, and Refik Molva. Establishment of ad-hoc communities through policy-based cryptography. In *the Proceedings of 2nd Workshop on Cryptography for Ad hoc Networks (WCAN)*, July 2006.

[31] D. Balfanz, D. Smetters, P. Stewart, and H. Wong. Talking to strangers: Authentication in adhoc wireless networks. In *the Symposium on Network and Distributed Systems Security (NDSS '02), San Diego, California*, 2002.

[32] John Bellardo and Stefan Savage. 802.11 denial-of-service attacks: Real vulnerabilities and practical solutions. In *Proceedings of the 8th USENIX Security Syposium (Security'03)*, 2003.

[33] S. Berkovits, S. Chokhani, J. Furlong, J. Geiter, and J. Guild. Public key infrastructure study final report. *MITRE report*, 1994.

[34] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The keynote trust-management system, version 2. In *RFC 2704*, September 1999.

[35] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *the Proceedings of IEEE Conference on Privacy and Security*, 1996.

[36] Cristian Borcea, Chalermek Intanagonwiwat, Porlin Kang, Ulrich Kremer, and Liviu Iftode. Spatial programming using smart messages: Design and implementation. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 690–699, 2004.

[37] David F.C. Brewer and Micheal J. Nash. The chinese wall security policy. In *the Proceedings of IEEE Conference on Privacy and Security*, 1989.

[38] J. Broch, D. A. Maltz, D. Johnson, Y. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Mobile Computing and Networking*, 1998.

[39] E.ROYER C. PERKINS and S. DAS. Ad-hoc on-demand distance vector - ad hoc on-demand distance vector (aodv) routing. In *RFC 3561*, Jul 2003.

[40] S. Capkun, L. Buttyan, and J. P. Hubaux. Self-organized public-key management for mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, Jan-Mar 2003.

[41] Augustin Chaintreau, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, and James Scottr. Pocket Switched Networks: Real-world mobility and its consequences for opportunistic forwarding. Technical Report UCAM-CL-TR-617, Computer Laboratory of University of Cambridge, 2005.

[42] Benjie Chen and Robert Morris. Certifying program execution with secure processors. In *the Proceedings of 9th Workshop on Hot Topics in Operating Systems*, 2003.

[43] Krishna Chintalapudi, Tat Fu, Jeongyeup Paek, Nupur Kothari, Sumit Rangwala, John Caffrey, Ramesh Govindan, Erik Johnson, and Sami Masri. Monitoring civil structures with a wireless sensor network. *IEEE Internet Computing*, 10(2):26–34, 2006.

[44] Jim Chow, Ben Pfaff, Tal Garfinkel, Kevin Christopher, and Mendel Rosenblum. Understanding data lifetime via whole system simulation. In *Proceedings of the 13th Usenix Security Symposium*, 2004.

[45] S. Dashtinezhad, T. Nadeem, C.Liao, and L. Iftode. Trafficview: A scalable traffic monitoring system. In *the proceedings of IEEE International Conference on Mobile Data Management*, 2004.

[46] S. Dashtinezhad, T. Nadeem, B. Dorohonceanu, C. Borcea, P. Kang, and L. Iftode. Trafficview: A driver assistant device for traffic monitoring based on car-tocar communication. In *In Proceedings of IEEE Semiannual Vehicular Technology Conference (VTC'04-Spring)*, 2004.

[47] Gang Ding and Bharat Bhargava. Peer-to-peer file-sharing over mobile ad hoc networks. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, page 104, 2004.

[48] P Dinsmore, D Balenson, M Heyman, P Kruus, C Scace, and A Sherman. Policy-based security management for large dynamic groups: An overview of the dccm project. In *the Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX00)*, January 2000.

[49] John R. Douceur. The sybil attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, 2002.

[50] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 2003.

[51] Keith I. Farkas, John Heidemann, and Liviu Iftode. Intelligent transportation and pervasive computing. In *IEEE Pervasive Computing Magazine*, number 4, pages 18–19, 2006.

[52] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of RC4. *Lecture Notes in Computer Science*, 2259, 2001.

[53] W Franz, R Eberhardt, and T Luckenbach. Fleetnet - internet on the road. In *Proceedings of 8th World Congress on Intelligent Transportation Systems (ITS 2001)*, 2001.

[54] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 2003.

[55] M Gasser, A Goldstein, C Kaufman, and B Lampson. The digital distributed system security architecture. In *Proceedings of 12th NIST-NCSC National Computer Security Conference*, 1989.

[56] R Goldberg. Survey of virtual machine research. In *IEEE Computer Magazine*, June 1974.

[57] M Guarnera, M Villari, A Zaia, and A Puliafito. Manet: possible applications with pda in wireless imaging environment. In *the Proceedings of the 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 2394–2398, 2002.

[58] J Haartsen. Bluetooth - the universal radio interface for ad hoc, wireless connectivity. In *Ericsson Review*, number 3, pages 110–117, 1998.

[59] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Ariadne: a secure on-demand routing protocol for ad hoc networks. In *Proceedings of the 8th annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 12–23, September 2002.

[60] Yih-Chun Hu, Adrian Perrig, and David B Johnson. Packet leashes: A defense against wormhole attacks in wireless ad hoc networks. In *the Proceedings of Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, April 2003.

[61] Jean-Pierre Hubaux, Srdjan Čapkun, and Jun Luo. The security and privacy of smart vehicles. *IEEE Security and Privacy*, 2(3), 2004.

[62] Pan Hui, Augustin Chaintreau, James Scott, Richard Gass, Jon Crowcroft, and Christophe Diot. Pocket switched networks and human mobility in conference environments. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking (WDTN05)*, pages 244–251, 2005.

[63] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11(1):2–16, 2003.

[64] S Ioannidis, A Keromytis, S Bellovin, and J Smith. Implementing a distributed firewall. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'00)*, 2000.

[65] J.Jubin and J.D.Tornow. Darpa packet radio network protocol. In *Proceedings of the IEEE*, volume 75, pages 21–32, Jan 1987.

[66] D. Johnson, D. Maltz, and J. Broch. *DSR The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks*, chapter 5, pages 139–172. Addison-Wesley, 2001.

[67] G Karjoth. The authorization service of tivoli policy director. In *the Proceedings of the 17th Computer Security Applications Conference (ACSAC)*, December 2001.

[68] Rick Kennell and Leah H. Jamieson. Establishing the genuinity of remote computer systems. In *Proceedings of 12th USENIX Security Symposium*, 2003.

[69] Sye Loong Keoh, Emil Lupu, and Morris Sloman. Peace: A policy-based establishment of ad-hoc communities. In *the Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC)*, September 2004.

[70] Jie. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing robust and ubiquitous security support for mobile ad-hoc networks. In *the Proceedings of the 9th IEEE International Conference on Network Protocols (ICNP'01)*, 2001.

[71] T Leinmuller, E Schoch, F Kargl, and C Maihofer. Influence of falsified position data on geographic ad-hoc routing. In *Proceedings of the second European Workshop on Security and Privacy in Ad hoc and Sensor Networks (ESAS 2005)*, 2005.

[72] David Lie, Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. Architectural support for copy and tamper resistant software. In *Architectural Support for Programming Languages and Operating Systems*, 2000.

[73] M. Malkin, T. Wu, and D. Boneh. Experimenting with shared generation of rsa keys. *In Proceedings of the Internet Society's 1999 Symposium on Network and Distributed System Security (SNDSS)*, 1999.

[74] A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *Proceedings of IEEE Symposium on Security and Privacy (S&P'00)*, 2000.

[75] Patrick McDaniel and Atul Prakash. Enforcing provisioning and authorization policy in the antigone system. In *Journal of Computers (JCP)*, November 2006.

[76] MeshDynamics Inc. MD4455/MD4325 Mobile Node with Mobility Scanner. http://www.meshdynamics.com/FAQ_4455.html.

[77] Microsoft Corp. Next generation secure computing base. http://www.microsoft.com/resources/ngscb.

[78] Naftaly Minsky Mihail Ionescu and Thu Nguyen. Enforcement of communal policies for peer-to-peer systems. In *the Proceedings of 6th International Conference on Coordination Models and Languages*, February 2004.

[79] Naftaly Minsky. Decentralized regulation of distributed systems: Beyond access control, April 2008. http://www.cs.rutgers.edu/ minsky/papers/IC.pdf.

[80] Naftaly Minsky and Victoria Ungureanu. Unified support for heterogeneous security policies in distributed systems. In *the Proceedings of 117th USENIX Security Symposium*, January 1998.

[81] Naftaly Minsky and Victoria Ungureanu. Law-governed interaction: A coordination & control mechanism for heterogeneous distributed systems. In *ACM Transactions on Software Engineering and Methodology (TOSEM)*, July 2000.

[82] Arunesh Mishra, Jr. Nick L. Petroni, William A. Arbaugh, and Timothy Fraser. Security issues in ieee 802.11 wireless local area networks: A survey. *Wireless Communication & Mobile Computing*, 4(8):821–833, 2004.

[83] MOBIDIS Project. ManetChat Application on the MOBIDIS Middleware. http://www.dis.uniroma1.it/pub/mecella/projects/MobiDIS/mobidis_user.htm.

[84] Robert Morris, John Jannotti, Frans Kaashoek, Jinyang Li, and Douglas Decouto. Carnet: a scalable ad hoc wireless network system. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 61–65, 2000.

[85] T Murata and N Minsky. Regulating work in digital enterprises: A flexible managerial framework. In *the Proceedings of the Cooperative Information Systems Conference (CoopIS)*, October 2002.

[86] Tamer Nadeem, Sasan Dashtinezhad, Chunyuan Liao, and Liviu Iftode. Trafficview: traffic data dissemination using car-to-car communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 8(3), 2004.

[87] J Nzouonta, N Rajgure, G Wang, and C Borcea. Vanet routing on city roads using real-time vehicular traffic information. 2007. Under submission.

[88] Bryan Parno and Adrian Perrig. Challenges in securing vehicular networks. In *Proceedings of Workshop on Hot Topics in Networks (HotNets-IV)*, 2005.

[89] V. Pathak and L. Iftode. Byzantine fault tolerant authentication. *Technical Report, Dept of Computer Science, Rutgers University*, 2003.

[90] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *SIGCOMM Computer Communication Review*, 24(4):234–244, 1994.

[91] Charles E. Perkins, Elizabeth Royer, and Samir R. Das. Ad hoc on demand Distance Vector(AODV) routing. In *2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*, 1999.

[92] Adrian Perrig, R. Canetti, D.Tygar, and Dawn Song. The TESLA Broadcast Authentication Protocol. In *RSA Cryptobytes*, 2002.

[93] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Netowrks. In *Proceedings of the 7th annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 189–199, July 2001.

[94] Tuan Phan, Zhijun He, and Thu D. Nguyen. Using firewalls to enforce enterprise-wide policies over standard client-server interactions. In *Journal of Computers (JCP)*, April 2006.

[95] Shankar Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. Icrafter: A service framework for ubiquitous computing environments. In *Proceedings of the 3rd international conference on Ubiquitous Computing*, pages 56–75, 2001.

[96] David Rising. Battlefield internet helps forces in iraq. *Globe Technology*, 2003.

[97] Manuel Roman and Roy H. Campbell. Gaia: Enabling active spaces. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 229–234, 2000.

[98] David Safford, Jeff Kravitz, and Leendert van Doorm. Take control of TCPA. *Linux Journal, August, 2003*. http://www.linuxjournal.com/article/6633.

[99] R Sailer, T Jaeger, X Zhang, and L van Doorn. Attestation-based policy enforcement for remote access. In *Proceedings of 11th ACM Conference on Computer and Communications Security*, 2004.

[100] R Sailer, X Zhang, T Jaeger, and L van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of 13th USENIX Security Symposium*, 2004.

[101] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert Van Doorn, and Pradeep Khosla. Pioneer: Verifying integrity and guaranteeing execution of code on legacy platforms. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, 2005.

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

[103] Elaine Shi, Adrian Perrig, and Leendert van Doorn. Bind: A time-of-use attestation service for secure distributed system. In *Proceedings of IEEE Symposium on Security and Privacy*, 2005.

[104] Jaewon Shin. *Multi-object Tracking and Identity Management in Wireless Sensor Networks*. PhD thesis, 2005. Adviser-Leonidas J. Guibas.

[105] V. Shoup. Practical threshold signatures. *Theory and Application of Cryptographic Techniques*, 2000.

[106] Trusted Computing Group. TCG 1.2 Specifications. https://www.trustedcomputinggroup.org/.

[107] Trusted Computing Group - Mobile Phone Working Group. Use Case Scenarios - v 2.7.

[108] Hua-Wen Tsai, Chih-Ping Chu, and Tzung-Shi Chen. Mobile object tracking in wireless sensor networks. *Computer Communications*, 30(8):1811–1825, 2007.

[109] Vincent S. Tseng and Kawuu W. Lin. Energy efficient strategies for object tracking in sensor networks: A data mining approach. *Journal of Systems and Software*, 80(10):1678–1698, 2007.

[110] Jeffrey Voas. A Recipe for Certifying High Assurance Software. In *IEEE Software*, 1999.

[111] Carl A. Waldspurger. Memory resource management in VMware ESX server. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 2002.

[112] X Wang, Y Yin, and H Yu. Finding collisions in the full SHA1. In *Proceedings of Crypto*, 2005.

[113] Steve H. Weingart. Physical security for the uABYSS system. In *Proceedings of IEEE Computer Society Conference on Security and Privacy*, pages 52–58, 1987.

[114] Geoffrey Werner-Allen, Konrad Lorincz, Matt Welsh, Omar Marcillo, Jeff Johnson, Mario Ruiz, and Jonathan Lees. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, 10(2):18–25, 2006.

[115] S White, S Weingart, W Arnold, and E Palmer. Introduction to the Citadel architecture: Security in physically exposed environments. Technical Report TR RC16672, IBM Thomas J. Watson Research Center, 1991.

[116] T Woo and S Lam. A framework for distributed authorization. In *the Proceedings of the 1st ACM Conference on Computer and Communications Security*, November 1993.

[117] Gang Xu, Cristian Borcea, and Liviu Iftode. Satem: A Service-aware Attestation Method Toward Trusted Service Transaction. In *the Proceedings of IEEE Symposium on Reliable Distributed Systems (SRDS)*, October 2006.

[118] Gang Xu, Cristian Borcea, and Liviu Iftode. Trusted application-centric ad-hoc networks. In *the Proceedings of the 4th IEEE International Conference on Mobile Ad-hoc Networks and Sensor Systems (MASS 2007)*, 2007.

[119] Gang Xu, Cristian Borcea, and Liviu Iftode. A Policy Enforcing Mechanism for Trusted Ad hoc Networks. Technical Report DCS-tr-635, Computer Science Department of Rutgers University, New Brunswick, 2008.

[120] Gang Xu and Liviu Iftode. Locality driven key management architecture. In *the Proceedings of the 1st IEEE International Conference on Mobile Ad-hoc Networks and Sensor Systems (MASS 2004)*, 2004.

[121] Y.D.Lin and Y.C.Hsu. Multihop cellular: A new architecture for wireless communications. In *Proceedings of IEEE INFOCOM*, pages 1273–1282, 2000.

[122] B Yee. Using secure co-processors. Technical report, Carnegie Mellon University, 1994. PH.D Thesis.

[123] S. Yi and R. Kravets. Moca: Mobile certificate authority for wireless ad hoc networks. In *the Proceedings of the 10th IEEE International Conference on Network Protocols (ICNP'02)*, 2002.

[124] L. Zhou and Z. J. Haas. Securing ad hoc networks. In *IEEE Networks*, volume 13(6). 1999.

[125] L. Zhou, F. Schneider, and R. van Renesse. Coca: A secure distributed on-line certification authority. In *Technical Report of Cornell University*. 2002.

[126] P. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.

Vita

Gang Xu

Education

Ph.D. Computer Science, Rutgers University, New Jersey (2008)

M.S. Computer Science, Florida International University, Florida (1999)

B.S. Computer Science, Nanjing Univ of Aero & Astro, China (1995)

Professional Experience

Member of Technical Staff, AT&T Labs, 1999 - 2001

Senior Member of Technical Staff, AT&T Labs, 2001 - 2006

Principal Member of Technical Staff, AT&T Chief Security Office, 2006 - present

Publications

- Gang Xu, Cristian Borcea, and Liviu Iftode "A Policy Enforcing Mechanism for Trusted Ad hoc Networks". In Technical Report DCS-tr-635, Computer Science Department of Rutgers University, New Brunswick, 2008.

- Gang Xu, Cristian Borcea, and Liviu Iftode "Trusted Application-centric Ad-hoc Networks". In the Proceedings of the 4th IEEE International Conference on Mobile Ad-hoc Networks and Sensor Systems (MASS 2007), 2007.

- Gang Xu, Cristian Borcea, and Liviu Iftode "Satem: A Service-aware Attestation Method Toward Trusted Service Transaction". In the Proceedings of IEEE Symposium on Reliable Distributed Systems (SRDS 2006), October 2006.

- Gang Xu and Liviu Iftode "Locality driven key management architecture". In the Proceedings of the 1st IEEE International Conference on Mobile Ad-hoc Networks and Sensor Systems (MASS 2004), 2004.

- Porlin Kang, Cristian Borcea, Gang Xu, Akhilesh Saxena, Ulrich Kremer, and Liviu Iftode "Smart Messages: A Distributed Computing Platform for Networks of Embedded System". The Computer Journal, Special Focus on Mobile and Pervasive Computing, Volume 47, British Computer Society, Oxford University Press, July 2004.

- Gang Xu, Cristian Borcea, and Liviu Iftode "Toward a Security Architecture for Smart Messages: Challenges, Solutions, and Open Issues". In the Proceedings of the 1st International Workshop on Mobile Distributed Computing (MDC 2003), May 2003.
- Avigdor Gal, Vijayalakshmi Atluri, and Gang Xu "An Authorization System for Temporal Data", In the Proceedings of the 18th International conference on Data Engineering Description (ICDE 2002), February 2002.