USING SECURE COPROCESSORS TO ENFORCE NETWORK ACCESS POLICIES IN ENTERPRISE AND AD HOC NETWORKS

by

Haidong Xia

B.S., Peking University, 1995

M.S., University of Pittsburgh, 2000

Submitted to the Graduate Faculty of

the Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2008

UNIVERSITY OF PITTSBURGH

ARTS AND SCIENCES

This dissertation was presented

by

Haidong Xia

It was defended on

April 7, 2008

and approved by

José Carlos Brustoloni, Department of Computer Science

Rami Melhem, Department of Computer Science

Ahmed Amer, Department of Computer Science

James B. D. Joshi, Department of Information Science & Telecommunications

Dissertation Director: José Carlos Brustoloni, Department of Computer Science

USING SECURE COPROCESSORS TO ENFORCE NETWORK ACCESS POLICIES IN ENTERPRISE AND AD HOC NETWORKS

Haidong Xia, PhD

University of Pittsburgh, 2008

Nowadays, network security is critically important. Enterprises rely on networks to improve their business. However, network security breaches may cause them loss of millions of dollars. Ad hoc networks, which enable computers to communicate wirelessly without the need for infrastructure support, have been attracting more and more interests. However, they cannot be deployed effectively due to security concerns.

Studies have shown that the major network security threat is insiders (malicious or compromised nodes). Enterprises have traditionally employed network security solutions (e.g., firewalls, intrusion detection systems, anti-virus software) and network access control technologies (e.g., 802.1x, IPsec/IKE) to protect their networks. However, these approaches do not prevent malicious or compromised nodes from accessing the network. Many attacks against ad hoc networks, including routing, forwarding, and leader-election attacks, require malicious nodes joining the attacked network too.

This dissertation presents a novel solution to protect both enterprise and ad hoc networks by addressing the above problem. It is a hardware-based solution that protects a network through the attesting of a node's configuration before authorizing the node's access to the network. Attestation is the unforgeable disclosure of a node's configuration to another node, signed by a secure coprocessor known as a Trusted Platform Module (TPM).

This dissertation makes following contributions. First, several techniques at operating system level (i.e., *TCB prelogging*, *secure association root tripping*, and *sealing-free attestation confinement*) are developed to support attestation and policy enforcement. Second,

two secure attestation protocols at network level (i.e., *Bound Keyed Attestation* (BKA) and *Batched Bound Keyed Attestation* (BBKA)) are designed to overcome the risk of a man-in-the-middle (MITM) attack. Third, the above techniques are applied in enterprise networks to different network access control technologies to enhance enterprise network security. Fourth, *AdHocSec*, a novel network security solution for ad hoc networks, is proposed and evaluated . AdHocSec inserts a security layer between the network and data link layer of the network stack. Several algorithms are designed to facilitate node's attestation in ad hoc networks, including *distributed attestation* (DA), and *attested merger* (AM) algorithm.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

This dissertation would, not have been possible without the help of many people, including my advisor, committee members, colleagues, the faculty and staff of my department, and my family!

First, I am very grateful to my advisor, Prof. José Carlos Brustoloni, for his guidance and support of my Ph.D. study. Prof. Brustoloni led me into the field that I like very much and gave me detailed advice about the projects that I have worked on, including my dissertation work. He is very patient and teaches me everything essential to be a researcher, which will be invaluable to me for the rest of my life.

Second, I thank my committee members, who have made my work better and stronger. Each talk with any of them is very informative and fruitful. Prof. Rami Melhem has always been encouraging me on Ph.D study. Prof. Daniel Mossé has provided not only suggestions and feedback on my dissertation, but also guidances for my future work and life . Prof. Amer is very nice and always there to help. Prof. James Joshi has always provided valuable suggestions quickly, despite his busy schedule.

Third, I would like to thank my colleagues and the faculty and staff of the Computer Science department. They have helped in many aspects of my Ph.D. study during the several years I have been at Pitt. The time I have shared with them is memorable. Special thanks go to Jiang Zhang for her many helpful responses to my dissertation while she has been busy.

Finally, I do not know how to adequately thank my family. This dissertation is my entire family's contribution. I thank my parents for their patience, my sisters for their encouragement, and my beautiful wife, Shihua Ren, for her invaluable support, which has helped me continue through all the hardships. I also thank my lovely son, Andrew, whose arrival during the final stage of my dissertation has brought me a lot of happiness.

1.0 INTRODUCTION

The widespread deployment of networks in the world has made network security critically important, and this urgency has been heightened by the development and spread of new, fast-growing wireless technology. Attackers are able to strike targets not only remotely, from outside the targeted network, but also from within it by using compromised or malicious nodes inside the network. In addition to causing enterprises to suffer significant economic losses, such attacks have impeded the application of new technologies, specifically ad hoc networks.

The most common security strategy is to set up a firewall, which establishes a security perimeter for the network. This approach relies on two assumptions: (1) nodes inside the perimeter are trustworthy and (2) all attacks originate from outside. However, in reality, the security perimeter often can be penetrated; thus, its effectiveness is limited. For example, employees can bring compromised laptops into their offices, viruses or worms can propagate into the network through email attachments, and spyware [1] can be downloaded into the network by employees. Once inside the security perimeter, only fragile defenses such as anti-virus software (AV) and intrusion detection systems (IDSs) are barriers to the attack. AV depends on virus signatures to detect and remove malicious codes; an IDS may rely on signatures, or it may be anomaly-based. Signature-based AV and IDS can identify known attacks precisely, but demand frequent updates and are entirely ineffective against new forms of attacks. Anomaly-based IDSs can detect new forms of attacks, but they usually generate an excessive number of false alarms.

To prevent attackers from entering an enterprise network, newly-initiated approaches such as Microsoft's NAP (Network Access Protection)) [2] and Cisco's NAC (Network Admission Control) [3] verify a node's configuration to the protected network's security policies

prior to allowing it network access. In NAP, before connecting to an Intranet, a host sends a list of its software components and configuration to a network-designated server. This list indicates, for example, what operating system and anti-virus software are installed in the host, and it details the version, security patches, and virus definitions. If the server determines that the host's software is up-to-date and acceptable, it allows the host to connect to the Intranet. Otherwise, the server confines the host to a restricted network, which only allows software to be updated and brought into compliance with Intranet policies. NAC extends this concept to control the connections of access points and other devices in addition to hosts. The use of such access control functionality by networks and nodes is expected to become commonplace during the next several years. However, NAP and NAC are weak against malicious users. The list of a node's configuration and software components can be forged or modified easily. In this way, attackers can circumvent NAP and NAC with a node that can greatly harm the network's security.

Ad hoc networks [4] can also be targeted by nodes with malicious software, for example in routing, forwarding, and leader-election attacks. Ad hoc networks enable nodes to communicate wirelessly without any infrastructure; nodes rely on one another to forward packets to their destinations. Such networks have important applications in military operations, emergency response, and impromptu meetings, yet their security is even worse than that of wired networks. They are subject to many security problems, including routing disruption and nodes' selfish forwarding. Cryptographic techniques (i.e., the use of symmetric or asymmetric cryptographic algorithms) have been proposed to be appropriate for securing routing in such networks. However, cryptography has problems in ad hoc networks; it requires efficient and robust key management and revocation, which can be very difficult for ad hoc networks to achieve. Node cooperation enforcement systems [5, 6], which help identify ill-behaved nodes in a network, have been proposed as a defense against selfish nodes, but attackers may leverage this defense to malign legitimate nodes. Finally, while many secure routing protocols have been proposed, they may depend on some assumptions which are hard to achieve or need support from other solutions. Altogether, despite the existence of several proposed solutions, there is no comprehensive solution for ad hoc networks that can solve both the problems of security routing and node cooperation enforcement.

This dissertation addresses the security problems caused by malicious or compromised nodes. We realize that many attacks against a network can be prevented by the network authenticating a node's configuration, rather than just its user identity, before authorizing its access to the network. By enforcing required network access policies for all nodes attempting to access a protected network, we can ensure both that attackers or compromised nodes are prevented from getting into a protected network and that nodes which initially gain access, but later violate required network access policies will be excluded from the network.

In this study, we used secure coprocessors–Trusted Platform Modules (TPM) [7]–to help enforce network security policies for both enterprise and ad hoc networks. TPM is a secure chip designed by Trusted Computing Group (TCG) [8]. It provides the functionality of attestation, enabling one party to attest the validity of the software and hardware configuration of another, without the weaknesses of NAP and NAC. It is cost-effective (only $4 a chip), and it is embedded in many computers, including IBM ThinkPad laptops, IBM NetVisa workstations, and HP D530 workstations.

The research discussed in this dissertation imposes several challenges on existing operating systems (OSs) and network protocols. First, general OSs do not support TPM attestation. TCG does not provide specifications for how its TPM-provided interfaces should be used by the operating system after boot time. Second, most OSs have privileged users (i.e., root) with authority to modify the OS or its configuration at any time. Modification of the OS's configuration after boot time can violate attestation consistency, and this problem is not addressed by TCG. Third, TPM's sealing function can cause lock-in problems, binding data access to a specific application and platform, and preventing users of other OSs or applications from accessing the data.

To address these barriers, we enhanced an existing OS for our approach to work. We developed TCB prelogging (discussed in section 3.2.1.1) and security association root tripping (discussed in section 3.2.1.2) to establish and maintain attestation consistency, and sealing free attestation confinement (discussed in section 3.2.2) to prevent lock-in problems.

Challenges to existing network protocols include both the design of a secure attestation protocol and the integration of attestation in existing network access control protocols. First, the attestation protocol as specified by TCG is incomplete and is vulnerable to man-

in-the-middle (MITM) attacks, if additional precautions are not taken. This risk cannot be eliminated simply by tunneling attestation packets by using, e.g., transport layer security protocol (TLS) [9]. Second, the TPM attestation process is slow (about 0.71 second), which might affect network performance if a node receives several attestation requests simultaneously. Third, applying attestation in enterprise and ad hoc networks require different techniques.

To address these concerns, we developed a novel form of attestation, called Bound Keyed Attestation (BKA, discussed in section 4.2.1) and a new attestation method dealing with multiple attestation requests, called Batched Bound Keyed Attestation (BBKA, discussed in section 4.2.2). BKA can thwart MITM attacks. BBKA allows a single TPM quote to be used for multiple attestation requests, which speeds up the attestation process so that network performance is not degraded. We use different approaches to enforce network access policies in each type of network. For enterprise networks, we integrate the attestation protocol with existing network access methods (i.e., 802.1x [10] for wired and wireless access, IPsec/IKE [11, 12] for remote access–virtual private network (VPN)). For ad hoc networks, since each node of an ad hoc network is independent and there is no central authentication server, we apply a distributed attestation algorithm and add a security layer (discussed in section 6.10) between the network layer and data link layer (DLL) of the network stack. This approach is transparent to routing protocols and able to detect compromised or malicious nodes; thus, it provides a framework to potentially solve both secure routing [13, 14] and the selfish node problem [5, 6] in ad hoc networks.

This dissertation differs from other related work, including NAP and NAC, TNC [15], and some TPM-related research [16, 17, 18, 19]. First, like NAP and NAC, our approach enforces client configuration verification to control network access. However, unlike those approaches, ours does not allow the verification to be forged or modified easily, and therefore is resilient to malicious users. Second, NAP and NAC only verify a node's configuration at the time it requests network access. As a result, they cannot guarantee the node's configuration is still valid after the node gains network access. However, our approach can. Third, our approach is like TCG's trusted network connection (TNC) protocol [15, 20, 21, 22, 23], which applies attestation to enterprise network access. However, our approach differs in that (1) it

addresses MITM attacks, while TNC's protocol does not, (2) it supports ad hoc networks, and (3) it considers the necessary operating system modifications. Fourth, while there are several other research projects that explore ways to enhance system security by using TPM, each of these has different focus than ours. NGSCB [16] requires new processor architecture. Both NGSCB and Terra [17] use virtual machine (VM) monitors, but neither considers in details how to use attestation to secure network access. TcgLinux [18] considers how to use attestation to secure access to virtual private networks. However, TcgLinux requires frequent attestation for detecting and responding to configuration changes. It also exposes in attestations the execution of unprivileged programs unnecessarily, reducing user privacy. Enforcer [19] is a Linux based OS that supports TPMs and uses trusted lists signed by trusted system administrator. It does not support attestation, relying instead on certificates with special semantics, and has no protection against privileged users. Finally, our research improves the security of both enterprise and ad hoc networks; none of above-listed research applies TPM technology to secure ad hoc networks.

In summary, the contributions of this dissertation research are as follows:

- We present a general and robust solution for excluding compromised or malicious nodes from a network. With the help of secure coprocessors, such nodes can be identified and excluded from the network.
- We provide several OS enhancement techniques to help enforce security policies on a network's participating nodes. These techniques include TCB prelogging, security association root-tripping, and sealing-free attestation.
- We detail a secure and efficient attestation protocol for enterprise networks. This protocol, which can thwart MITM attacks, is implemented and integrated with existing access control technologies.
- We provide AdHocSec, a security layer inserted between the network and data link layer of the network stack of ad hoc networks to enforce security policies. Several efficient algorithms (i.e., batched bound keyed attestation, distributed attestation, attested merger) are detailed.
- We describe many simulations and implementations, which can be shared with other researchers to facilitate their research.

This first chapter has provided an introduction to the dissertation research. The rest of the dissertation is organized as follows (see Figure 1.1). Chapter 2 describes the background and related work. Chapter 3 discusses the problems in existing operating systems and our solutions. Chapter 4 discusses the problems with attestations and our solutions. Chapters 5 and 6 address the security problems in enterprise and ad hoc networks, respectively, and describes how to integrate our solutions described in chapters 3 and 4 to solve these security problems. Chapter 7 reviews and summarizes the findings of this dissertation, while chapter 8 suggests future research that could extend the work of this dissertation project.

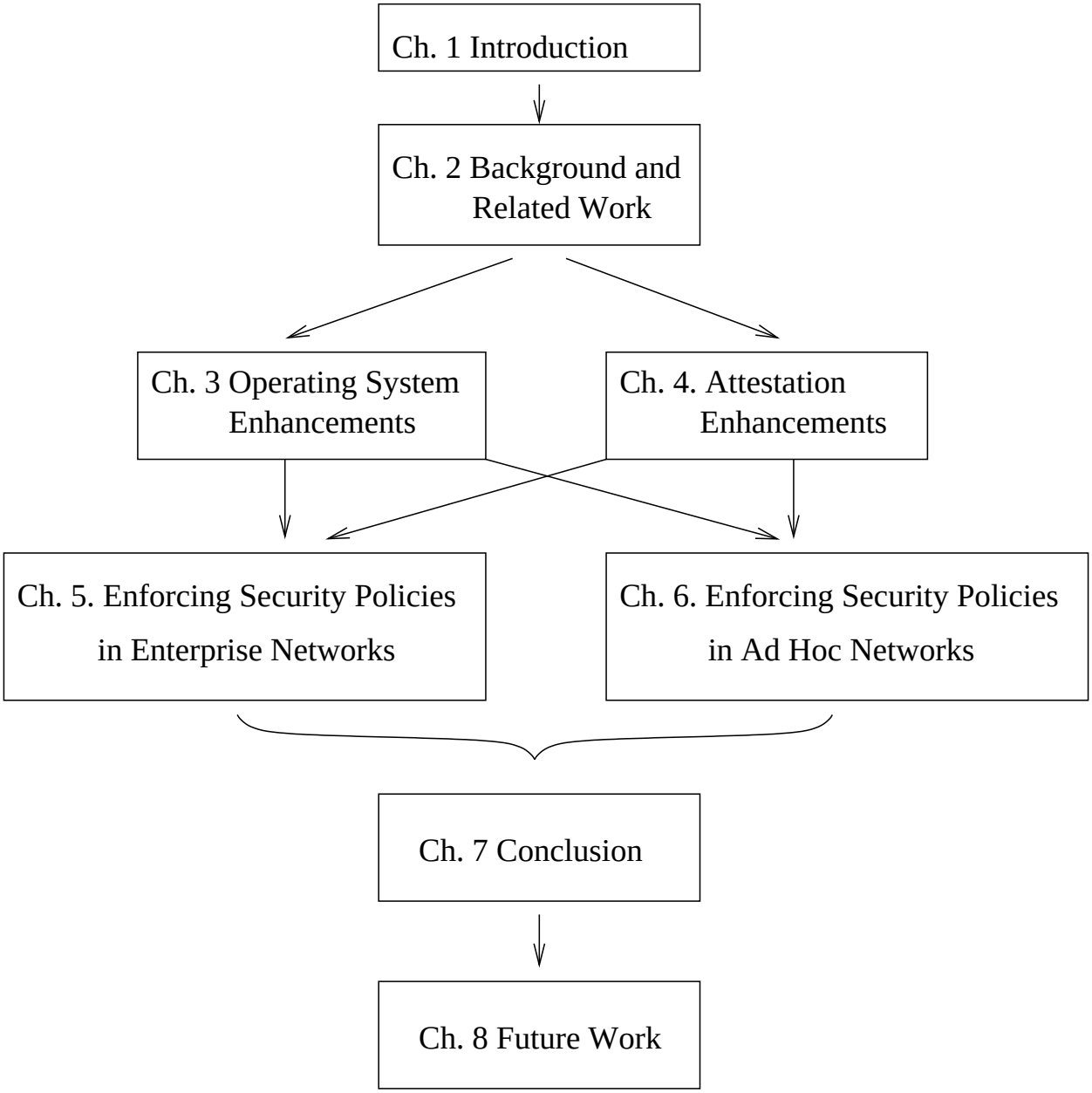


Figure 1.1: Dissertation organization

2.0 BACKGROUND AND RELATED WORK

This chapter provides background information and describes work related to this dissertation. Since the solutions this dissertation proposes for securing enterprise and ad hoc networks are based on the use of the TPM secure coprocessor, we first present an overview of the TPM secure coprocessor (section 2.1) and its related research work (section 2.2). Then, we discuss related work on the security of enterprise and ad hoc networks in sections 2.3 and 2.4, respectively.

2.1 TPM SECURE COPROCESSORS

TPM is a secure coprocessor designed by TCG [8], an industry organization that aims to improve the security of computing platforms such as personal computers (PC), personal digital assistants (PDA), or cellular phones. As shown in Figure 2.1, the structure of TPM consists of a processor, cryptographic units, and memory units. Some of the memory units are platform configuration registers (PCR), which can be used to store values such as the secure hash algorithm (SHA1) [24] hash from a main system. Cryptographic units include the hash, HMAC digest, and RSA encryption and signature components.

The TPM serves two important functions: *authenticated boot* and *attestation*. *Authentication boot* enables a computer's boot sequence to be authenticated from the time it powers on. TPM implements authenticated boot by building a chain of trust along with the booting sequence (illustrated in Figure 2.2). The current execution unit measures (i.e., takes SHA-1 hash of) the next unit before transferring control to it. The measurement is recorded in a log, and the measured results are extended to one of the TPM's PCRs. Extending data to TPM

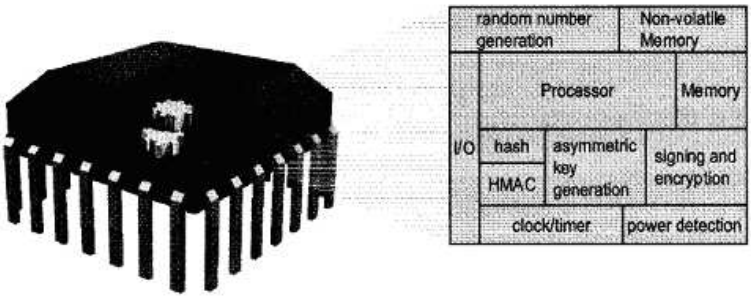


Figure 2.1: TPM architecture (Figure 3-3 in [25])

is a special way of writing data into TPM. Specifically, once the TPM receives the command to extend the data to one of its PCRs, it concatenates the current value in the PCR with the input data, takes the SHA-1 hash of the concatenation, and writes the hash result to the PCR. Therefore, the value of the PCR at any moment is equal to the compression of all the components executed since the computer was powered on. For example, before BIOS loads an OS loader, it measures the OS loader, extends the result into the TPM PCR, and appends the result into the measurement log. Any modification to code in the boot sequence results in a different measurement result and, subsequently, a different value in the TPM PCR. This difference is detected if the boot sequence is being authenticated. Since BIOS is the first component to start when a computer is turned on, it is the core root of trust measurement (CRTM). The boot sequence can be authenticated using attestation, another function provided by TPM.

Attestation is the process of vouching for the accuracy of information. In TPM attestation, a remote party can verify the hardware and software configuration of another party. The TPM generates private and public attestation identity key (AIK) pairs internally and get its AIK certificate from a third-party certifying authority. The TPM uses its private AIK only to sign quotes and never reveals it externally. The TPM attestation takes the following procedure. First, the remote party (Challenger) sends to the host a nonce, which is a cryptographic random number that is never reused. The host (Platform Agent) requests information from the TPM by issuing a quote command to its chip. TPM signs the informa-

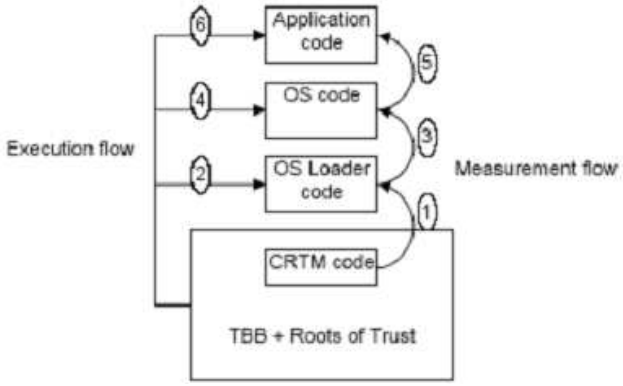


Figure 2.2: TPM chain of trust (from [26]). The chain of trust is formed as follows. The CRTM code measures OS Loader (①) first, then transfers control to OS Loader (②), OS loader measures OS (③) before it transfers control to the OS code (④). OS code measures an application (⑤) before it executes the application (⑥).

tion containing the PCR value and the nonce by using its AIK. The presence of the nonce in the quote guarantees that the quote cannot be replayed later. The host sends the corresponding AIK certificate, the quote and the measurement log to the remote party. Next, the remote party authenticates the AIK certificate using the certifying authority's public key, which it is assumed to be known out-of-band. Then, it uses the nonce and the public AIK to authenticate the quote. Finally, it uses the quote to authenticate the measurement log. This way of using TPM for attestation guarantees the information can be verified and cannot be forged.

Another function provided by the TPM is *sealing*. Sealing is the process of binding a message to a set of platform metrics. In sealing, the encrypted message is associated with a set of PCR register values and a non-migratable asymmetric key. The sealing function occurs when the data is input and the authentication credential for using a sealing key is provided. The TPM returns an encrypted object. Thus, the sealed message is bound to a specific hardware platform and software configuration; it can only be unsealed (released) when the hardware and software configuration matches that from when it was sealed. Sealing provides very strong protection for stored data by binding it with both the authentication credential

and a specific platform configuration. However, there is debate [27] on the use of TPM, as well. One of the problems caused by sealing is lock-in problem. An application, e.g. an editor, may seal to itself content created by the host owner. This can make it impossible for the host owner to switch to a different application or to access the data using a future host.

2.2 TPM-BASED SYSTEMS

Since its introduction, TPM has generated a great deal of interest among researchers. As a result, many applications and systems have been proposed. Among them are NGSCB (Next Generation Secure Computing Base) [16], Terra [17], TcgLinux [18], and Enforcer [19]. These are discussed below.

NGSCB [16], which was Microsoft's vision of the next generation operating system, is based on the TPM secure coprocessor and virtual machine (VM) technology. It consists of two isolated OS kernels on top of the computer's hardware. One is the traditional Windows OS, which supports general Windows applications. The other, called *Nexus*, has a small security kernel and manages small trustworthy applications called Nexus computing agents (NCA). Some drawbacks of NGSCB are that it needs special hardware support (e.g., CPU, memory) and may enable lock-in problems.

Terra [17] is also based on virtual machine technology. Instead of supporting only two OSs, as NGSCB does, it supports multiple "open box" and "closed box" virtual machines. An "open box" can be a general-purpose OS, while a "closed box" can be a specific trusted application (e.g., a game console, set box, cell phone). A VM is certified by a chain rooted in hardware and including each software layer's measurement up to the application. Each layer measures the higher layer block-by-block to certify it. A remote party can verify the current trustworthiness of a VM by requiring a special form of attestation (different from TCG's). However, Terra does not by itself guarantee this VM's integrity or confidentiality after attestation. If a VM hosts a general-purpose OS, techniques such as we propose in chapter 3 would help secure it.

TcgLinux [18] transforms a Linux box into a trusted client using a TPM-based Linux

Security Module. It builds a chain of trust by measuring *every* component starting from system boot to the Linux kernel. The Linux kernel then measures each executable, library, or kernel module loaded into the run-time system before they affect the system. TcgLinux also requires shells and other executables to be modified to do the same with their security-sensitive scripts and configuration files. This design makes it harder to verify that all TCB configuration files are being measured. It also exposes in attestations the execution of unprivileged programs, reducing user privacy.

Enforcer [19] was developed to secure a web server (Apache) on a Linux system. It classifies system components according to how frequently they are expected to change. Components may be long-term (e.g., the kernel), medium-term (e.g., system applications and configuration files), or dynamic (e.g., HTML web pages). The TPM and a modified LILO (LInux LOader) boot loader are used to protect and authenticate long-term components. One of these is enforcer, a Linux security module, that protects medium-term components. Enforcer uses a list with the hash value of each medium component. Enforcer verifies that the actual hash of each component on the list matches that on the list when the component is loaded. The list is signed by a trusted system administrator. Enforcer can also bind dynamic components to specific applications. Bound components are stored in an encrypted loop-back file system. This file system's keys are sealed by the TPM. Enforcer does not support attestation, relying instead on certificates with special semantics, and has no protection against privileged users (e.g., system administrator).

This dissertation differs from these operating system enhancement efforts in both goal and implementation. The goal of our research is to enhance OS such that networks can enforce required access policies on their client nodes. Our system does not require new processor architecture as NGSCB does; it prevents lock-in problem by applying sealing-free attestation confinement, which is discussed in section 3.2.2. Our system uses TCB prelogging (discussed in section 3.2.1.1), which does not require the measurement of the entire system binary as Terra does and exposure of the execution of unprivileged programs as TcgLinux does. To protect against privileged users, a problem in Enforcer, we have secure association root tripping, which is discussed in section 3.2.1.2.

2.3 ENTERPRISE NETWORK ACCESS CONTROL METHODS

In addition to other approaches to security (e.g., firewalls, IDS, AV software), enterprise networks often authenticate nodes that attempt access. This section discusses the most common access control protocols, 802.1x [10] and IPsec's [11] Internet Key Exchange (IKE) [12], as well as recently proposed approaches, including NAP [2], NAC [3], TNC [22, 23], and attestation-based VPN [28].

IEEE 802.1x [10] is a standard, widely supported protocol that can be configured to require mutual authentication between a node and a local area network (LAN) before the node is allowed to communicate through the network (beyond authentication). The participants to this protocol are the *supplicant*, i.e. the node that requests access, the *authenticator*, e.g. a switch or access point that mediates the supplicant's access to the network, and the *authentication server*, e.g. a RADIUS server that authenticates and authorizes the supplicant's access. A variety of authentication protocols can be used over 802.1x, including PEAPv2 [29]. PEAPv2 begins by creating a TLS tunnel between authentication server and supplicant, with certificate-based authentication of the former by the latter. The server then typically uses MS-CHAPv2 for password-based supplicant authentication. Finally, PEAPv2 binds the TLS and MS-CHAPv2 keys to guarantee that the respective endpoints are the same. 802.1x also enables the creation of a security association between authenticator and supplicant, with cryptographic keys for encrypting and authenticating packets sent between them. On Ethernet, such packet-level cryptography is being standardized by IEEE 802.1ae [30]; on Wi-Fi [31], packet-level cryptography is enabled by IEEE 802.11i [32].

IPsec [11] is the leading approach for VPN access in enterprise networks. IPsec's IKE provides peer authentication and session key generation. Like PEAP, IKE works in two phases. In phase 1, the communicating parties authenticate each other and establish an IKE security association (SA). The IKE SA is bidirectional and determines what cryptographic algorithms and keys the parties will use to communicate during phase 2. In phase 2, parties use the IKE SA to negotiate one or more AH (Authentication Header) or ESP (Encapsulating Security Payload) security associations between them. AH and ESP SAs are unidirectional. AH provides packet authentication, while ESP can provide packet authentication and/or

encryption. Phase 2 negotiations determine the cryptographic algorithms and keys used in each AH and ESP SA. Both 802.1x and IKE authenticate a client's identity only. They do not determine whether the client computers are compromised or configured well.

To overcome this weakness of 802.1x and IKE, the newly-initiated approaches NAP and NAC authenticate a client node's configuration when it tries to access a protected network. NAP is proposed by Microsoft as the access control platform for "Longhorn", the next version of the Windows server operating system. It provides policy enforcement components that help ensure that computers connecting to a network meet administrator-defined requirements. In the NAP platform, when a computer or NAP client connects to a network, it has to present its state of health (SoH) to network access servers (NAS). A NAS validates the client's state. If it conforms to the network's policy, the client is granted access. Otherwise, the client is isolated and directed to a restricted network. SoH includes information about the client's operating system, the anti-virus program's signature, and software versions. However, a malicious or infected client can forge the SoH. NAP verifies but do not authenticate SoHs.

NAC is similar to NAP. At the endpoint system, a Cisco trust agent is installed to collect security state information from security software solutions, such as anti-virus programs. The network access devices (e.g., switches, access points) are modified to enforce and demand host security information. When an endpoint system connects to the network, the trust agent sends the collected information to network access devices, which relay this information to policy servers for evaluation. Only endpoint computers with an approved configuration are allowed to connect. However, NAC has the same vulnerability as NAP. It does not authenticate the configuration information.

Recently, TCG has published TNC (Trusted Network Connection) specifications [15, 20, 21, 22, 23] to strengthen network security by checking the integrity and identity of access nodes before they are allowed to join a protected network. TNC defines both the components needed for platform authentication, such as Integrity Measurement Collectors and Integrity Measurement Verifiers, and communication interfaces among these components. It also provides guidelines for leveraging the TNC architecture with the existing network access technologies. However, TCG specifications are incomplete. TCG does not specify operat-

ing system modification that are necessary to ensure that attestations are consistent with current configuration. TCG also does not specify precautions that are necessary to protect attestation from MITM attacks, as discussed in section 4.1.1. Also, TNC addresses platform authentication only in conventional corporate networks. Ad hoc networks are outside TNC's scope.

Sailer et al. propose attestation-based policy enforcement for VPN access [28]. Like the VPN approach used in this dissertation research for enterprise networks, their approach applies TPM attestation. However, that solution has several shortcomings. First, TCG-specified attestation is vulnerable to MITM attacks, as shown in Section 4.1.1. Second, tcgLinux, used in [28], would be vulnerable to MITM attacks even if our designed attestation protocol BKA (discussed in section 4.2.1) were used, because it does not prevent privileged users from reading secret keys and other parameters of VPN tunnels. In contrast, security association root tripping (discussed in section 3.2.1.2) would close the VPN tunnels before such reading would be possible. Third, the VPN gateway has to verify fresh attestations of each client frequently. If attestation frequency is low, users may be able to connect an insecure node to the VPN long enough to cause harm. In contrast, security association root tripping (discussed in section 3.2.1.2) achieves security with a single attestation at the beginning of each client's session.

2.4 SECURITY IN AD HOC NETWORKS

Security is a primary concern in ad hoc networks. Therefore, many security solutions have been proposed to improve the situation. Recommendations include how to secure the routing protocols and how to enforce the nodes' cooperation in the network.

Research in secure routing protocols for ad hoc networks includes SEAD [14], Ariadne [13], SAODV [33], SRP [34], and ARAN [35]. These protocols aim to improve the security of earlier ad hoc routing protocols such as DSDV [36], DSR [37], and AODV [38]. SEAD enhances the security of DSDV by using a one-way hash chain to protect the sequence number and hop count in routing messages. This approach ensures that the malicious in-

crementing of the sequence number or decrementing of the hop count will be detected. Ariadne improves the security of the DSR protocol. It uses a source-destination secret key to authenticate routing messages and the TESLA [39] broadcast authentication protocol to authenticate intermediate nodes. TESLA requires node clock synchronization, which may be hard to achieve in an ad hoc environment. SAODV uses certificates to enhance the security of AODV by attaching signature extensions to routing messages with the goal of protecting important information (e.g. hop count, sequence number) in routing messages. SRP prevents attacks that disrupt DSR's route discovery process by using a source-destination shared secret to authenticate the routing discovery messages. Intermediate nodes maintain route request frequency counters in their routing tables to defend against the potential of overwhelming route requests from malicious nodes. ARAN is designed to secure on-demand distance vector routing protocols. It requires each node to have a certificate. Routing messages are verified by each node on the path checking the signature of the preceding node. Using public keys for message encryption and authentication makes ARAN's CPU overhead large.

One of the potential problems these secure routing protocols have in common is the difficulty of key management. Nearly all of them assume a priori trust or secret associations between nodes. Depending on a priori trust and pre-established secret associations makes it hard for nodes to join a network dynamically. For example, SRP and Ariadne both assume that pairs share secret keys for message authentication, while Ariadne and SEAD assume that each node can distribute its hash keys to the network. ARAN and SAODV depend on the use of certificates and public keys. ARAN also assumes the existence of a centrally-trusted certificate server, which is a particular challenge in a mobile ad hoc environment. Another common problem in these secure routing protocols is that they address only the security of routing messages and do not prevent node forwarding misbehavior.

Techniques for nodes' cooperation enforcement fall into three categories: (1) reputation-based mechanisms, (2) currency-based mechanisms, and (3) token-based mechanisms. Reputation-based mechanisms, such as CONFIDANT [5, 40] and CORE [6], require monitoring neighboring nodes' behavior. Monitored events are fed into the reputation system so that misbehaving nodes can be identified. A major problem with reputation-based mechanisms

is that an attacker may be able to malign and exclude legitimate nodes from the network. These systems may also be vulnerable to sybil attacks [41], where a node impersonates other nodes.

Currency-based mechanisms, such as Nuglets [42], use virtual currency to stimulate nodes' forwarding cooperation. A node needs to pay currency to intermediate nodes in order to transmit a packet to its destination. To earn enough currency for its packet transmission, the node needs to participate in packet forwarding. Currency-based mechanisms require tamper-resistant hardware that is not available commercially. They can also be problematic for nodes that require more service than they provide.

Token-based mechanisms [43] require each node to have a token in order to participate in network operations. Once a node's token expires, the node needs to renew the token with its neighboring nodes. Neighboring nodes, which have been collaboratively monitoring the behavior of the node, decide whether to issue a new token to the node based on its past behavior. However, it is a problem for a node to renew its token if the node moves quickly since the node's new neighbors do not know the node's past behavior.

It is noteworthy that previous solutions solve only particular problems in certain protocols. No general solution for securing ad hoc networks have been proposed.

This dissertation describes the use of secure coprocessors to enforce access policies in ad hoc networks. We add a secure layer between the network and data link layers of the network protocol stack. This secure layer is transparent, without modifying the routing protocols. We enforce network access policies on each participating node by using TPM attestation. This attestation ensures that only nodes with expected configurations are allowed into the network. Our solution also enforces node cooperation, since nodes with selfish configurations cannot join the network.

2.5 CHAPTER SUMMARY

In summary, this chapter provides background information about this dissertation and its related work. Since this dissertation presents solutions based on TPM secure coprocessors

to protect both enterprise and ad hoc networks, we discuss TPM and related work, as well as some additional security approaches to enterprise and ad hoc networks, respectively. Table 2.1 lists the relationships between the discussed work and this dissertation.

Table 2.1: Dissertation and Related Work Comparison

Area	Related work	Dissertation
TPM-based systems	NGSCB, TcgLinux, Enforcer, Terra	OS enhancement
Enterprise network	802.1x, IPsec/IKE	BKA integration
	NAP, NAC, TNC	
Ad hoc network	Secure routing protocols ARAN, Ariadne, SAODV, SRP, SEAD	AdHocSec
	Node cooperation enforcement CONFIDANT, Nuglets, etc.	

As we discussed earlier, many TPM-based operating system enhancement efforts, including NGSCB, TcgLinux, Enforcer, and Terra, have been proposed. However, this dissertation differs from them in both goal and implementation.

Traditional security approaches on enterprise networks, such as firewalls, IDS, and AV software, cannot effectively defend against insider attacks. Existing access control technologies authenticate only users identities. They do not authenticate node configuration. Newer approaches, such as NAP and NAC, verify but do not authenticate node configuration. TNC specifies how to use attestation for such authentication, but does not specify all the necessary operating system modifications. Furthermore, NAP, NAC, and TNC do not support ad hoc networks.

Most approaches to secure ad hoc networks only seek to address a specific problem. For example, secure routing protocols only secure routing messages, even only specific routing protocols. The node cooperation problem is also addressed by many approaches. However, these two problems actually have the same cause–malicious or compromised nodes. This dissertation provides a solution that is able to solve both of the problems by defending ad hoc networks against attacks from malicious or compromised nodes. Thus, it is able not only to secure routing protocols, but also to help enforce nodes' cooperation in ad hoc networks.

3.0 OPERATING SYSTEM ENHANCEMENTS

The goal of this dissertation is to improve network security by preventing malicious or compromised nodes from accessing a protected network. Authentication of the nodes is based on attestation enabled by TPM secure coprocessors. During and after attestation, security policies need to be enforced such that only authenticated nodes can stay in a protected network.

This chapter discusses operating system problems involved in such enforcement (section 3.1), and our proposed solutions (section 3.2). Sections 3.3 and 3.4 discuss the implementation and evaluation respectively. Section 3.5 discusses the limitation and other issues of our solutions. Related work is discussed in section 3.6. Section 3.7 summarizes this chapter.

3.1 PROBLEMS

Existing operating systems lack features that would be required to support TPM attestations and security policy enforcement. We identify two problems that need to be solved in existing operating systems.

3.1.1 HOW CAN OPERATING SYSTEMS MAINTAIN CONSISTENCY BETWEEN ATTESTATION AND ACTUAL CONFIGURATIONS?

TCG has defined the TPM software stack (TSS) [44]. This specification provides basic interfaces for application developers. However, it does not include many necessary details.

Attestation is based on a chain of trust. TCG defines this chain of trust only up to the point the operating system loads, as shown in Figure 2.2. TCG does not specify what the OS needs to do to maintain attestation consistency after boot time.

To support attestation, the operating system needs to measure the integrity of the *trusted computing base* (TCB) and record results in the measurement log and TPM's PCR. A system's TCB is the set of components whose malfunction (due to a bug or attack) would allow the system's policies to be compromised. TCB includes, typically, the BIOS (basic input/output system), MBR (master boot record), OS loader, OS kernel, loadable kernel modules, configuration files, and root-owned automatic scripts (boot time or scheduled for later). The TCB also includes any privileged applications, including daemons and interactive commands that are used by root-owned automatic scripts or are set with privileged effective user even if used by unprivileged real users (e.g., Unix setuid commands). In contrast, unprivileged applications are not part of the TCB, and can be created, configured, modified, or destroyed without compromising the system's ability to enforce policies.

After the operating system kernel gains control of the system, the TCB may change as a result of loading modules into the kernel, or executing privileged user-level code, such as daemons or interactive commands. The operating system must ensure that these changes are recorded in the measurement log and TPM's PCR, so that attestations remain consistent with the actual configuration of the system.

Therefore, solutions are needed for preserving attestation consistency in operating systems.

3.1.2 HOW CAN OPERATING SYSTEMS PREVENT ABUSE OF ATTESTATION AND SEALING FOR SOFTWARE LOCK-IN?

Software lock-in occurs when a user creates a file with some software and subsequently cannot open the file with another software.

TPM's sealing function could enable software lock-in. An application can store a file encrypted with a sealed key that the operating system makes available only to that application, and that the TPM makes available to the operating system only if the computer's TCB

is the same as at the time of storage. The operating system denies the file's key to other applications. If the user tries to reconfigure or replace the operating system to circumvent this protection, the TPM will not reveal the file's key.

Alternately, an application can use the TPM attestation function for the same purpose. The application stores file keys in a remote server. The server reveals such keys to a computer's operating system only if attestation shows that the computer's TCB is one that the server trusts, and the computer's operating system vouches to provide the keys only to the original application.

Software lock-in can benefit well-established software publishers since they can prevent their customers from switching to other software. It can also harm users because it can impede interoperation, competition, and make users unable to access their own data in other computers or using other applications [45]. In the case of archival data, e.g., such access can be necessary when the computer and application originally used become obsolete.

3.2 SOLUTIONS

This section presents our solutions to overcome the above discussed problems.

3.2.1 MAINTAINING ATTESTATION CONSISTENCY

Our approach for ensuring that a system's attestation remains consistent with the system's actual configuration is based on the observation that privileged users (e.g., root) usually do not need to log interactively into a system. In fact, enterprise users typically are unprivileged. System administrators usually are the only privileged enterprise users, and typically log interactively into a system only if the system has a problem that may require configuration change.

It is therefore useful to analyze a system's TCB as follows:

1. If no privileged user has logged interactively into a system since the latter booted, the maximum set of kernel modules that may have been loaded and privileged user-level

code that may have been executed is defined by the system's files at the time the system boots.

2. If a privileged user *has* logged interactively into the system since the latter booted, arbitrary configuration changes may have happened. Privileged users usually have the authority to modify the OS or its configuration interactively and arbitrarily. For example, in Unix, a root user can use "ifconfig" to modify network configuration or "sysctl" to change in-kernel variables, thus changing the system's configuration. These modifications may not be defined by any files present in the system at boot time, and therefore may be unpredictable. The operating system needs to catch them dynamically after boot time.

This analysis suggests the following solutions for maintaining attestation consistency: TCB prelogging and security association root tripping.

3.2.1.1 TCB PRELOGGING TCB prelogging calls for each system to maintain a configuration file, called *TCB list*, that maintains entries for the system's TCB components (e.g., privileged applications and shared libraries they use, loadable kernel modules, configuration files, device files, and system log files) and their respective digests (i.e., SHA1 hashes) and attributes (e.g., owner, group, permission bits). A flag in each entry of the TCB list indicates what information needs to be verified for each component.

At boot time, after the kernel mounts the file systems and before it starts the first user-level application (i.e., /sbin/init), the kernel:

1. Measures the TCB list file, extends the result in the TPM PCR, and appends the result to the measurement log. Therefore, from the moment the system boots, attestations reveal the system's entire TCB, including components that may not yet have executed.
2. Reads each entry in the TCB list and constructs in-memory hash tables that contain these entries. The kernel uses these in-memory hash tables for verifying each component in the TCB list when the component is accessed later.

Thereafter, whenever a file that is in the TCB list or is a root-owned script, daemon, or root setuid application is opened or executed, the kernel verifies that the file actually has the attributes and/or digest specified in the TCB list. If there is a discrepancy, the kernel: (1)

closes any security associations established with previous attestations, and (2) compresses the file's digest into the TPM and appends the new measurement to the log. Reestablishment of closed security associations will require new attestations. The new attestation will show the system's actual TCB, including components not on the TCB list.

3.2.1.2 SECURITY ASSOCIATION ROOT TRIPPING Unless the system has a bug, unprivileged users cannot cause the OS kernel, daemons, or setuid applications to compromise the system's policy enforcement. However, privileged users (e.g., root) can modify system configuration so as to violate system policy enforcement. For example, privileged users can use commands such as sysctl or ifconfig, or use a debugger for attaching and modifying privileged processes after boot time. Moreover, privileged users typically can read any information in the system. Although such reading does not modify the configuration, it may reveal secrets (e.g., security association keys) that enable violating security policies. It can be difficult or impossible to guarantee that all such modifications and accesses are captured by the system's measurement log and extended into the TPM.

To avoid attacks by privileged users, we modify the OS so that it detects when a privileged user attempts to gain interactive access to the system (e.g., by logging in or using the su command). If there are security associations established with previous attestations, the system warns the user that, if the user wishes to continue, the system will immediately drop those associations. In the case of secure Intranet access, this means that the system will destroy the keys needed for packet-level authentication and encryption, thus making access impossible. Then, if the user does continue, the system drops security associations, adds this event to the measurement log with a well-known digest, and extends the digest into the TPM. Therefore, subsequent attestations will show that a privileged user has logged in interactively since the system started. Network administrators can configure authentication servers to deny access to such systems. Such a system would then need to reboot before regaining access to the network. Rebooting erases the measurement log, as well as any non-persistent configuration changes. Persistent changes are captured by the new measurement log after reboot. These mechanisms do not preclude remote system administration or help, as long as these use daemons that the network's authentication server is configured to trust.

3.2.2 SEALING-FREE ATTESTATION CONFINEMENT

Our approach for preventing abuse of attestation and sealing for software lock-in is based on the following observations:

1. Network access protocols, such as 802.1x and IPsec's IKE, typically are implemented by trusted OS components, and cannot go through firewalls.
2. Network access protocols also typically do not require sealing.

These observations suggest the following solution: (1) the OS supports authenticated boot and attestation, but not sealing, and (2) the OS uses attestation only in conjunction with network access control protocols and does not export to other applications an interface for attestation. Thus confining attestation is advantageous because abusive applications cannot attempt to store file encryption keys in remote servers. If attestations were accessible to such applications, the remote servers could reveal the keys only to those applications, locking the user in. Since such applications cannot use attestation or sealing, lock-in is not possible.

3.3 IMPLEMENTATION

We implemented the OS solutions described in section 3.2 in FreeBSD 4.8. We first ported IBM's TPM driver for Linux [46] to the FreeBSD kernel and set the TPM device to be accessible only by privileged users. To build the chain of trust for attestation, we used a version of the GRUB [47] boot loader that was modified by IBM. GRUB includes the MBR and an OS loader. The modified MBR measures the modified OS loader and extends the result into the TPM PCR 8. Likewise, the modified OS loader measures the OS kernel and extends the result into the TPM PCR 9. The OS kernel uses TPM PCR 10 when it extends any measurement result to the TPM. The usage of TPM PCRs is outlined in Table 3.1.

We implemented the measurement log using syslogd, a daemon that runs in the background, collecting and storing log information from the kernel and user-level processes. Syslogd collects log information from the kernel by reading a special character device file

Table 3.1: TPM PCR Usage. There is a total of 16 PCRs. Usage of PCRs 0-7 is specified by the TCG PC client specific implementation specification [48]. We do not use PCRs 11-15.

PCR Index	Usage
0	CRTM, BIOS, and Host Platform Extensions
1	Host Platform Configuration
2	Option ROM Code
3	Option ROM Configuration and Data
4	IPL (Initial Program Loader) Code (usually the MBR)
5	IPL Code Configuration and Data (for use by the IPL Code)
6	State Transition and Wake Events
7	Host Platform Manufacturer Control
8	OS loader and its configuration data
9	OS kernel
10	Measurements by OS kernel

/dev/klog, which provides an interface to an in-kernel log buffer. The kernel writes to the log buffer when it needs to log some information. We configured syslogd's configuration file /etc/syslog.conf so that syslogd logs the measurements in file /var/log/tcpahash.log. When the kernel measures a file and the result has not yet been recorded in the TPM and measurement log, the kernel first extends the result to the TPM, and then writes the file's path name and measurement to the log buffer. Syslogd reads /dev/klog to retrieve this information and stores it in /var/log/tcpahash.log. When attestation is requested by a remote party, an operating system daemon sends that party the content of /var/log/tcpahash.log along with the TPM's quote and AIK certificate (note that only privileged applications (part of the TCB) can read the /var/log/tcpahash.log file and obtain a TPM quote for generating an attestation reply).

In principle, the TCB list can be manually created using any text editor. However, to facilitate its creation, we built an instrumented FreeBSD kernel. This instrumented kernel collects into the log file, at boot and run time, the absolute path name and category of the following files: (1) executable files with privileged effective user identity, (2) shared libraries linked to the above applications, (3) any files opened by the above applications (including device files), and (4) loaded kernel modules. Information about these files is collected,

respectively, by modified exec(), mmap(), open(), and kldload() system calls. The instrumented kernel is booted and subjected to a workload that requires all trusted components to be accessed. The resulting log file is then manually edited to file tcbpol.txt. Files may be added to categories 1, 2, and 4. Flags are set for checking the file attributes and secure hash of files in these categories at boot and run time. Files may be added or deleted for category 3. Some files in this category (e.g., system configuration files) have flags set for checking both file attributes and secure hash at boot and run time. Other files in this category (e.g, device files or system log files) have flags set for checking only file attributes at boot and run time. File attributes checked may include file owner, group, and permission bits.

Another program, "tcbsetup", processes tcbpol.txt and generates file tcb.pol. For each file in tcbpol.txt, tcb.pol contains an entry with the respective absolute path name, flags of file information to be checked, and value of this information.

The tcb.pol file is the TCB list that will be read by the kernel at boot time before the first user-level application in a system is started. The information in the file is used by the kernel to verify the attributes of each component when it is accessed.

To implement TCB prelogging, as discussed in section 3.2.1.1, we modified the FreeBSD kernel as follows. After mounting the root partition and before starting the first user-level process (i.e., /sbin/init), the kernel:

1. Reads the TCB list file (tcb.pol) from the hard drive, measures its digest, extends the digest into the TPM PCR 10, and appends the result in the measurement log (note that the TCB list is stored in the root partition and can be accessed only after mounting the latter).
2. Reads in memory each record from the TCB list file.
3. Creates in kernel a hash table (TCBEXEC) with 512 slots for binary executables (i.e., privileged applications, shared libraries used by privileged applications, kernel modules). This hash table is keyed by the digest, or device and inode numbers of each binary executable. Each entry in the hash table includes (from tcb.pol) an executable's absolute path name, owner, group, permission bits, digest, and (initialized to 0) time of last modification, device number, and inode number. This TCBEXEC hash table stores information about executables in the TCB list. It enables quick lookup of an executable

based on the executable's digest or its device and inode numbers.

4. Creates in kernel another hash table (TCBNONEXEC) with 512 slots for non-executables (e.g., configuration files, or device files). This hash table is keyed by the absolute path name of each non-executable. Each entry in the hash table includes (from tcb.pol) the non-executable's absolute path name, owner, group, permission bits, and (initialized to 0) time of last modification and digest (special files, e.g., device files, are assigned an empty digest value.) This hash table stores the information about non-executable components in the TCB list. It enables quick lookup of a non-executable file based on the file's absolute path name.

The kernel's exec() system call is modified such that when an executable is invoked by a privileged effective user, the kernel checks the executable's integrity by searching TCBEXEC with the executable's device and inode numbers. (1) If the kernel does not find the executable by device and inode numbers, the kernel measures the digest of the executable, and searches TCBEXEC again with the digest. If the kernel does not find the executable by digest, or the entry found has file attributes different from the executable's, the executable is not trusted. We say that the kernel then *trips security associations*: (a) extends the measured digest in TPM, (b) appends the executable's name and digest in the measurement log, and (c) closes any security associations established with previous attestations. The kernel also puts in TCBEXEC the executable's device and inode numbers and modification time. Otherwise (the kernel found entry in TCBEXEC with matching digest and file attributes), the kernel writes in the entry the executable's device and inode numbers and modification time. (2) If the kernel finds the executable's entry by device and inode numbers, the kernel compares the executable's and the entry's modification time. If they differ, the kernel measures the executable's digest and processes it as above, using the digest to search the executable's entry in TCBEXEC. Otherwise (the modification time matches), the kernel simply proceeds. The kernel uses the modification time to measure an executable at most once, when it is first accessed, provided that the executable is not modified.

The kernel's mmap() system call and elf_load_file() are modified the same as exec() system call to check the integrity of shared libraries and the ELF loader, respectively. Kldload() system call is also modified for kernel module loading. Actually, a subroutine, measure_file(),

is implemented to do the function as described above. We only need to insert measure_file() in exec(), mmap(), elf_load_file(), and kldload().

The kernel's open() system call is modified such that when a file in the TCB list is opened, the kernel checks the file's integrity. After resolving the file's path name, the kernel searches TCBNOEXEC with the file's name. If the kernel does not find the file's entry in TCBNOEXEC (meaning the file is not part of the TCB list), it simple proceeds. If the kernel finds an entry, the kernel checks the flag in the entry to determine whether to check the file's attributes and/or digest. If the flag indicates so, the kernel compares the entry's and the file's attributes. If the flag requests comparison of the digest, the kernel compares the file's and the entry's modification time. If they differ (e.g., on the file's first access), the kernel measures the file's digest, writes the file's modification time in the entry, and considers that there is a discrepancy; otherwise, the kernel considers that digests match. If the kernel found an attribute or digest discrepancy, the kernel trips security associations and stores the file's attributes and digest into the entry. (Since the digest of a device file cannot be measured, the kernel can use a predefined digest value for the purpose of recording it in the TPM and the measurement log.)

To support security association root tripping, as discussed in section 3.2.1.2, we defined new kernel variables: *attested* and *rootlogin*. *Attested* indicates whether there are security associations established with earlier attestations. *Rootlogin* records whether root has logged interactively into the system since the system booted.

We modified two important applications, *su* and *login*, such that, when root attempts to log into the system, if *attested* is true, the application warns the user that security associations will be dropped. If the user continues, the application performs a system call that: (1) changes *rootlogin* to true, (2) extends a predefined dummy value to the TPM and measurement log, and (3) closes security associations established with previous attestations.

To close security associations established with previous attestations, the kernel sends a non-fatal signal (e.g., SIGHUP) to the daemons that established them (e.g., xsupplicant for 802.1x, racoon for IPsec's IKE). These daemons are modified to respond to the signal (by defining the signal handler using sigaction()), so as to destroy the security associations with remote parties and the keys used for the communication, and notify the kernel by making

a system call after they are finished. In the case of IPsec's IKE, racoon first sends to the kernel the SADB_FLUSH message through the PF_KEY socket interface [49], causing the kernel to delete all entries in its key table for IPsec. Racoon then deletes the IKE security associations and notifies the kernel. In the case of 802.1x, xsupplicant sends the access point a EAP-logoff message [10] (forcing the access point to transition the client's port to an unauthorized state), resets the secret key shared between the client and the access point, and notifies the kernel. In order to send such signal to these daemons, the kernel needs to know these daemons' process identities. The kernel records them when these daemons set the in-kernel variable *attested* after security associations are established.

3.4 EVALUATION

We evaluated our enhancements on an IBM ThinkPad T30 computer with 1.8 GHz Pentium 4 CPU, 256 MB RAM, TPM version 1.1b, TPM-aware BIOS, and built-in 802.11b interface. We report the average and standard deviation (σ) of six instances of each measurement. The master boot record and GRUB boot loader were modified for measuring digests and compressing them into the TPM, as specified by TCG.

We first used the instrumented kernel to collect components that should be included in the TCB list for a representative configuration. The result is listed in Table 3.2. Each entry in the TCB list is a record including the component's absolute path name, verification flags, attributes, and digest. With each entry taking 1,052 bytes, the size of the TCB list for this configuration is 275,624 bytes (uncompressed).

We then installed the enhanced OS on the evaluation platform. We measured a total boot time of 18.05 s ($\sigma = 0.18$) before and 18.49 s ($\sigma = 0.04$) after our modifications. Although TCB prelogging and file digest measurements do impose overheads, they are dominated by other boot costs. During system operation, each digest file measurement is cached and is not repeated as long as the file is not modified. Because TCB components change infrequently, file digest measurements can be expected to have little impact on steady-state performance.

Security association root-tripping affects only certain commands (i.e., login and su) and

Table 3.2: TCB list's component categories and number of entries for a representative configuration. Verification flags are denoted: 'D' for digest, 'O' for owner, 'G' for group, and 'P' for permission bits.

Category	Number of entries	Examples	Verification flags
Privileged applications	114	/usr/sbin/syslogd	DOGP
Shared libraries	27	/usr/lib/libc.so.4	DOGP
Kernel modules	2	/modules/ipfw.ko	DOGP
Scripts	14	/etc/rc	DOGP
Configuration files	85	/etc/rc.conf	DOGP
Device files	18	/dev/tpm	OGP
Log files	22	/var/log/cron	OGP
Total	262		

only when used by privileged users. We measured the extra time that a privileged user has to wait to interactively log in a system if security associations were established with previous attestations. For IPsec, the extra time is 1.018 s ($\sigma = 0.001$). For 802.1x, the extra time is 9.48 ms ($\sigma = 0.54$). Tripping IPsec security associations takes longer than 802.1x associations. However, the overhead is acceptable in both cases.

To evaluate the impact of our kernel modifications to exec(), mmap(), elf_load_file(), kld_load(), and open(), we measured the time needed for compiling the kernel without and with the modifications. The compilation time is 384.25 s ($\sigma = 1.41$) or 388.15 s ($\sigma = 0.48$), respectively. Our modifications add only 3.90 seconds (about 1%) to the kernel compilation time.

3.5 DISCUSSION

TCB prelogging does not necessarily log TCB components in the actual sequence in which they are accessed. The access order between TCB components may in some cases be relevant to security. Our scheme guarantees the order of execution of the chain BIOS→MBR→OS loader→OS kernel. Thereafter, if two TCB components need to be accessed in a particular

order to guarantee a certain configuration (e.g., Linux security module (LSM) [50] installation), such accesses need to be established by a script or configuration file that is itself part of the TCB.

3.6 RELATED WORK

There are several approaches to verify a node's configuration. Our TCB lists are similar to what Tripwire uses to verify host integrity [51]. However, unlike Tripwire, we enable the host's integrity to be verified remotely.

TcgLinux [18] is Linux-based OS with TPM support. Because tcgLinux does not have a TCB list, it logs and compresses into the TPM digests of *all* files that are executed, and requires shells and other programs to be modified to do the same with their security-sensitive scripts and configuration files. This design makes it harder to verify that all TCB configuration files are being measured. It also unnecessarily exposes in attestations the execution of unprivileged programs, reducing user privacy. Moreover, since the system's configuration changes dynamically whenever a file is executed, a remote server in a network needs to verify fresh attestations of each client frequently to maintain attestation consistency. If attestation frequency is low, users may be able to connect an insecure node to the network long enough to cause harm. In contrast, our solutions do not expose unprivileged programs in attestation (TCB prelogging) and can maintain attestation consistency without requiring frequent attestation (security association root-tripping). TcgLinux has mechanisms to prevent privileged users from making system modifications that might not be detected by attestation. It does allow, however, any modifications that would be detected by attestation. TcgLinux may allow compromising a system's confidentiality because it does not prevent privileged users from reading secret keys that may be established between the system and a remote server. In contrast, security association root tripping would close the security association before such reading would be possible.

Terra [17] uses virtual machine (VM) technology to support multiple "closed-box" and "open-box" VMs. The closed-box VM's integrity can be verified by a remote server. However,

the attestation of the closed-box is based on certificates, which is different from the TPM attestation approach in this dissertation. In addition, Terra requires the measurement of the entire closed-box VM's binary image for attestation. In contrast, TCB prelogging only measures the trusted components in a system.

Schoen [52, 53] proposed the idea of "owner override" to prevent the use of secure co-processors for software lock-in or digital rights management (DRM). Owner override allows the owner of a computer to force the computer's TPM to sign arbitrary quotes, regardless of the computer's actual configuration. Such fictitious quotes would prevent use of attestation for software lock-in. However, it does not prevent use of sealing for the same purpose. It also prevents secure verification of a node's configuration before a node is admitted into a network, as proposed in this dissertation.

3.7 CHAPTER SUMMARY

In this chapter, we discuss OS problems that need to be solved to reach the goals of this dissertation, and provide solutions to them. The two OS problems are: (1) how to maintain attestation consistency, and (2) how to prevent software lock-in. System integrity, privacy, and confidentiality need to be considered for maintaining attestation consistency. We propose TCB prelogging and security association root tripping for solving this problem. TCB prelogging maintains attestation consistency and client system privacy by logging and extending into the TPM expected measurements of all TCB components at boot time. Therefore, from the start, attestations reveal a system's entire expected TCB. Actual measurements are deferred until the actual first use of a TCB component. If there is a discrepancy, the OS drops any security associations established with previous attestations. Unprivileged applications are not measured and exposed to remote parties. Security association root tripping likewise drops such security associations if a privileged user logs interactively into a system. This technique prevents compromise of the system's integrity or confidentiality by privileged users. To prevent software lock-in, we propose sealing-free attestation confinement. We evaluated our solutions by implementing them in FreeBSD. Experiment results show that

our solutions cause little overhead.

4.0 ATTESTATION ENHANCEMENTS

This chapter describes problems in TCG's attestation specifications and our proposed solutions. It also discusses related work.

4.1 PROBLEMS

TCG's attestation specifications can be considered incomplete because they do not address the security and performance problems discussed in the following subsections:

4.1.1 HOW CAN ATTESTATION BE PROTECTED FROM MITM ATTACKS?

TCG's specifications are sufficient for protecting attestation against forgery and replay attacks. Forgery is not possible because attestation is signed by a secure coprocessor using a private key that is generated inside the secure coprocessor, is used only for signing attestations, and is never revealed outside the secure coprocessor. A trusted authority certifies that the corresponding public key belongs to the secure coprocessor and that the latter is securely bound to the respective platform. An attestation cannot be replayed because it contains a nonce provided by the verifier. A responder can use an attestation only when the verifier uses the respective nonce (i.e., once).

TCG's specifications leave room, however, for MITM attacks. In a MITM attack between communicating parties P_1 and P_2, the attacker impersonates P_1 to P_2 and P_2 to P_1. The attacker then relays packets between P_1 and P_2. Even if the packets are authenticated and encrypted, the attacker can read, modify, insert, or delete any packets between P_1 and P_2.

It is commonly assumed that a MITM attacker is an adversary both to P_1 and P_2. However, this is not necessary the case. In an attack against attestation, the attacker would have a configuration that is unacceptable to some verifier V. The attacker would seek to pass as its own the configuration of a responder R, which is acceptable to V. R might have colluded with or be under the control of the attacker (in such cases, we say that R is "colluding" with the attacker).

If V and R do not use an authenticated channel to communicate, the MITM attack against attestation is trivial, even without R's cooperation. The attacker simply relays V's nonce to R, and relays R's response to V.

If V and R do use an authenticated channel to communicate, a MITM attack against attestation may still be possible. Suppose, for example, that V and R establish a TLS channel, and then V obtains R's attestation via this channel. A MITM attacker can establish a first TLS channel with V, a second TLS channel with R, and then relay attestation messages between V and R, as shown in Figure 4.1. The attacker can establish the first channel because TLS usually authenticates only the server (V) to the client. If the first TLS channel requires client authentication, the attacker can use its own identity certificate, or that of a colluding R. The attacker can also establish the second TLS channel because a colluding R can (1) be directed to connect to the attacker, instead of V, or (2) be configured to trust a phony certificate authority that associates the attacker's public key with V's identity. Even if R is not colluding, the attack may still be possible if R's software allows users to override certificate errors (as most Web browsers and SSH clients do). Users typically click through warnings and continue the channel establishment despite certificate verification failures [54].

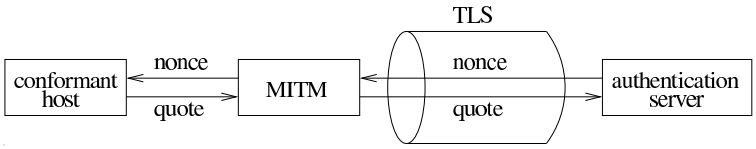


Figure 4.1: TCG defined attestation and MITM attack

4.1.2 HOW CAN ATTESTATION BE USED WITH REAL WORLD PROTOCOLS WITHOUT INTRODUCING EXCESSIVE LATENCY?

An important design goal of TCG-specified secure coprocessors is to have low cost, so as to enable widespread deployment. Commercially available TPMs indeed have lost cost (about $4 a piece), but often correspondingly slow performance. In our tests on IBM's ThinkPad T30 computers, for example, attestation takes about 0.71 seconds.

For responders that engage in attestation only sporadically (e.g., client computers), such attestation latencies may be acceptable. If a responder needs to respond to many verifiers at once, however, latencies could rise excessively. Such a scenario could occur, for example, if the responder is a sensor or node in an ad-hoc or peer-to-peer network. Multiple (say, N) clients or other nodes, respectively, could send attestation requests to the same responder at the same time. If the responder processes such requests sequentially, attestation latency could be as high as N*0.71 seconds. As N increases, latencies could become unacceptable.

4.2 SOLUTIONS

Our solutions to the above problems are Bound Keyed Attestation (BKA) and Batched Bound Keyed Attestation (BBKA), described in the following subsections.

4.2.1 BKA

Bound Keyed Attestation (BKA) is a novel form of attestation that prevents MITM attacks by binding attestation with a secret key shared by attestation endpoints. Binding proves to the verifier that the attestation responder knows a secret that the verifier shares only with the responder. This secret is not the verifier's nonce, because the verifier does not identify who it sends a nonce to.

Suppose that the attestation endpoints have a shared secret key K_s, and the attestation verifier sends a nonce N_{orig} to the responder. The responder binds K_s and N_{orig} by concatenating them first, then calculating the SHA1 hash of the concatenation. The responder uses

the result as the nonce $N_{binding}$ for TPM quote:

$$N_{binding} = SHA1(N_{orig}|K_s) \tag{4.1}$$

When the verifier receives the attestation from the responder, it verifies that the TPM quote includes $N_{binding}$. Because of the one-way property of SHA1, only a responder that knows K_s can generate a quote containing $N_{binding}$. (Note also that an attacker cannot obtain K_s from N_{orig} and $N_{binding}$.)

If there is a secure channel established between the endpoints before attestation is invoked, the endpoints already have a shared secret key. BKA binds this key to attestation. On the other hand, if there is no secure channel between the endpoints before attestation, BKA uses a Diffie-Hellman key exchange [55] to generate a shared key, and binds this key with attestation.

4.2.1.1 BKA WITH SECURE CHANNEL

In many situations, a secure channel is already established between the endpoints before attestation is invoked. For example, if attestation is to be integrated with protocols such as IPsec's IKE, TLS, or PEAPv2, these protocols can establish a secure channel C before attestation. We assume that C is cryptographically secured with a dynamically-generated secret key K_C that is not revealed to users. C can have state either *attested* or *not-attested*; *not-attested* is the default. System policies can prevent all communication except for attestation while the channel is not-attested.

All BKA messages are transmitted using the established secure channel with which the attestation will be bound, as illustrated in Figure 4.2. BKA messages can be of type: *request* (M_i), *response* (M_r) or *response-request* (M_{rr}), *success* (M_s) or *success response* (M_{sr}), or *error* (M_e).

BKA messages are exchanged as follows. After the secure channel C is set up, the attestation verifier A sends a *BKA request* message M_i containing a nonce N_A to the attestation responder B. The responder then computes the *verifier's attestation nonce* $n_A = \text{SHA1}(N_A|K_T)$ and gets from its TPM the quote Q_B containing n_A. Finally, the responder sends to the verifier a *BKA response message* M_r containing Q_B, B's AIK certificate C_B, and measurement log L_B. Alternatively, if the responder also wishes to obtain

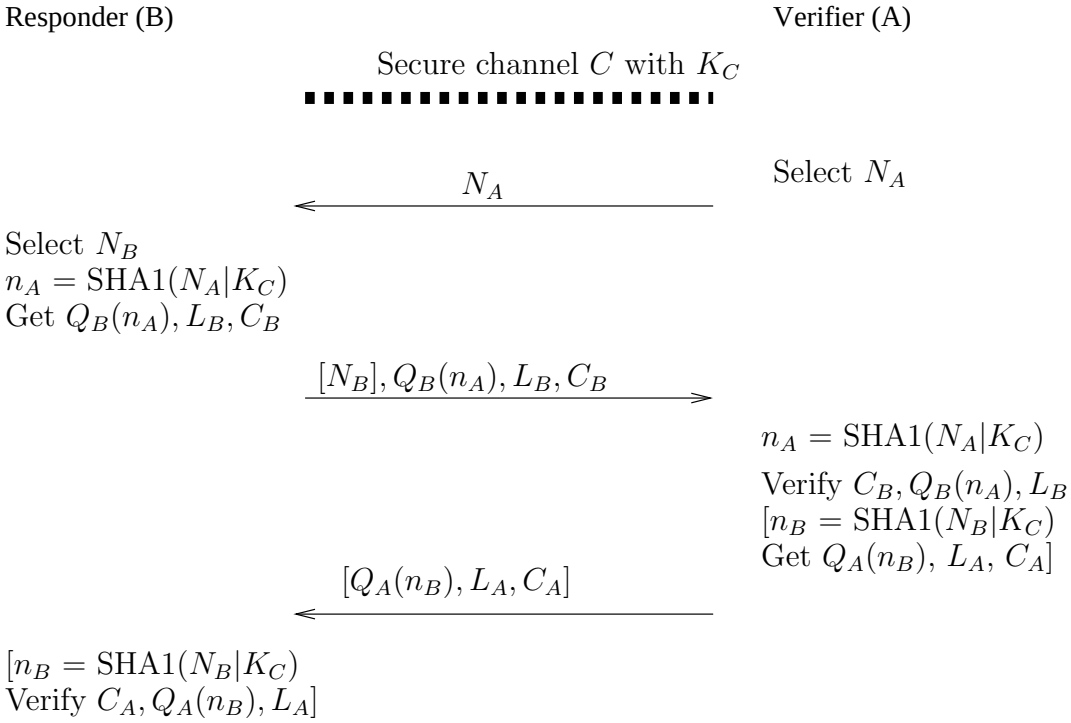


Figure 4.2: Bound Keyed Attestation (BKA) with secure channel thwarts MITM attacks by guaranteeing that the endpoints of the communication channel (e.g., TLS, PEAPv2) and attestation are the same. Items between brackets are needed only for mutual attestation.

the verifier's attestation, the responder sends to the verifier a *BKA response-request* message M_{rr} containing the same fields as M_r, plus a fresh nonce N_B.

If the verifier receives M_r, it processes the message as follows. First, the verifier computes the bound nonce $n_A = \text{SHA1}(N_A|K_T)$ used by the responder for the TPM quote. The verifier authenticates C_B using the corresponding certificate authority's public key (known securely out-of-band), authenticates the responder's quote Q_B using n_A and C_B, and authenticates the responder's measurement log L_B using Q_B. If any of the authentications fails, or if the verifier does not recognize a measurement in the responder's log L_B as that of a trusted software component, the verifier sends a *BKA error* message to the responder and returns failure. Otherwise, the verifier then sends to the responder a *BKA success* message M_s, transitions T's state to *attested*, and returns success.

The verifier's processing of M_{rr} is similar to that of M_r, with the following exceptions: (1) The verifier also computes the *responder's attestation nonce* $n_B = \text{SHA1}(N_B|K_T)$ and obtains from the verifier's TPM its quote Q_A containing n_B. (2) Instead of M_s, the verifier sends to the responder a *BKA success-response message* M_{sr} containing Q_A, A's AIK certificate C_A and measurement log L_A.

After the responder processes M_s, if the attestation response is successful, it changes its state to *attested*. Otherwise, its state remains as *not-attested*. Processing of M_{sr} is similar, with the following exceptions: (1) The responder also authenticates C_A, Q_A, and L_A; (2) If all authentications succeed and the responder identifies every measurement in the verifier's log as that of a trusted software component, then the responder transitions its state to *attested*; otherwise, its state remains *not-attested*.

4.2.1.2 BKA WITHOUT SECURE CHANNEL

When attestation is needed but there is no secure channel between attestation endpoints, BKA includes a Diffie-Hellman (DH) key exchange [55]. Attestation endpoints derive a shared secret key and securely bind it to attestation, as shown in Figure 4.3.

BKA uses two publicly-known numbers: a prime number q and an integer α that is a primitive root of q. These numbers do not need to change. The *attestation verifier* A picks two random numbers: a nonce N_A, which is never reused, and another integer $0 \leq X_A < q$,

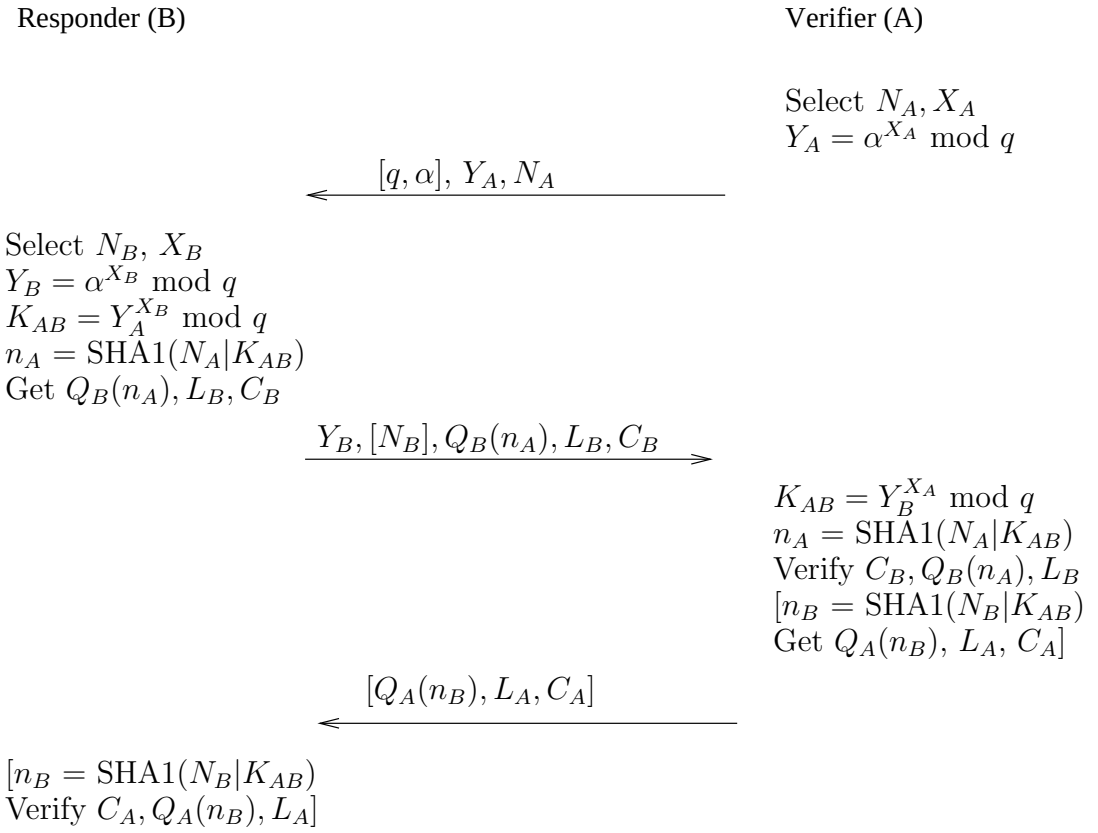


Figure 4.3: Bound Keyed Attestation (BKA) without secure channel includes a Diffie-Hellman (DH) key exchange. It thwarts MITM attacks by guaranteeing that DH and attestation endpoints are the same. Items between brackets are needed only for mutual attestation.

which may be reused. The verifier computes its public key $Y_A = \alpha^{X_A} \bmod q$. The verifier then sends a *BKA request* message M_i containing q, α, Y_A, and N_A to the *attestation responder* B. If q and α are known a priori by the responder (for example, if verifier and responder use a previously agreed upon Oakley group, as defined for IPsec's IKE [12]), these parameters do not need to be included in the BKA request message.

The attestation responder B then picks a random integer $0 \leq X_B < q$, which may be reused, and computes its public key $Y_B = \alpha^{X_B} \bmod q$. The responder computes the *attestation shared secret* as $K_{AB} = Y_A^{X_B} \bmod q$. The responder also computes the *verifier's attestation nonce* $n_A = \text{SHA1}(N_A|K_{AB})$. The responder then gets from its TPM its quote Q_B containing n_A. Finally, the responder sends to the verifier a *BKA response* message M_r containing Y_B, Q_B, B's AIK certificate C_B, and measurement log L_B. Alternatively, if the responder also wishes to obtain the verifier's attestation, the responder sends a *BKA response-request* message M_{rr} containing the same fields as M_r, plus a fresh nonce N_B.

The verifier processes M_r as follows. First, it computes the attestation shared secret as $K_{AB} = Y_B^{X_A} \bmod q$, as well as the bound nonce $n_A = SHA1(N_A|K_{AB})$ used by the responder for the TPM quote. The verifier authenticates C_B using the respective certificate authority's public key (known securely out-of-band), authenticates the responder's quote Q_B using n_A and C_B, and authenticates the responder's measurement log using Q_B. If any of the authentications fail, or if the verifier does not recognize a measurement in the responder's log L_B as that of a trusted software component, the verifier sends a *BKA error* message to the responder and returns failure. Otherwise, the verifier sends a *BKA success* message M_s to the responder, transitions T's state to *attested*, and returns success.

Processing of M_{rr} by the verifier is similar to that of M_r, except that (1) the verifier also computes the *responder's attestation nonce* $n_B = \text{SHA1}(N_B|K_{AB})$ and gets from the verifier's TPM its quote Q_A containing n_B, and (2) instead of M_s, the verifier sends to the responder a *BKA success-response* message M_{sr} containing Q_A, A's AIK certificate C_A, and measurement log L_A.

The responder processes M_s as follows. If the attestation response is successful, it changes its state to *attested*. Otherwise, its state remains *not-attested*. Processing of M_{sr} is similar, with a couple of exceptions: (1) The responder also authenticates C_A, Q_A, and L_A; (2) If all

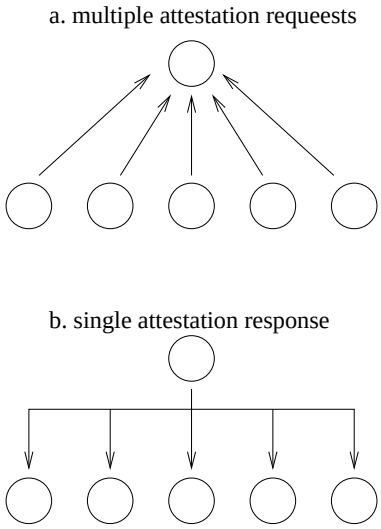


Figure 4.4: Batched Bound Keyed Attestation allows a node to respond to multiple attestation requests with a single quote and a single message.

authentications succeed, and the responder identifies all measurements in the verifier's log as those of trusted software components, then the responder transitions its state to *attested*; otherwise, its state remains *not-attested*.

Nonces N_A and N_B guarantee quote freshness and allow bound keyed attestations to be obtained frequently, without burdening the processors each time with the expensive computation of Y_A, Y_B, and K_{AB}, if the nodes' Diffie-Hellman keys are reused. Performance can be improved by caching the authentication of C_A and C_B and the values of common intermediate expressions that change infrequently. The attestation nonce n_x is the one-way function of both the nonce N_x and the attestation shared key K_{AB}. Therefore, the quotes are bound to the attestation shared key.

4.2.2 BBKA

BKA uses a TPM quote to reply to a single BKA request. However, the time necessary for a TPM to produce a quote may be significant: We measured an average of 0.71 s on an IBM ThinkPad T30 computer running Linux. If a single node receives many requests at

once (e.g., in an ad hoc or peer-to-peer network), BKA could introduce unacceptable delays, because it requires as many quotes as there are requests. This section introduces Batched Bound Keyed Attestation (BBKA), a novel form of attestation that uses a same quote to respond securely to multiple attestation requests.

This section assumes absence of pre-established secure channels among nodes. Therefore, BBKA includes DH key exchange. However, BBKA can use instead keys derived from pre-established secure channels, similarly to BKA in section 4.2.1.

To amortize quote delays, BBKA *batches* attestation requests that are received from multiple verifiers while a responder's TPM is busy. BBKA then uses a single quote to respond securely to all batched requests. This approach is shown in Figure 4.4. BBKA incorporates a DH key exchange between the responder and each verifier and securely binds the resulting shared secrets to the quote used in the response. The DH shared secrets enable confidential and authenticated communication between the responder and each verifier after attestation. The binding of these secrets with the attestation response's quote prevents man-in-the-middle attacks, in which a node with a malicious configuration attempts to circumvent attestation by relaying attestation requests and replies between nodes with mutually-trusted configurations (see section 4.1.1).

In BBKA, the attestation request received from a verifier i contains not only i's nonce, N_i, but also i's DH public key Y_i and DH group g_i. Y_i is computed from i's DH private key X_i using equation $Y_i = \alpha^{X_i} \bmod q$. Group g_i identifies the DH modulus q and primitive root α used by i, as do Oakley's groups in IKE [12]. Assuming that all n verifiers and responder use the same DH group and that the responder's DH private and public keys are respectively X_r and $Y_r = \alpha^{X_r} \bmod q$, the responder calculates each verifier's shared secret S_i and *bound nonce* B_i respectively as:

$$S_i = Y_i^{X_r} \bmod q \tag{4.2}$$

$$B_i = HMAC(N_i, S_i) \tag{4.3}$$

where the HMAC algorithm[56] uses the SHA-1 security hash [24]. The responder then

computes the *batch nonce* N_b:

$$N_b = SHA1(B_1|...|B_n) \tag{4.4}$$

where "|" denotes concatenation. The responder requests from its TPM a quote containing N_b.

BBKA's response contains not only the responder's quote Q_r with N_b, attestation identity certificate C_r, and measurement log L_r, but also n and $B_1, ..., B_n$. If the responder has not already sent an attestation request to the verifier (e.g., for mutual attestation), then the response also contains Y_r. A verifier i should calculate its shared secret S_i as:

$$S_i = Y_r^{X_i} \bmod q, \tag{4.5}$$

obtain its bound nonce B_i from N_i and S_i, verify that B_i is in $B_1, ..., B_n$, compute from the latter N_b, verify Q_r using N_b and C_r, and use Q_r to authenticate L_r.

Note that it is not feasible for intruders or verifiers other than i to obtain X_i or S_i from the BBKA requests or responses. BBKA also assumes that verifiers and responders have configurations that do not reveal secrets such as X_i and S_i to users (this assumption can be verified by the respective attestations). Otherwise, an intruder could obtain shared secrets from a node with trusted configuration, then use these secrets for impersonating the latter with a maliciously configured node.

To save computation, verifiers and responders can reuse DH keys and cache intermediate results of DH computations. However, to prevent replay attacks, they should use different nonces in each attestation request. Since attestation latency is largely dominated by the quote operation, BBKA latency with multiple verifiers is not much greater than latency for responding to a single attestation request.

4.3 IMPLEMENTATION

We implemented BKA and BBKA in a library called libbkap.a. Differences between BKA and BBKA are: (1) BBKA calculates a nonce based on information from multiple verifiers, and (2) BBKA attestation response includes more information. Implementation in a library enables, e.g., some applications to link only BKA and not BBKA. Header files define the BKA and BBKA application programming interfaces (APIs) and data structures. A parameter specifies whether BKA or BBKA is being used with or without a pre-established secure channel between endpoints. The BKA and BBKA library handles message processing and maintains attestation state. The implementation hides BKA and BBKA details, simplifying integration with other protocols.

4.4 RELATED WORK

Garfinkel et al. [17] have previously discussed MITM attacks against attestation. Their proposed solution is to embed attestation into SSL's handshake protocol. The verifier must complete the handshake's key exchange only after verifying the responder's attestation. Details needed for implementing this proposal and evaluating its security against MITM attacks are missing, however. It appears that MITM attackers would be able to obtain the verifier's nonce and the responder's attestation response in clear text. This cannot be prevented simply by encrypting the attestation response with the verifier's public key, because a colluding or inattentive responder would use the attacker's public key instead of the verifier's. Encrypting the attestation response with keys derived from the SSL exchange would also be ineffective, since a MITM attacker would know these keys. If, unlike our solution, attestation responses do not cryptographically depend on the SSL key exchange, it seems that a MITM attacker would be able to insert the responder's attestation response in the attacker's SSL handshake with the verifier. Other potential advantages of our solution include its ready applicability to other protocols (in addition to SSL) and scalability to applications requiring multiple simultaneous attestations (section 4.2.2).

Burger et al. [57] proposed vTPM, a system that enables trusted computing in virtual machines (VMs) by virtualizing the TPM. vTPM makes it appear to each VM that the VM has access to its own private TPM, even when multiple VMs share the same physical TPM in a system. The virtualized TPM can securely attest the integrity of the respective VM without interference from other VMs. vTPM can be combined with BKA or BBKA in VM-based systems.

4.5 CHAPTER SUMMARY

In this chapter, we discussed attestation security and performance issues that are not addressed by TCG specifications, and described our solutions, BKA and BBKA. BKA and BBKA thwart MITM attacks by cryptographically binding attestations with secret keys shared by attestation endpoints. They can be used with or without pre-established secure channel between endpoints. BBKA improves attestation performance by using a same quote to reply to multiple attestation requests. We implemented BKA and BBKA as a library that can be easily integrated with other protocols. The design and implementation of our solutions have the advantages of simplicity, ready applicability, and scalability.

The performance impacts of BKA and BBKA on enterprise and ad hoc networks are discussed in chapters 5 and 6, respectively.

5.0 ENFORCING SECURITY POLICIES IN ENTERPRISE NETWORKS

In this chapter, we discuss how we apply the proposed TPM-based security solution on enterprise networks. We first analyze the existing access control protocols used in enterprise networks (section 5.1) and their insufficiency (section 5.2). Then we describe the design and implementation of our solution in section 5.3 and 5.4. Section 5.5 provides an evaluation of our solution. Section 5.6 summarizes this chapter.

5.1 ACCESS CONTROL PROTOCOLS FOR ENTERPRISE NETWORKS

Enterprise processes are increasingly integrated via computer networks. Data transiting such networks may include, e.g., employee records, customer orders, warehouse inventories, and business plans. It is imperative to protect the integrity and confidentiality of this data. Network availability also needs to be protected, as lack of communication can often bring enterprise processes to a halt.

Ethernet is the most common enterprise network. A wired local area network (LAN), Ethernet is confined by an enterprise's walls. Thus, ordinary physical protection of enterprise buildings (e.g., checking employees' badges at entrances) also controls access to these networks.

Physical access control may be insufficient, however. Enterprises often have within their premises part-time, outsourced, contractors, or partners' employees or visitors. Once inside an enterprise's buildings, such people can plug their mobile computers into the enterprise's Ethernet ports.

IEEE 802.1x [10] is a relatively recent protocol that enables an enterprise to authenti-

cate a computer's user before authorizing communication through the port the computer is attached to. IEEE 802.1x can generate session keys for authenticating and encrypting an authorized computer's packets. Packet-level security prevents an attacker from piggybacking on a legitimate user's session, e.g. by connecting a hub to that user's port. IEEE 802.1ae [30] is a draft standard that will enable such packet-level security in Ethernet networks.

Wi-Fi [31] is a wireless LAN that is used in an increasing number of enterprises. It can be installed much more cheaply than Ethernet, but is not confined by an enterprise's walls. Attackers may be able to access a company's Wi-Fi network, e.g., from the company's parking lot. Wi-Fi's original security scheme, Wired Equivalent Privacy (WEP) [58], has been shown to be deeply flawed. IEEE 802.11i [32] is a newer standard that greatly improves Wi-Fi security. It uses 802.1x for authenticating a user before authorizing the user's computer to access the network. It also derives session keys for strong packet-level authentication and encryption.

Three parties participate in 802.1x: a *supplicant* (user's computer), an *authenticator* (e.g., Ethernet switch or Wi-Fi access point), and an *authentication server* (e.g., RADIUS server). By default, the authenticator's ports are unauthorized. (In Wi-Fi, "port" is interpreted as an association between a mobile computer and an access point.) If a port is unauthorized, the authenticator allows the port's supplicant to communicate only with the authentication server. When the latter sends the authenticator a message authorizing the supplicant's access, the authorizer switches the port's state to authorized. With the port in that state, the supplicant can communicate with the rest of the network.

IEEE 802.1x supports EAP (Extensible Authentication Protocol) [59]. EAP is a framework over which many authentication methods can be implemented. A frequent choice is Protected EAP (PEAP) [29]. PEAP has two phases. In the first phase, the network's authentication server proves its identity to the user's computer, and a TLS [9] channel is established between the server and the user's computer. In the second phase, the user proves her identity to the server. The first phase's TLS channel protects the integrity and confidentiality of the second phase's packets. Many user authentication methods may be used in the second phase. MS-CHAPv2 is a popular choice because it is password-based and therefore easy to deploy. Like many other tunneled protocols, PEAP has been found to be vulnerable

to MITM attacks. PEAP version 2 (PEAPv2) [29] adds a cryptographic binding to the end of PEAP's second phase, guaranteeing that both phases' endpoints are the same.

PEAP arguably achieves a better tradeoff between usability and security than do its earlier alternatives, EAP-MD5 and EAP-TLS. Like PEAP, EAP-MD5 can conveniently use passwords for user authentication. However, EAP-MD5 does not authenticate the network to the user's computer, and therefore is vulnerable to MITM attacks. EAP-TLS [60] mutually authenticates user and network, overcoming this vulnerability. However, EAP-TLS requires issuing, verifying, and possibly revoking user certificates, and therefore is harder to deploy that is PEAP. TTLS [61] is another alternative to PEAP. Differences between both protocols are slight. PEAP is more common because it is included in Microsoft operating systems, while TTLS is not.

LANs, such as Ethernet and Wi-Fi, are insufficient for interconnecting an enterprise's multiple locations. LANs are also insufficient for connecting telecommuters and traveling personnel ("road warriors") to an enterprise's network. Enterprises typically use virtual private networks (VPNs) for such connections. An enterprise's VPN gateways at different locations, or a VPN gateway and a road warrior, mutually authenticate each other and establish a secure channel for communicating over the Internet. The channel can be configured such that all packets sent between endpoints are authenticated and encrypted.

The protocol suite most commonly used in VPNs is IPsec [11]. IPsec secures packets between endpoints. IPsec endpoints may not coincide with the ultimate source and destination of secured packets. For example, an IPsec connection between a road warrior and a VPN gateway may secure packets sent between the road warrior and an enterprise mail server.

IPsec's Internet Key Exchange (IKE) [12] protocol provides mutual authentication between endpoints and enables negotiation of AH (Authenticated Header) and ESP (Encapsulating Security Payload) security associations between them. The AH and ESP protocols secure other IPsec traffic. AH provides packet authentication, while ESP can provide packet encryption and/or authentication.

IKE comprises two phases. In Phase 1, IPsec endpoints (1) mutually authenticate each other and (2) agree on packet authentication and encryption algorithms and keys for an IKE security association between the endpoints. In Phase 2, the endpoints use the IKE

security association to negotiate algorithms and keys for an AH or ESP security association between the endpoints. Unlike IKE, AH and ESP security associations are unidirectional. Two modes are defined for Phase 1, main mode and aggressive mode. The quick mode is defined for Phase 2. Supported authentication methods include digital signature, two variants of public-key encryption, and pre-shared key. A benefit of IKE's phased approach is that mutual authentication, an expensive operation, needs to be performed only once. A same IKE security association can be used to negotiate multiple AH and ESP security associations.

5.2 INSUFFICIENCY OF EXISTING ACCESS CONTROL PROTOCOLS

Existing access control protocols, such as PEAP and IKE, attempt to prevent access by untrusted users. An enterprise can use these protocols, e.g., to grant access only to full-time employees.

A trusted user can, however, unwittingly use a compromised computer or insecure configuration. For example, the user can have a mobile computer that became infected while outside the enterprise. Similarly, a telecommuter can use a computer that is shared with other household members and that is infected by them. In such cases, authenticating the user is insufficient.

These vulnerabilities can be eliminated by having the 802.1x authentication server or VPN gateway obtain an attestation of the user's computer. The attestation allows the network to authenticate the computer's configuration, before authorizing its access.

IEEE 802.1x authentication servers and VPN gateways typically are physically well protected and are more carefully maintained than are user computers. Therefore, in most cases, it will be considered sufficient for the user's computer to authenticate the identity, but not the configuration of the authentication server or gateway.

The following two sections describe how we integrated PEAP and IKE with attestation, so as to authenticate not only the user's identity, but also the configuration of the user's computer.

5.3 INTEGRATING PEAP WITH ATTESTATION

5.3.1 DESIGN

We integrate PEAP with attestation by adding BKA to the end of PEAP's second phase. The TLS channel established in PEAP's first phase protects BKA messages. To guarantee that TLS and BKA endpoints are the same, we (1) use TLS's pseudo-random function to derive an extra 128-bit key from the TLS channel's master secret (as defined in section 3.5 of [60]), and (2) cryptographically bind this key to attestation, using the method described in Section 4.2.1. BKA could be performed before or after PEAP's second phase. The latter choice offers greater resilience against denial-of-service attacks: The authentication server verifies a user computer's configuration (an expensive computation) only after verifying the user's identity (a comparatively inexpensive computation).

No modifications are necessary in PEAP packet formats for BKA integration. PEAP's second phase is inherently extensible to accommodate future user authentication methods. PEAP user authentication messages are carried in generic TLV (Type-Length-Value) fields inside TLS records. Fig. 5.1 illustrates the organization of a TLV field. A TLV field contains subfields M (flags whether the sender considers support for that TLV type mandatory), R (reserved), Type, Length, and Value. The Value subfield's organization and length is determined by the TLV's Type and Length subfields, respectively. We define BKA messages as a new TLV type. The latest PEAP draft defines TLV types 1-21; we used type 22 for BKA.

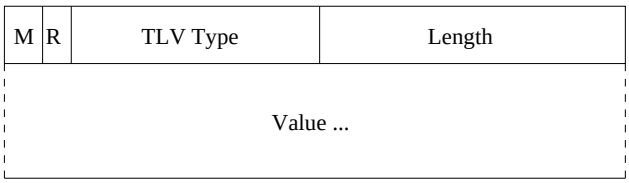


Figure 5.1: PEAPv2 TLV format

BKA responses can be several thousand bytes long, requiring fragmentation. PEAPv2 defines a general mechanism for message fragmentation and reassembly. The suggested

maximum message length is 64 KB, which is likely to be much more than necessary for BKA messages.

Only localized changes in PEAP's processing are necessary for BKA integration. After PEAPv2's cryptographic binding of PEAP's first and second phases, and before the authentication server sends the authenticator the access authorization packet, the server checks whether it is configured to require attestation from the user's computer. If so, the server sends the supplicant a BKA request in a mandatory PEAP TLV field with type BKA. If the supplicant is configured to accept such a request, the supplicant computes its BKA reply and sends the latter to the server in another mandatory PEAP TLV field with type BKA. The server verifies the reply and, if the latter is acceptable, sends the authenticator the access authorization packet.

These modifications are backward-compatible. If a modified supplicant requests access to an unmodified server, PEAP processing is unmodified. The server authorizes access without attestation, and the client accepts it. On the other hand, if an unmodified supplicant requests access to a server that requires attestation, PEAP processing fails: The supplicant is unable to process the mandatory TLV field with the BKA request. Failure is the correct outcome in this case because the supplicant cannot provide the attestation that the server requires.

5.3.2 IMPLEMENTATION

We implemented an attestation-capable supplicant by modifying Xsupplicant version 1.0. Xsupplicant is an open-source 802.1x supplicant that supports many EAP methods, including EAP-MD5, EAP-TLS, TTLS, and PEAP. It also supports MSCHAPv2 for user authentication in TTLS and PEAP. However, Xsupplicant implements PEAP version 0, as do current Microsoft operating systems. This version lacks two features of PEAPv2 that are necessary for BKA, namely cryptographic binding and fragmentation. Each Xsupplicant EAP method is implemented as a module.

We implemented a new Xsupplicant EAP module, PEAPc. We defined its EAP type to be 0x08 (previously unused). It differs from the previous Xsupplicant PEAP module in three ways. First, it has PEAPv2's cryptographic binding at the end of the second phase.

Second, it supports the additional BKA handshake after the cryptographic binding. Third, it supports fragmentation and reassembly for BKA messages as follows: (1) it defines two fields: M and fragment offset, indicating if another fragment follows this one and this fragment's offset in the message, respectively; (2) If the fragment is the last fragment of a message, M is set to 0. Other than that, M is set to 1; (3) Once a receiver receives all the fragments of a message, it reassembles the fragments into message.

PEAPc depends on our TPM and BKA libraries, libtcpa.a and libbka.a, respectively. We installed these libraries in /usr/local/lib/ and the respective header files in /usr/local/include/tcpa/, and /usr/local/include/bka/. Xsupplicant is a root setuid application and therefore is included in the TCB list of systems where it is installed.

We implemented a corresponding attestation-capable authentication server by modifying open-source FreeRADIUS version 1.0.0 prerelease 3. FreeRADIUS supports proxying, failover, and load balancing. It also supports many EAP methods, including EAP-MD5, EAP-TLS, TTLS, and PEAP.

We implemented a new FreeRADIUS module, rlm_eap_peapc, and defined its EAP type to be 0x08. It differs from the previous FreeRADIUS rlm_eap_peap module just as Xsupplicant's PEAPc differs from PEAP. The new module can be configured to require or not require attestation by defining a configuration option in radius.conf. The list of acceptable configurations is configured as follows: We defined a configuration file trustedapp.conf which includes absolute names of all the trusted applications and their corresponding SHA1 hash value. This configuration file is read and parsed to build a hash table with the name of a trusted application as key and its digest as value while FreeRADIUS daemon starts. When a server receives BKA response from a client, it checks the digest of each component of the client configuration against its record by looking up the trusted hash table. If the received digest matches the record, that component is regarded trusted, otherwise, it is believed malicious.

In our implementation, we added two options: *acceptatt* and *serverlist* in Xsupplicant's configuration file *xsupplicant.conf*. Acceptatt has true or false value, indicating if the Xsupplicant accepts attestation or not. Serverlist maintains a list of servers that the Xsupplicant only accept attestation from. If a server is not in the server list, the client does not accept

its attestation request during the authentication process. These two options give a client the flexibility to control whether or not it is being attested by a server.

Meanwhile, our implemented client and server have backward compatibility with legacy Xsupplicant and authentication server implementation. If an attestation capable Xsupplicant client communicates with a legacy authentication server, the server will not request attestation, therefore, the attestation phase can be skipped. Although the client is attestation capable, it behaves based on the server's request. If a legacy Xsupplicant communicates with a server with attestation implementation, it will fail since it cannot respond to the server's attestation request. The server will time out and fail the client's authentication since it cannot receive attestation from the client.

5.4 INTEGRATING IKE WITH ATTESTATION

5.4.1 DESIGN

We integrate IKE with attestation by adding BKA in between IKE's first and second phases. This makes BKA messages be protected by the security channel established during phase one, and guarantees either or both endpoints (depending on configuration) are attested before the two end points negotiate IPsec security association. To guarantee that IKE and BKA endpoints are the same, we (1) use IKE's pseudo-random function (as defined in section 5 of [12]) to derive an extra 128-bit key from the channel's master secret, and (2) cryptographically bind this key to attestation, using the method described in section 4.2.1.

The integration of IKE with BKA requires no change to IKE's packet format. In IKE, messages exchanged between the endpoints are formatted as defined in ISAKMP [62]. The exchange type and next payload in the ISAKMP header 5.2 dictates the type and payload sequence of the message being transferred. To accommodate BKA messages, we define a new exchange type ISAKMP_ETYPE_ATT with number 240, and a new payload type ISAKMP_NPTYPE_ATT with number 128. These two numbers are not currently being used in the ISKAMP specification. The format of attestation payload in ISAKMP consists

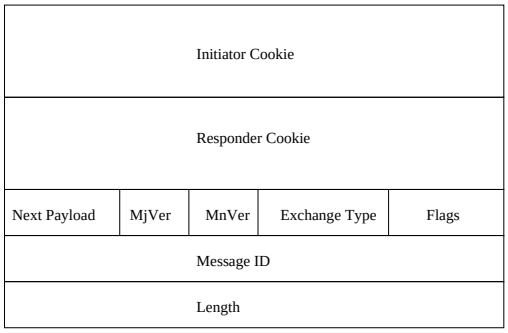


Figure 5.2: ISAKMP Header Format. Next Payload indicates the type of the first payload in the message. Exchange Type indicates the type of exchange being used. Flags indicates specific options that are set for ISAKMP exchange.

of the generic payload header and attestation message, which is illustrated in Figure 5.3.

Either party in IKE can initiate attestation by sending attestation request after the first phase finishes. The initiator can (1) encapsulate an attestation request message in the attestation payload as illustrated in 5.3, (2) encrypt the attestation payload by using the key materials negotiated in phase one, and (3) construct an ISAKMP packet with exchange-type assigned to ISAKMP_ETYPE_ATT and the next payload assigned to ISAKMP_NPTYPE_ATT. Once the packet is formed, the initiator sends it to its peer. When the peer receives an ISAKMP packet, it can (1) identify the attestation exchange by checking if the exchange type is ISAKMP_ETYPE_ATT, (2) checks that the next payload in the ISAKMP is ISAKMP_NPTYPE_ATT, (3) uses the key materials negotiated in phase one to decrypt the

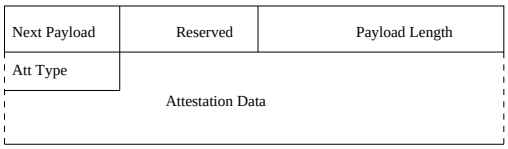


Figure 5.3: ISAKMP Attestation Payload Format

payload and extracts the attestation payload. Finally, it calls the BKA protocol handler to process the attestation payload.

For backward compatibility and configuration flexibility, we add configuration options in the IKE configuration file so that either party can enable/disable attestation or initiate/response attestation request.

Attestation message fragmentation or retransmission is handled by IKE or its lower level protocol. Since IKE is an UDP protocol. If the attestation message is too large, the UDP protocol will fragment/defragment the message for transmission.

5.4.2 IMPLEMENTATION

We implemented IKE with BKA integration using *racoon* [63] (version 20040818a), an open-source implementation of the IKE [12] protocol. Like the integration of 802.1x with BKA, we installed the TPM and BKA libraries, libtcpa.a and libbka.a in /usr/local/lib/ and the respective header files in /usr/local/include/tcpa/, and /usr/local/include/bka/.

We defined all the required functions associated with attestation in a file isakmp_att.c, and inserted attestation hooks after phase one of IKE finishes. This implementation tried to avoid changing existing files as much as possible.

We added two configuration options: *acceptatt* and *initatt* in racoon's configuration file *racoon.conf*. Each of them has true or false value, indicating if an entity accepts attestation from a peer and if it should initiate attestation to its peer after IKE phase one is finished, respectively. An entity can set the acceptatt to true to accept attestation, otherwise, set it to false. Furthermore, it can initiate the attestation after IKE phase one is finished by setting initatt to true, otherwise set it to false. In the typical enterprise VPN environment, A VPN gateway can set these options true to enable attestation and solicit attestation request to its clients. A client can be configured to accept or not accept attestation. In the latter case, the client will not be able establish a VPN connection to the VPN gateway that requests client attestation. The combination of these two options enables an entity the flexibility of controlling attestation.

These options also enable the IKE implementation with attestation to be compatible to

IKE client or server without attestation implementation. If a legacy IKE client connects to a VPN gateway with attestation request, the client will not respond to attestation request from the VPN gateway, causing the attestation request time out, thus it cannot establish the IPsec security channel. If a client with attestation implementation connects to a legacy VPN gateway, the connection can succeed since the VPN gateway does not request attestation after IKE phase one finishes and directly establish the IPsec security association.

These two options can be added as two options for each IKE peer's configuration, or apply to a group of peers. Therefore, an IKE entity can specify the options to accept attestation to each individual peer or a group.

5.5 EXPERIMENT RESULT

This section describes how we evaluated the integration of BKA with 802.1x and IKE, and the experiment results. We emphasize the impact of BKA on the performance of OS, 802.1x, and IKE.

5.5.1 IMPACT ON 802.1X PERFORMANCE

We integrated PEAPv2/802.1x with BKA on both the FreeRADIUS authentication server [64] and the Xsupplicant [65]. In order to estimate a comparison with NAP, we alternatively integrated PEAPv2/802.1x with a NAP-like LOG protocol. Both in BKA and in LOG, the supplicant sends its list of software components and configuration to the authentication server, who may or may not approve it. However, unlike the BKA's list, LOG's list can be forged by the supplicant. We installed FreeRADIUS on a Dell Dimension 4550 computer with 2.4 GHz Pentium 4 CPU, 256 MB RAM, and unmodified FreeBSD 4.10. We installed Xsupplicant on the aforementioned IBM T30 computer. As authenticator, we used a Cisco 1100 802.11b access point connected to the authentication server via Fast Ethernet.

Table 5.1 shows that the time it takes for the supplicant to connect to the network increased from about 65 ms to 95 ms with LOG or to 798 ms with BKA. To understand the

cause of BKA's large overhead, we measured the time it takes for the supplicant to obtain a quote from its TPM, and found it to be 0.71 s. Therefore, most of the latency added by BKA is due to the TPM. However, even with a low-cost, slow TPM, such as the one we used, the connection latency is acceptable.

In order to evaluate the impact of LOG and BKA on the authentication server's CPU usage, we instrumented the OS's idle loop and interrupt vector. The modified OS uses the CPU's built-in cycle counter to measure the CPU's idle time, excluding interrupts. We also instrumented FreeRADIUS to measure the total time necessary for the authentication and authorization of supplicant access. Table 5.1 shows the time it takes the CPU to process a single supplicant, as well as the projected throughput. From an estimated 2995 supplicants per minute, LOG and BKA reduced the projected throughput to 2800 and 1842 supplicants per minute, respectively. BKA's large overhead is due to its use of authentication of certificates and quotes, as well as repeated applications of SHA-1. In spite of its substantial overhead, BKA's projected throughput may be acceptable for medium- and small-sized networks. Note that, in our scheme, a supplicant imposes no load on the authentication server once authorization is accessed. In larger networks, the load can be distributed simply through the use of multiple low-cost authentication servers.

5.5.2 IMPACT ON IKE PERFORMANCE

We ran our modified racoon on the aforementioned IBM T30 laptop running FreeBSD 4.8 as a VPN client and on the Dell desktop described above running FreeBSD 4.10 as a VPN gateway. We measured the latency of each phase of the original IKE and the modified IKE, as well as the CPU busy time during the entire IKE process. Table 5.2 shows the experiment results using the IKE main mode and the RSA signature authentication method.

As shown in Table 5.2, VPN client attestation increased latency from about 153 ms to about 867 ms. This increase is due mostly to the client's TPM quote time (710 ms). For VPN access, the increased total latency can be considered acceptable. Table 5.2 also shows both the VPN gateway's CPU busy time for a single client's connection request and the projected throughput. BKA reduces projected throughput from roughly 500 to about 445

Table 5.1: 802.1x/PEAP Authentication Latency and Projected Throughput (standard deviations represented between brackets: [σ]). The delay between TLS and MS-CHAPv2 is the time interval when a server sends MS-CHAPv2 request to a client and when it receives the first MS-CHAPv2 response from the client

Step	PEAPv2	PEAPv2 + LOG	PEAPv2 + BKA
TLS	38.32 ms [1.11]	38.49 ms [1.00]	38.60 ms [0.99]
Delay	7.42 ms [0.27]	7.66 ms [0.14]	8.27 ms [0.90]
MS-CHAPv2	12.78 ms [0.75]	12.04 ms [0.16]	12.27 ms [0.55]
Binding TLS/MS-CHAPv2	6.74 ms [0.42]	6.95 ms [0.76]	6.71 ms [0.14]
LOG		30.42 ms [1.08]	
BKA			732.78 ms [2.81]
Total	65.26 ms [1.54]	95.55 ms [2.12]	798.04 ms [3.62]
CPU busy	20.03 ms [1.12]	21.42 ms [1.38]	32.57 ms [1.54]
Projected throughput	2995 supp/min	2800 supp/min	1842 supp/min

Table 5.2: IKE Authentication Latency and Projected Throughput With or Without BKA for VPN client attestation (standard deviations represented between brackets: [σ])

Step	Unmodified IKE	IKE + BKA
Phase 1	110.57 ms [2.23]	112.77 ms [0.85]
BKA		719.48 ms [2.94]
Delay	14.03 ms [0.35]	8.48 ms [0.06]
Phase 2	28.48 ms [0.10]	28.48 ms [0.12]
Total	153.08 ms [2.23]	869.20 ms [2.78]
CPU busy	120.09 ms [0.69]	134.73 ms [0.84]
Projected throughput	500 clients/min	445 clients/min

clients per minute. Although BKA's overhead is significant, the projected throughput is acceptable for many applications. Multiple VPN gateways can be used for higher loads.

5.6 CHAPTER SUMMARY

In this chapter, we discuss our approach to improving the security of enterprise networks. Our solution is hardware-based, relying on the attestation function of TPM secure coprocessors.

The security problems addressed by our work in this chapter are how to prevent malicious or compromised clients from accessing enterprise networks and how to enforce network security policies on the nodes that have gained access. Current access control protocols for enterprise networks only authenticate users, but not clients' configurations. Existing security solutions (e.g., firewalls, IDS, and anti-virus software) do not address these problems, and newer approaches, such as NAP and NAC are weak against malicious users. The TNC group does define some interfaces of TCG for applying attestation in enterprise networks. However, TNC does not specify how OS should be improved to support attestation.

We made two contributions to the existing scholarship: (1) we propose the idea of

hardware-based, operating system enhanced approach to authenticate and enforce client's configuration, (2) we identify a means of integrating the BKA with existing network access control technologies. We have implemented our strategies and conducted experiments to evaluate our approach. Our findings show that the addition of BKA attestation to enterprise network access control costs acceptable overhead and enforces very secure access control that protects enterprise networks.

6.0 ENFORCING SECURITY POLICIES IN AD HOC NETWORKS

In this chapter, we discuss how we apply the proposed TPM-based security solution on ad hoc networks. Following the same procedure as in chapter 5, we first introduce the existing network protocols used in ad hoc networks (section 6.1), and analyze their insufficiency (section 6.2). Then, after describing the assumptions (section 6.4) and notations (section 6.5) used in this chapter, we describe the design and implementation of our solution (in sections 6.3, 6.7, 6.8, 6.9, and 6.10). Lastly, we evaluate and analyze our proposed solution (in sections 6.11, 6.12, and 6.13). Finally, we conclude this chapter (in section 6.14).

6.1 EXISTING AD HOC NETWORK PROTOCOLS

Due to the unique ad hoc feature, routing protocol is fundamental to ad hoc networks. Since there is no infrastructure support, each node has to participate packet forwarding to enable normal communication among nodes. For this reason, each node has to function as a router, maintaining routing table, such that it knows to which neighbor it needs to forward an outgoing packet. Since wired network have been mature, most ad hoc routing protocols leverage the ones in wired networks.

Most routing protocols for wired network fall in one of the two categories: *link-state* and *distance vector*. Link-state routing protocol, e.g., open shortest patch forward (OSPF) [66], requires each router maintain a global view of the network topology and the cost of each link. The best route is the one with lowest cost. Distance vector routing protocols, e.g., routing information protocol (RIP) [67], require each router maintain for each destination a next hop and the distance (number of hops) to the destination through that interface.

Ad hoc routing protocols leverage from the ones in wired networks. Examples of ad hoc routing protocols designed at the early stage are DSDV (Destination sequenced distance vector) [36], AODV (ad hoc on demand distance vector) [38], and DSR (Dynamic source routing) [37]. DSDV leverages distance vector routing protocols used in wired network. It removes some features that are not required in ad hoc networks and add other features (e.g., destination sequence number). DSDV requires periodic routing message exchange to keep the routing table current, thus, consuming network bandwidth. To avoid period routing information update, AODV doesn't require periodic routing message exchange, but rather establishing a route path to a destination when it is needed. DSR is a dynamic source routing protocol. It differs from AODV in that the routing path is contained in a data packet once a route is dynamically established.

However, these early stage ad hoc routing protocols didn't take security into consideration, therefore, are weak against many attacks. There is no access control and protocol enforcement, thus allowing attackers get into the networks easily. Some of the attacks include eavesdropping, packet insertion or replay, man-in-the-middle (MITM) attacks, wormhole [68] attacks, and black-hole attacks.

To mitigate attacks on ad hoc networks, some security routing protocols and node cooperation enforcement techniques have been proposed. These techniques typically protect the networks by encrypting or authenticating messages based on cryptographic techniques. Some of them use symmetric algorithms, which requires nodes in the network to have pairwise or group-wide shared secrets. Others use asymmetric algorithms, requiring each node to have a public key and certificate authorized by a known certifying authority (CA). Examples of the secure routing protocols are SEAD [14], Ariadne [13], SAODV [33], SRP [34], and ARAN [35]. Example of protocols that enforce node cooperation are CONFIDANT [5, 40], CORE [6]), and Nuglets [42]. However, these protocols still have problems.

6.2 INSUFFICIENCY OF EXISTING PROTOCOLS

There are several problems with these secure routing protocols. First, almost all of them assume the existence of a priori trust or secret associations among nodes, but they do not account for this well. SRP and Ariadne assume pairs share secret keys for message authentication. Ariadne and SEAD assume each node can securely distribute its hash keys to the network. ARAN and SAODV depend on the use of certificates, with ARAN assuming the existence of a centrally-trusted certificate server, which is hard to achieve in an ad hoc environment. Second, they all offer weak protection against compromised nodes. Once a node is compromised and the shared secrets are revealed, the secure protocols are not secure any more. Third, each proposed secure routing protocol only improves the security of a certain *original* routing protocol. For example, Ariadne secures DSR [37], but not AODV [38] or DSDV [36]. SAODV secures AODV, but not DSDV or DSR. Thus, the application of security in ad hoc networks is very complicated. Fourth, these secure routing protocols only address the security of messages. They fail to address node forwarding misbehavior, which isolates them from the techniques used for cooperation enforcement.

Existing solutions for enforcing node cooperation in a network have problems as well. CONFIDANT [5, 40] and CORE [6]) are reputation-based systems, which depend on nodes monitoring of their neighboring nodes' behavior. Monitored events are fed into a reputation system to detect misbehaving nodes. A major problem with reputation-based mechanisms is that an attacker may be able to malign and thus exclude legitimate nodes from the network. Like CONFIDENT and CORE, Nuglets [42] is a currency-based system. It uses virtual currency to stimulate node forwarding cooperation. However, to determine how much currency should be used in a transmission is a problem. Other mechanisms [43] are token-based and require each node to have a token to participate in network operations. Once a node's token expires, it must renew the token from its neighboring nodes. However, token renewal can be a problem due to node's movement. Likewise, key management is a problem for cooperation enforcement efforts. CONFIDANT assumes that nodes are authenticated and that no node can pretend to be another; thus, it is weak against node spoofing.

Furthermore, a common problem with these existing solutions is that no single solution

addresses both secure routing and node selfishness problems. Most proposed secure routing protocols fail to address the node selfishness problem at all; instead, they assume these solutions already exist. At the same time, approaches to solve the node selfishness problem do not address either secure routing or key management.

6.3 SOLUTION AND CHALLENGES

Our solution is based on the observation that many attacks on ad hoc networks are due to the presence of malicious or compromised nodes, machines with malicious software that are participating in the network. Attackers usually control such nodes with the goal of attacking networks. For example, since the communication between a source and destination pair depends on other intermediate nodes' routing and forwarding, compromised or malicious nodes could disrupt routing by using techniques such as worm-hole [68] or black-hole. These cause problems such as routing loops, routing detours, and network partitions. Selfish nodes, a kind of passive malicious node, may selectively not forward packets on behalf of other nodes for the purpose of saving battery power. If a network is able to prevent malicious or compromised nodes from joining the network, it is able to head off many such routing-based attacks and node selfishness problems.

Therefore, we propose to secure ad hoc networks by enforcing the attestation of the node's configuration to prevent malicious or compromised nodes from participating in the network. Only nodes that pass the configuration attestation can join the protected ad hoc network. Furthermore, once a node is allowed to access the network, its enhanced OS is able to enforce the required network security policies such that any node that initially gains access, but subsequently violates these policies will be excluded from the network promptly. As we can see, the requirement for OS support in ad hoc networks can be the same as for enterprise networks. However, the particular features of ad hoc network require the design of different attestation algorithms and alternative ways of integrating attestation with the ad hoc network protocols.

The challenges of our approach are many. First, there is the problem that no dedicated

server exists in an ad hoc network. To address this, each node should be able to authenticate the other nodes. Therefore, two nodes need to mutually attest each other if they want to communicate. However, attesting every node in a network one-by-one involves significant overhead. To solve this problem, we enable mutually-attested nodes to form a group with a group identity. The nodes in the group do not have to attest each other directly. Instead, two nodes can trust each other as long as there is an attested chain of trust among the nodes. In addition, the nodes in the group can derive a shared secret, which can be used to protect the messages among the nodes in the group. As an example of the approach discussed above, if nodes A and B attest each other successfully and nodes B and C attest each other successfully, then nodes A and C do not need to attest each other. They may form a trusted group with a group identity and a shared secret. Communication among the nodes in the trusted group can be authenticated by using the group secret. If more than one trusted group forms in a network, the groups can merge if the groups attest each other successfully; this can be done if representative nodes from each group attest each other.

This chapter contributes several algorithms and protocols. We develop a distributed attestation (DA) algorithm, which triggers attestation in the entire network by broadcasting attestation requests. Therefore, all the nodes can participate in attestation in parallel and network-wide attestation can be finished in a short time. To help groups in a network converge, we designed the attested merger (AM) algorithm. Finally, we propose to add the AdHocSec layer, a secure layer that integrates our BBKA, DA, and AM attestation algorithms, between the network layer and data link layer to protect the network layer protocols and user-level applications.

The rest of this chapter discuss our design in detail. Section 6.6 discusses promiscuous unicast, a new message transmission method in ad hoc networks to facilitate attestation and reduce message collisions. Section 6.7 and 6.8 discuss distributed attestation (DA) and attested merger (AM) algorithms, the two protocols used to organize all the nodes in an ad hoc network to do attestation in parallel. Section 6.9 looks at the problem of message fragmentation, while section 6.10 presents the method we use to integrate our designed algorithms in ad hoc nodes. Section 6.11 discusses our evaluation of this work and presents our experiment results. Sections 6.12 and 6.13 analyze the performance and security of our

approach, respectively. Lastly, section 6.14 summarizes this chapter.

6.4 ASSUMPTION

Our work relies on the following assumptions:

- Since our solution is based on the TPM secure coprocessor, we assume it is present on each node. This requirement does not present a particular hardship. Nowadays, many computer vendors ship their computers with TPM coprocessors embedded.

- Each node has at least one attestation identity key (AIK) certificate issued by a known certificate authority. All nodes can use the same certifying authority or different nodes can use different authorities, but all authorities involved with the nodes should be known by all nodes either directly or in a chain of certificate trust.

- Each node is able to verify the configurations of the other nodes, including the OS version, system configuration files, and trusted software applications. Therefore, each node should have a repository of integrity measurements of these components. It should be noted that we do not require all nodes to have the same configuration, but the configuration of any node should be a subset of the repository, in order to pass attestation. In other words, we do not require every node to have the same, immutable configuration; a node with multiple safe configurations is supported.

The latter two assumptions may not be easily achieved in wild ad hoc networks since heterogeneous computer devices exist in MANET [4]. However, our proposed solution can be applied immediately in administrated environments, such as ad hoc networks formed during corporate meetings, by a designed emergency response team, or in a military environment. In addition, our solution can be applied in more environments if certificate administration is solved and software integrity measurements can be published and maintained carefully.

6.5 NOTATION

We use the following notations in describing our algorithms and protocols in this chapter:

- R: a root node that initiates distributed attestation.
- N_A, N_B: node entities A and B in the network.
- P_N: node N's parent.
- $AltP_N$: node N's alternative parent.
- S: sender.
- N: node N or its identity.
- KID_N: node N's group key identity.
- S_N: the current state of Node N.
- PRI_N: node N's group priority.
- $MrgPeer_N$: node N's merge peer when N is engaged in a merge.
- T_{AttReq}: the attestation request timer used in DA. It is the time during which a node waits for attestation request messages from its children after the node broadcasting an attestation request message. When this timer is triggered, the node stops to collect its children and starts to do TPM quote.
- $T_{AttRply}$: the attestation reply timer used in DA. This is the maximum time a node waits for attestation reply messages from its parents and children. When this timer is invoked, the node sends an attestation challenge to the parents and children from whom the node does not receive attestation reply.
- T_{Attkey}: the attestation new group key timer used in DA. This is the timer a node waits for a new group key. If a node does not receive the new group key message in this time, it sends an attestation challenge to its parents for the key.
- $t_{BrdDelay}$: the delay that a node must wait for in sending its attestation request message. The purpose is to avoid attestation message collision with its neighbors.
- $t_{AttRplyDelay}$: the delay that a node must wait for in transmitting its attestation reply message. This is determined based on the number of children a node has and the time needed to transmit the AttRply message.
- t_{Quote}: the time used in quoting the TPM.

- $C_{AttRply}$: the time interval used in transmitting a packet from its sender to its receiver.

6.6 PROMISCUOUS UNICAST

A node in an ad hoc network could send a single BBKA (as described in section 4.2.2) request and response to all its neighbors: BBKA messages need not vary per neighbor. Therefore, the BBKA could broadcast its messages, instead of unicasting them. Broadcasting reduces the number of messages and latency necessary for attestation. However, wireless data link layers, such as Wi-Fi [58], often support broadcasting much less reliably than unicasting. This section discusses promiscuous unicast, a technique that enables the network to reduce the number of messages and the latency of certain distributed algorithms more reliably than broadcasting.

Wi-Fi automatically acknowledges unicast frames and retries transmission if an acknowledgment times out. Wi-Fi can also be configured to automatically fragment large unicast frames and/or use small request-to-send/clear-to-send (RTS/CTS) control frames before sending them. These mechanisms can promote fast recovery from collisions (e.g., due to hidden terminals), and they can actually reduce the likelihood of collisions. However, they are unavailable for broadcast.

Promiscuous unicast enables the BBKA and other distributed algorithms to achieve broadcast benefits, such as fewer messages and less latency, without foregoing unicast's automatic RTS/CTS, fragmentation, and acknowledgments. Promiscuous unicast requires wireless interfaces to be set in promiscuous mode. When a node needs to broadcast a frame and it knows at least one destination, it sends a unicast frame to that destination with a *promiscuous unicast* bit rather than a regular broadcast frame set in its header. The frame's destination enables the data link-layer frame RTS/CTS, fragmentation, and acknowledgment, greatly reducing hidden-terminal effects. Other nodes that also receive the frame treat it as a regular broadcast frame, thereby achieving the message-reduction benefit of broadcasting.

Reception of a promiscuous unicast frame by n nodes may be less reliable than reception

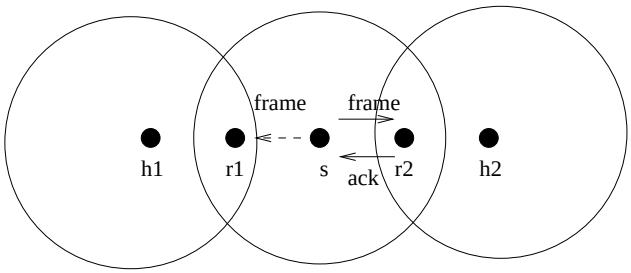


Figure 6.1: Like broadcast, a promiscuous unicast frame can be received by all nodes (r_1, r_2) within range of the sender (s). However, promiscuous unicast also provides fragmentation/reassembly and, for a designated receiver (r_2), automatic retransmission (e.g., in case of collision with h_2). Higher-layer recovery is needed only for other errors (e.g., collision with h_1).

of n unicast frames, each destined for one of the nodes. As illustrated in Figure 6.1, promiscuous unicast does not promote automatic recovery, e.g., in case a hidden terminal transmits at the same time, causing collision at a receiver other than the promiscuous unicast destination. Therefore, distributed algorithms that use promiscuous unicast need to have also higher-layer acknowledgments, timeouts, and retransmissions. As a possible optimization, on a higher-layer retransmission, the sender may vary the promiscuous unicast destination, in an effort to avoid collisions in as many directions as possible.

6.7 DISTRIBUTED ATTESTATION

This section discusses distributed attestation (DA), a novel approach for nodes in an ad hoc network to attest one another efficiently. DA first organizes the nodes in the network systematically such that they can attest one another at the same time, then utilizes the BBKA algorithm (described in section 4.2.2) to do the attestations. We assume that no pre-established security channels are established among nodes in such ad hoc networks. Therefore, we use the version of BBKA with DH key exchange. DA enables nodes in a

network can carry out attestation in parallel, thus greatly reducing the overhead that can result from the attestation of many nodes in a large network if attestation is not well-organized.

DA defines four types of messages: attestation request (*AttReq*), attestation reply (*AttRply*), new group key (*AttKey*), and no acknowledgment message (*NAK*). AttReq is used by a node to solicit attestation with other nodes. AttRply is the BBKA attestation reply message, which includes all of the necessary information about the node's configuration. AttKey is used to broadcast a new group key in a group. NAK is a message that tells the receiver that a message that was expected has not been received by the sender. If necessary, the node receiving the NAK message resends the requested message to the sender. Besides these messages, several timers (i.e., T_{AttReq}, $T_{AttRply}$, T_{Attkey}) are defined to regulate the progress of DA such that a node can take some actions if it does not receive an expected message from its neighbor in a limited time or if a specific event should be triggered at an expected time. Several time variables and constants are defined to facilitate DA, including $t_{BrdDelay}, t_{AltDelay}, t_{Quote}$, and $t_{AttRplyDelay}$ (as describe in section 6.4).

Algorithm 1 Receiving an Attestation Request Message

```
1: if KID_N ≤ 0 then
2:   if S_N == IDLE then
3:     P_N = S; Set T_AttReq;
4:   else if N = P_S then
5:     if S_N ≠ Quote then
6:       Add S to N's children list;
7:     end if
8:   else if P_N ≥ 0 and (AltP1_N = −1 or AltP2_N = −1) then
9:     if AltP1_N == −1 then
10:      AltP1_N = S;
11:    else if AltP2_N == −1 then
12:      AltP2_N = S;
13:    end if
14:  else if P_N < 0 and R_S > ID_N then
15:    P_N = S;
16:  end if
17: else if S_N == IDLE then
18:   Try merging with S;
19: end if
```

DA works in three stages. The first stage is *attestation tree construction*, which establishes the attestation relationship among the nodes. Figure 6.2 illustrates how the attestation

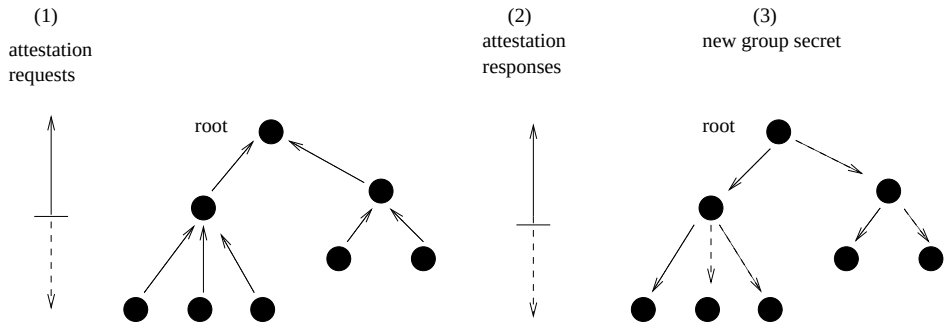


Figure 6.2: Distributed attestation achieves transitive configuration trust and key agreement with as few as one attestation request and one attestation response per node (both destined to node's parent, but received promiscuously by node's children) and one new group secret message per interior node (destined to one of the node's children, but received promiscuously by the others).

tree is formed through the broadcast of an AttReq message. Any node can initiate this stage by broadcasting an AttReq message to the network. An AttReq message includes a nonce, a node's parent, and its Diffie-Hellman (DH) public key. If a node receives an AttReq from another node for the first time, it sets the sender as its parent and propagates the AttReq to its neighbors after a random delay of time $t_{BrdDelay}$, which is uniformly distributed within a time constant $C_{BrdDelay}$. After forwarding the AttReq, the node sets the timer T_{AttReq}, during which it adds a node into its children list if it receives AttReq from the sender with the parent field set to its identity. Details about how a node processes an AttReq are shown in Algorithm 1. Therefore, as the attestation request message is propagated, an attestation tree is constructed. The initiating node is the root of tree and each intermediate node builds the parent and children relationship with its neighbors. As a result of the first stage, each node in the network aggregately receives AttReq from its neighbors at about the same time. This facilitates applying the BBKA in the entire network. To further consolidate connections among the nodes, our design allows each node to have alternative parents as well, which can be set up as follows. After a node received an AttReq from its parent and is waiting to propagate the message, it can set other nodes that it receives AttReq from as its alternative

parents. After this, in the second stage, each node can do attestation with its established parents and children.

Algorithm 2 Receiving Attestation Reply Message

```
1: if P_N == S and (N did not receive. AttRply from P_N) then
2:   if AttRply is verified then
3:     set P_N verified;
4:   else
5:     P_N = −1;
6:   end if
7: else if AltP1_N = S then
8:   if AttRply is verified then
9:     set AltP1_N verified;
10:  end if
11: else if AltP2_N = S then
12:  if AttRply is verified then
13:    set AltP2_N verified;
14:  end if
15: else if S is child of N then
16:  if AttRply is verified then
17:    update group size;
18:  end if
19: end if
```

The second stage, *BBKA attestation*, is triggered by the timer T_{AttReq}. As discussed in section 4.2.2, when the timer is invoked, each node starts to quote its TPM for attestation information based on BBKA since it has collected AttReq messages from its parents and children. Once the node finishes its TPM quote, it waits for a random delay $t_{AttRplyDelay}$, which is uniformly distributed within the time that is based on the number of children the node has as well as the time needed to transmit the AttRply message. Then, it promiscuously unicasts its AttRply message to its parents and children, while simultaneously setting another timer $T_{AttRply}$ to wait for AttRply messages from its parents and children. The random time delay at each node is used to help reduce message collision, since a node could have many children and it cannot receive AttRply messages from all of its children at the same time. Once a node receives an AttRply from another node, it checks whether the sender is its parent or child. If so, it updates its record which corresponds to the sending node. If not, it drops the message. Algorithm 2 details how a node processes AttRply messages. A NAK message is used to solicit an AttRply from a node's parent or children if no reply has been received in a limited time $T_{AttRply}$. When the timer $T_{AttRply}$ is invoked, a node sets a separate timer

T_{AttKey} for receiving a new group key. After the second stage, trust among nodes has been built. Each node knows which of its neighbors are trustworthy. If a node cannot attest one of its neighbors, it puts the neighbor into its blacklist, which maintains the list of untrusted nodes in the network. The blacklist can be shared among trusted nodes later on.

The third stage of DA is called *new group key dissemination*. When T_{AttKey} is invoked, the root of the attestation tree sends its attested children the AttKey message, including a newly-selected group identity, group key, and group priority. The group key is encrypted using the secrets shared between the parent and its trusted children (derived from DH key exchange algorithm). The AttKey message is propagated to other attested nodes in the network. How a node processes the AttKey message is illustrated in Algorithm 3. This new group key message is encrypted such that only nodes that succeed the attestation can receive the key. The new identity key is used to authenticate messages among members of this group later.

Algorithm 3 Receiving an Attestation New Group Key

```
1: if S == P_N or S == AltP1_N or S == AltP2_N or (P_S == N and S is trusted) then
2:   if N is trusted by S then
3:     update KID_N; update group size;
4:     if Number of children ¿ 0 then
5:       send new group key message;
6:     end if
7:   end if
8: end if
```

Along with the new group key, the root node also sends its *group priority number* in the AttKey message. The group priority number is an integer calculated as follows. When the attestation tree is established, every parent node knows how many children it has. After the second stage, the root node knows how many of its children are trusted. Also, since each AttRply message from a node contains its number of children, the root also knows how many children each of its trusted children has. The group priority number is the sum of the root node's trusted children and their children. The group priority number, which is the estimation of the size of a trusted group, is used as a criteria when two groups merge.

6.8 ATTESTED MERGER

Attested merger (AM) is a merge and rekey protocol designed to help nodes in groups with different group keys merge into one group. Nodes in a network are supposed to form a unique group after the DA finishes. However, this does not always happen.

There could be many reasons why nodes in a network have different group keys after DA. The two main reasons are the nodes' mobility and message collisions, which could cause partitions among nodes in the network at any stages of DA. One way such partitions can form is if the nodes are physically partitioned from other nodes when DA starts. This can cause the nodes to form their own groups. Also, even if the network is not physically partitioned, a node can move from one location to another during DA such that it cannot be reached by its established parents or children. This breaks the initial attestation relationship. The broken parent and children form their own groups. Another possibility is that a node's AttRply or AttKey message may not be received by its neighbors due to message collision, thus causing the group to fall apart during formation. Finally, malicious nodes may be inside the network. Although they cannot pass the attestation of other nodes, their presence may break the group into parts.

The attested merger protocol consists of two phases: *merge* and *rekey*. Both of these phases are designed for simplicity. During the first stage, nodes in two groups attest each other. It works as follows. When two nodes N_A and N_B in two different groups G_A and G_B meet each other, they send merge request (MrgReq) messages to each other. The two nodes then process these messages. As shown in Algorithm 4, based on the information in the MrgReq message, the two nodes know each other's group key identity, group priority number, and more. They then compare the information received with their own.

Then the node (N_A) in the group (G_A) with the lower group priority number sends its merge reply (MrgRply) message to the other node (N_B). N_B then verifies N_A's attestation information, as shown in Algorithm 5. If the verification succeeds, N_B sends MrgRply to N_A. Once N_A receives and verifies N_B's MrgRply, it goes to the rekey stage. If the attestation of either N_A to N_B or N_B to N_A is not successful, AM stops.

In the second stage, the *rekey* protocol helps the group with the lower group priority

Algorithm 4 Receiving a Merge Request Message

1: **if** $KID_N == KID_S$ **then**
2: return;
3: **end if**
4: **if** $S_N == IDLE$ **then**
5: set N's merge peer; set $S_N = MERGE$; start TPM quote;
6: **if** $PRI_N < PRI_S$ **then**
7: set $T_{MrgSend}$ to send merge reply to S;
8: **else**
9: set $T_{MrgRply}$ to wait for merge reply from S;
10: **end if**
11: **else if** $MrgPeer_N == S$ and (N initiated merge request to S) **then**
12: start TPM quote;
13: **if** $PRI_N < PRI_S$ **then**
14: set $T_{MrgSend}$ to send merge reply to S;
15: **else**
16: set $T_{MrgRply}$ to wait for merge reply from S;
17: **end if**
18: **end if**

Algorithm 5 Receiving Merge Reply Message

1: **if** $S_N == MERGE$ and $MrgPeer_N == S$ and (N has sent and received MrgReq from S) **then**
2: **if** $PRI_N > PRI_S$ **then**
3: **if** S is verified by N **then**
4: send merge reply to S;
5: **end if**
6: **else**
7: **if** S is verified by N **then**
8: N updates its group key and identity to S's; send RekeyDec message to other nodes in its group;
9: **end if**
10: **end if**
11: **end if**

number update its group key and identity to the group with a higher group priority number. This happens through a broadcast of a rekey decision (RekeyDec) message with the new group key and identity encrypted, along with the current group key. In the above example, N_A simply broadcasts the new group key (from G_B), encrypted with its current group key, to its group members. Once other nodes in group N_A receive the rekey decision message, they update their group key and broadcast the same message to their neighbors in the same group. Algorithm 6 shows how a node in a group processes the RekeyDec message.

Algorithm 6 Receiving a Rekey Decision Message

```
1: if KID_S == KID_N and (KID_N ≠ the new group key id) then
2:    update N's KID_N and group size;
3:    if S_N == MERGE and PRI_N < PRI_MrgPeer and (N's group size ¿ MrgPeer's group size)
      then
4:       reset N's merge state;
5:    end if
6:    forward the rekey decision message;
7: end if
```

6.9 MESSAGE FRAGMENTATION

This section discusses how we deal with message fragmentation in the AdHocSec layer.

Message fragmentation is needed if the length of a message sent from the AdHocSec layer is greater than the Maximum Transmission Unit (MTU). In our design, most of the messages used in DA and AM are very short. However, two messages (AttRply and MrgRply) are typically about 5K bytes in length, since they contain a lot of information (e.g., attestation measurement log list, TPM quote result, TPM AIK certificate). Therefore, these two messages need to be fragmented before being sent to the data link layer (DLL).

The method we use to fragment attestation messages in this work is similar to the method used for IP fragmentation. The AdHocSec header includes two fields that are used in fragmentation: *isfrag* and *frag_off*. *Isfrag* indicates that a current packet is a fragment and if there are subsequent fragments; *frag_off* indicates the position of current fragment in the entire message. Therefore, for the first fragment, isfrag is set to 1 and frag_off is set to 0; for the last fragment, isfrag is set to 0.

When a node sends an attestation message to the data link layer, it first checks the length of the message. If the message length exceeds the MTU, it slices the message into fragments, which are then sent to the DLL. All of the fragments of a message share the same sequence number in their headers. Therefore, when another node receives these fragments, it is able to identify these fragments and defragment them.

When a node receives an attestation packet, it first checks the isfrag and frag_off fields in the AdHocSec header. If it finds the received packet is a fragment, it checks its fragment queue. The receiver only starts to process an attestation message if it receives all the fragments of the message and defragments them correctly.

Each node maintains a fragmentation hash table for all the fragmented messages it receives. Each entry in the table is a structure identifying a fragmented message, and this is where the node keeps all of the received fragments for a given message. When a node receives a fragment, it finds the corresponding entry in the table by taking a hash based on the sender's identity, the sequence number, and the message type. Then, the node puts the received fragment into its fragment list, which indexed by message. Next, it checks if it has received all of the fragments. If it has, the node defragments the message and starts to process it. Otherwise, it waits for the arriving of other fragments.

The AdHocSec layer does not contain any explicit mechanism for acknowledging each fragment sent. Rather, the acknowledgement for each fragment relies on a mechanism of the 802.11 MAC layer. However, the AdHocSec layer does provide a timeout mechanism for each attestation message. If a node expects to receive a specific attestation message and it does not within a limited time, it sends a NAK message to a corresponding node. Once another node receives the NAK message, it tries to resend the attestation message to the node. This mechanism is similar to what transpires in the TCP and IP layers. The IP layer provides fragmentation, but no reliability, while TCP provides reliable data transfer. The disadvantage of this scheme is that the entire message must be resent if even one fragment is not delivered correctly.

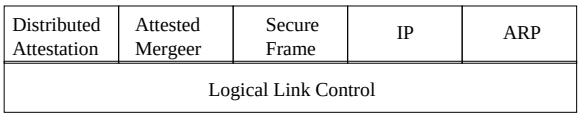


Figure 6.3: AdHocSec is implemented between the network and data link layers and is comprised of three protocols: Secure Frame, Distributed Attestation, and Attested Merger.

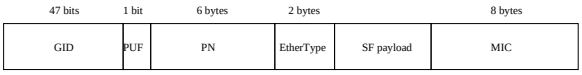


Figure 6.4: Secure frame includes fields of group secret ID (GID), promiscuous unicast flag (PUF), packet number (PN), payload, Ethernet type, and message integrity code (MIC).

6.10 ADHOCSEC LAYER

This section describes AdHocSec's layering and frame format.

AdHocSec is comprised of three protocols: DA, AM, and Secure Frame (SF). They are layered directly on top of LLC, with different EtherType values from those of IP and ARP. DA and AM frames are used for distributed attestation and attested merger. SF provides confidentiality, authentication, and replay protection for data frames. Figure 6.3 illustrates the layering of DA, AM, and SF.

SF defines a header and a trailer for each SF payload, as shown in Fig. 6.4. The SF trailer contains a message integrity check (MIC). SF uses an efficient symmetric-key algorithm similar to AES-CCMP [32] for encrypting the SF payload and computing the MIC. The SF header's group secret id (GID) field identifies the key used for encryption and authentication. This key is set by the DA or AM protocols. If an SF receiver receives an SF frame with group secret id whose corresponding key the receiver does not know or indicates that the MIC field is inconsistent, the receiver drops that frame. When DA or AM modifies the group secret id and key, SF saves the previous values for enough time to accommodate frames that may be in transit. The SF header's EtherType field may have values IP or ARP.

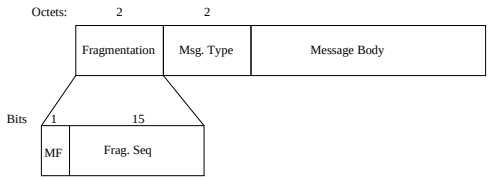


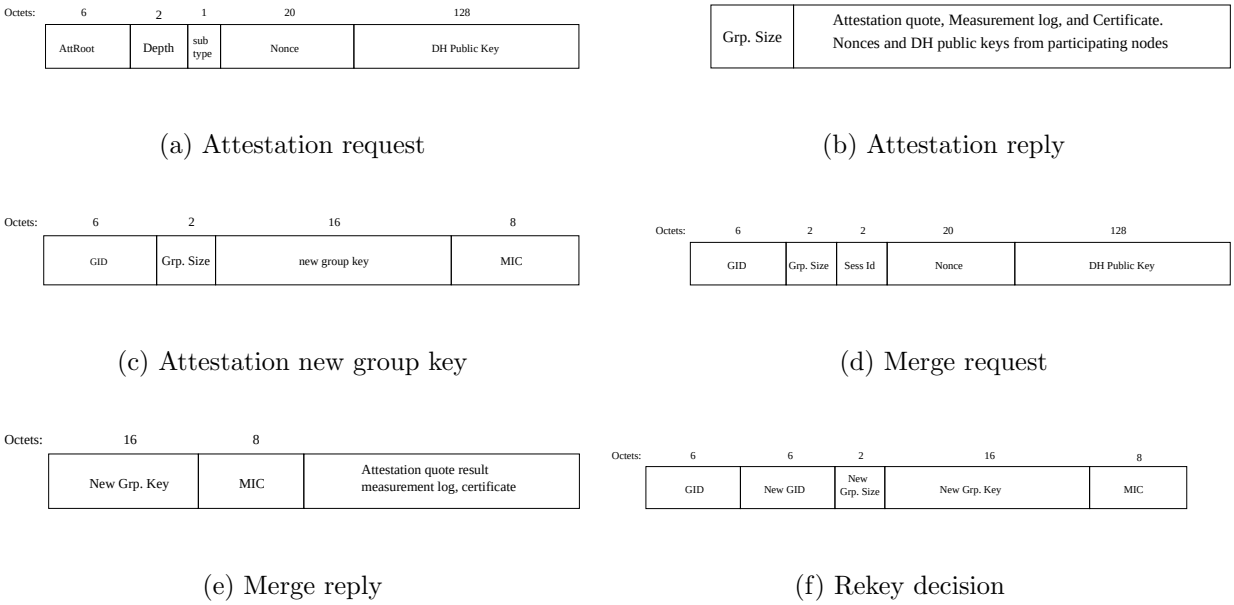
Figure 6.5: DA/AM frames include fields of more fragment flag, fragment number, message type, and payload.

Use of the SF layer could be governed by a Security Policy Database (SPD) similar to IPsec's [11]. In this case, communication with certain nodes might use SF, while communication with other nodes might not. If an outgoing IP datagram needs SF and the node is not in key agreement, AdHocSec could automatically invoke DA, much like IPsec invokes IKE. We have not implemented this functionality, however. In our simulations, either all or no traffic used AdHocSec.

The sequence number in the SF header increments by one for each SF frame transmitted to a receiver with a given group key. SF receivers use this field to detect and discard replayed frames. SF receivers keep in a hash table the highest sequence number received (if any) with a given key from each sender. SF receivers drop frames with sequence number not greater than that found for the sender in the receiver's table. This replay protection is effective for unicast frames, but may fail for broadcast frames. In particular, an intruder can capture a broadcast frame and replay it to any receiver not within range of the original sender (wormhole attack [68]). To protect ad hoc networks against such attacks, defenses such as *packet leashes* [68] are needed.

The DA/AM protocol comprises a header and a payload, as shown in Fig 6.5. The DA/AM header contains the message type for DA and AM. It also contains the fields for DA/AM message fragmentation, especially for DA/AM attestation reply message since its length is usually greater than 2K bytes.

Figure 6.6 lists the format for different types of DA and AM messages. The *Grp. Size* in all of the messages in Figure 6.6 indicates the size of attestation group; *GID* is the group



(a) Attestation request

(b) Attestation reply

(c) Attestation new group key

(d) Merge request

(e) Merge reply

(f) Rekey decision

Figure 6.6: Distributed attestation and attested merger message format

identifier; MIC is the message integrity code for the message. In figure 6(a), *AttRoot* identifies the root of the distributed attestation tree and is usually the node that initiates DA; *Depth* indicates the depth of the node in the tree; and *subType* indicates whether the attestation request is an initial request or an attestation refresh request. The *Sess. ID* in Figure 6(d) is a random number that uniquely identifies the session between the two nodes that engage in attested merger. Since two nodes could send merge request randomly when they meet each other, the session ID in the message acknowledges that two nodes receive merge request from each other and agree to engage to merge.

Implementing AdHocSec between layers 2 and 3 provides both benefits and difficulties. The major benefit is that it enables AdHocSec to transparently secure layer-3 functionality, such as ad hoc routing and forwarding, as well as higher-layer protocols and applications, without modifying them. However, this architecture also imposes certain constraints. In particular, we had to carefully avoid the use in AdHocSec of primitives that might introduce circular dependencies with higher layers. For example, one might use a leader-election

algorithm in DA to avoid the formation of multiple trees. However, one would then need to secure the leader election, which would be difficult. Instead, we used in AdHocSec only link-layer functions, enabling AdHocSec to secure higher-layer algorithms, including leader election.

6.11 EVALUATION

We evaluate our work by both simulation and implementation. Simulation enables us to see the performance of our work in a large network with many nodes. Implementation tests if our approach works well in real systems.

For simulation, we used the ns-2 network simulator [69]. To add the AdHocSec layer into ns-2, we create and insert a new component in the ns-2's wireless model, in which we implement our attestation algorithms.

For implementation, we modified the Linux kernel and used the implementation technique similar to PRAN [70].

6.11.1 SIMULATION

We use the following set up for simulation. We have 50 nodes with random location in the field of 1500m x 300m. Of the 50 nodes, there are 20 pairs of sources and destinations. Each pair transmits UDP packets with a CBR (Constant Bit Rate) of 4 packets per second. The length of each UDP packet is 512 bytes. Each pair starts to send and receive packets at a random time uniformly distributed between 0 and 180 seconds. Each node propagates using the two-way ground propagation model with nominal radio range of 250 meters. The original routing protocol used is DSR [37]. Each simulation takes 900 seconds.

We run the simulation using a Dell Dimension 4550 workstation with a Pentium-4 2.4GHZ CPU, 256M RAM, Fedora core 3, Linux 2.6.14 kernel, and ns-2 version 2.28. We collect the simulation traces and use Perl scripts to process them. The following sections discuss our results, as they relate to attestation overhead and network performance.

To evaluate our work, we designed *two sets* of simulation experiments. The first set evaluates the performance of the designed attestation algorithms, while the second set compares AdHocSec with other secure approaches (Ariadne).

Table 6.1: Nodes' Speed Ranges and Steady-State Average Speeds

Speed range	Steady-state avg. spd.
[12,32]	20.39
[11,31]	19.30
[10,30]	18.20
[9,29]	17.09
[8,28]	15.96
[7,27]	14.82
[6,26]	13.64
[5,25]	12.43
[4,24]	11.16
[3,23]	9.82
[2,22]	8.34
[1,21]	6.57
[1,19]	6.11
[1,17]	5.65
[1,15]	5.17
[1,13]	4.68
[1,11]	4.17
[1,9]	3.64
[1,7]	3.08
[1,5]	2.49
[1,3]	1.82
[0.2,1.0]	0.50

6.11.1.1 ATTESTATION PERFORMANCE EVALUATION We evaluated the attestation performance to answer the following questions: how long it takes the network to reach a trusted state in which every node has been attested and the network has a global group key and what network load (i.e., number of attestation packets and bytes) do our algorithms bring into the network? We collect the following matrices for performance measurement during the simulation:

- *Latency for Global Key Agreement*: the attestation time used from the beginning of the initial distributed attestation to the time when all nodes in the network has the global

key.

- *DA and AM Packet Overhead*: the number of DA and AM messages used.
- *DA and AM Byte Overhead*: the number of bytes used for DA and AM.
- *DA and AM message type distribution*: the distribution of types of message used during attestation.

Table 6.2: Timers and Time Variables Used in DA and AM

Timers	time (seconds)
T_{AttReq}	0.08
$T_{AttRply}$	0.80
T_{Attkey}	0.60
t_{Quote}	0.71
$C_{AttRply}$	0.021
$t_{BrdDelay}$	uniformly distributed within 0.024
$t_{AttRplyDelay}$	jitter(maxslot(depth)) * $C_{AttRply}$
$t_{AltDelay}$	0.012

To evaluate the attestation performance, we warmed up each simulation until the nodes' average speed reached a steady state before starting our algorithms and measurements [71]. Using the tool provided by LeBoudec and Vojnovic [71], we generate the node's movement pattern in this simulation. Table 6.1 shows the speed ranges and steady-state average speeds of 23 nodes used in our simulations. For each average speed, we ran 10 random scenarios with different seeds and initial node locations. The timers and time constants used in the simulation are listed in Table 6.2.

The overhead for the AdHocSec layer of encryption and authentication is based on the benchmark result from the Crypto++ Library 5.2.1 [72]. The benchmark was run on a system with an AMD opetron CPU 1.6GHZ and Linux kernel 2.4.21. The throughput for the AES algorithm is 46.461MB per second. Therefore, the delay caused by the AdHocSec layer is calculated based on the size of message payload and the AES algorithm throughput.

The experiment results for the performance of attestations are presented and discussed in the follows.

Figs. 6.7 through 6.12 evaluate the DA and AM protocols with null random-waypoint pause time. Figure 6.7 shows the latency for all 50 nodes to reach transitive configuration

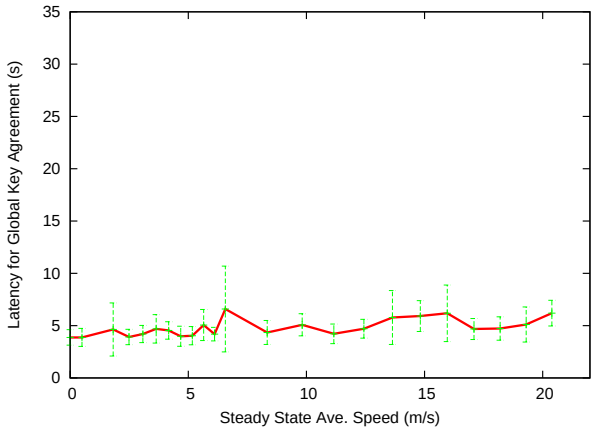


Figure 6.7: The latency for reaching global key agreement does not change very much with steady-state average node speed and typically remains between 4 and 8 s.

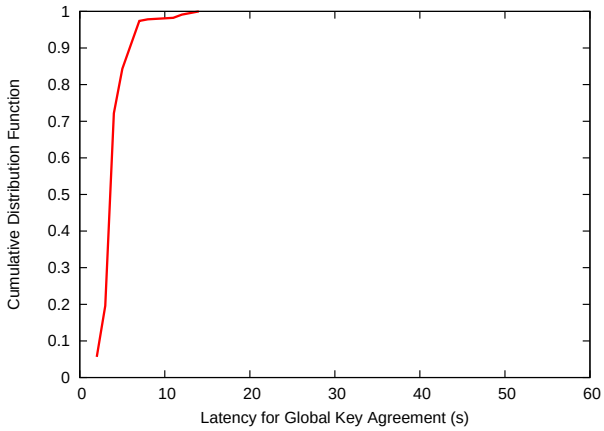


Figure 6.8: About half of the scenarios reached global key agreement in less than 5 s, and 95% did so in less than 8 s.

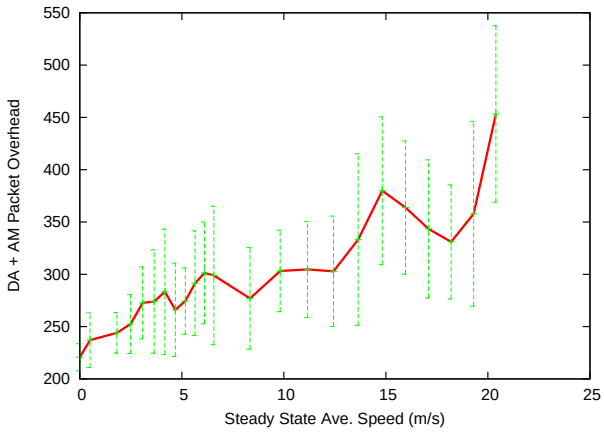


Figure 6.9: The average number of packets sent by the DA and AM protocols is close to the theoretical lower bound when nodes do not move; increases are roughly proportional to the steady-state average node speed.

trust and global key agreement. As the standard deviation shows, latency fluctuates at each steady-state average speed. However, the average latency does not change too much with average node speed. This indicates that the DA and AM work very well, regardless of the average node speed in the speed range listed in Table 6.1. Figure 6.8 shows the cumulative distribution function of latency for global transitive configuration trust and key agreement. For all simulated scenarios, global key agreement takes less than 5.0 s in about 50% of the cases, and less than 8.0 s in about 95% of the cases.

Figures 6.9 and 6.10 illustrate the number of packets and the number of bytes transmitted, respectively. In both cases, the number used for DA and AM grows roughly proportionally to the steady-state average node speed.

Figure 6.11 explains the relationships seen in Figures 6.9 and 6.10. For networks in which the nodes do not move, the average number of partitions is less than two, and AM often is not needed for global key agreement. However, as nodes move faster, neighbors move further apart before DA completes, and the number of partitions that AM needs to merge increases. At 20.39 m/s, the average number of partitions is three. The higher the number of partitions, the higher the packet and byte overhead incurred to merge them.

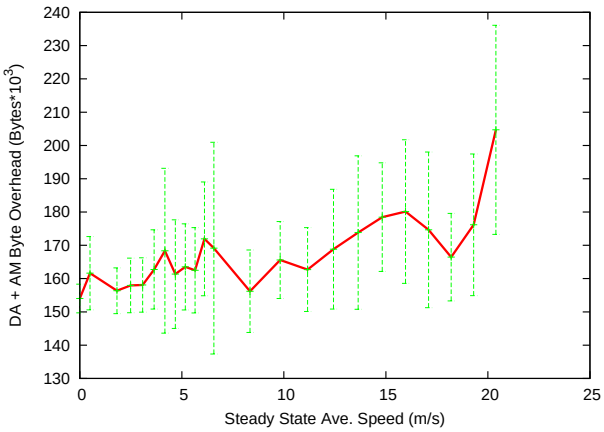


Figure 6.10: Number of bytes sent by DA and AM.

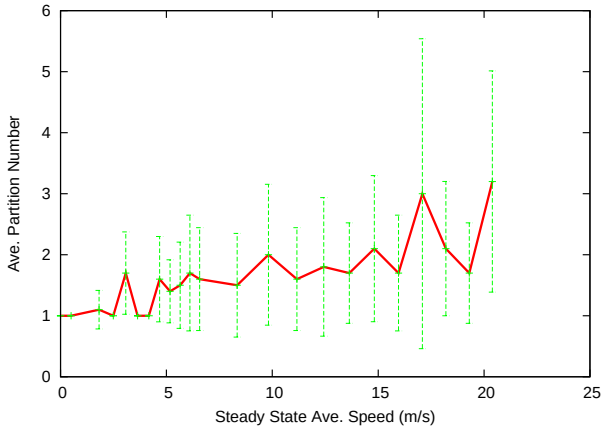


Figure 6.11: When nodes do not move, DA often can achieve global key agreement without any mergers. However, higher node speeds tend to fracture DA's spanning trees, increasing the number of partitions that need to be merged by AM.

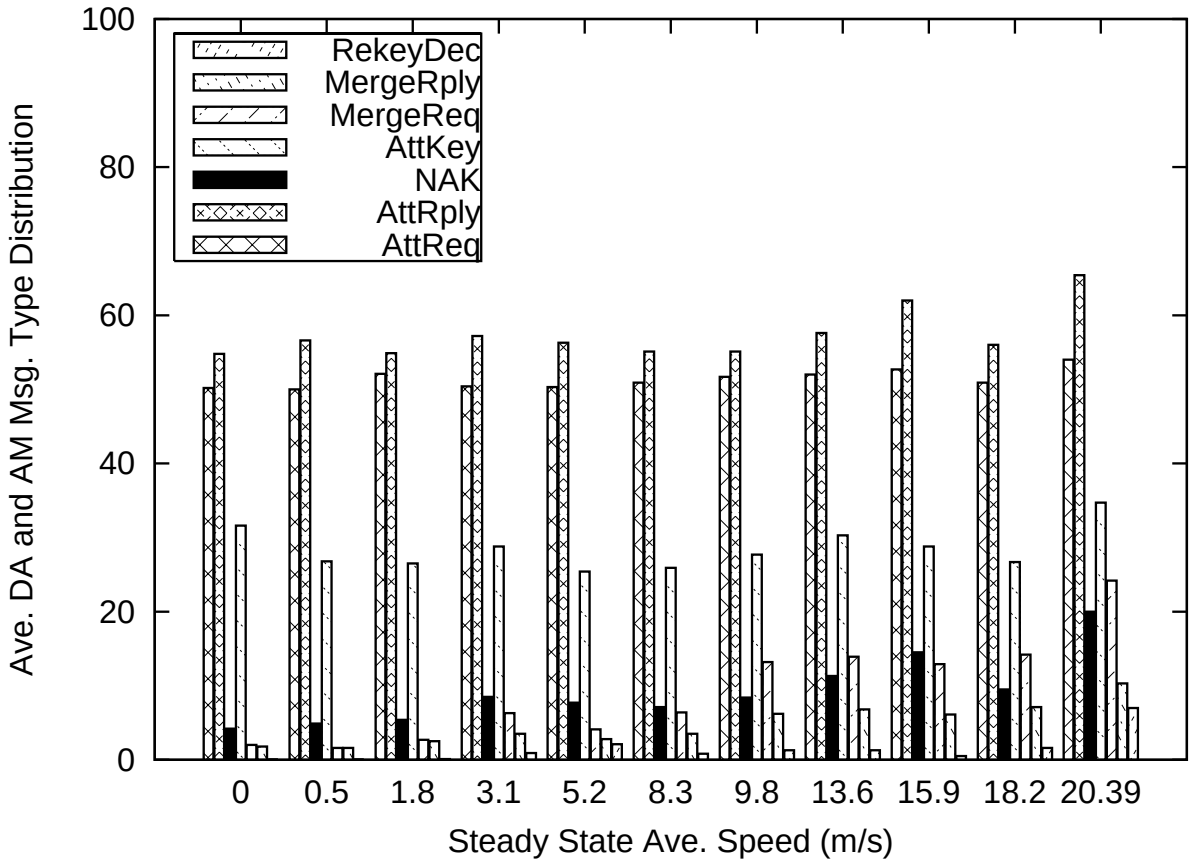


Figure 6.12: On average, when nodes do not move, each node transmits only about one attestation request and one attestation reply, as expected. However, when the average node speed increases, the frequency of NAKs and merger attestation requests rises markedly: more neighbors move out of range before DA completes and therefore start AM.

Figure 6.12 shows the average number of packets of each type. The marked increase in the frequency of MrgReq is symptomatic of the higher number of partitions that exist as node speeds increase. The number of AttReq is close to the number of nodes. However, the number of AttRply is usually more than the number of nodes, due to AttRply fragmentation, message collision, and loss caused by node movement. The number of other attestation messages increases a little with the average node speed, but not much. This indicates that the DA and AM protocols work well in handling increases of node speed.

6.11.1.2 COMPARING WITH ARIADNE The objective of this second simulation is to compare the network impact of AdHocSec with other security approaches. Since our solution is the first approach that integrates attestation in ad hoc networks, we cannot compare our solution with other experiments using the same approach. However, since AdHocSec protects any original routing protocols (e.g., DSR [37]), its addition to an ad hoc network replaces the use of secure routing protocols (e.g., Ariadne [13]). Therefore, we glean valuable information by comparing AdHocSec and the alternative it may replace, Ariadne, in terms of their network performance. (It is worth noting, however, that AdHocSec protects more than just routing protocols, and so the benefits of these two options are not initially equal.)

For this purpose, we run the simulation (almost the same configuration as Ariadne) with different node pause times of 900, 600, 300, 120, 60, 30, and 0 seconds. For each pause time, we run the simulation with 10 different scenarios (i.e., different node locations and random seeds). In total, we complete 70 simulations. For each pause time, we calculate the average of the 10 runs. These are the figures from 6.13 to 6.16 illustrated in the results graph and discussed below. The 70 simulation scenarios are generated using tools included in the ns-2 simulator. The node movement pattern is generated using "setdest," while the traffic pattern is generated by using "cbrgen.tcl."

We collect the following metrics for comparing AdHocSec and Ariadne during the second set of simulations:

- *Data packet delivery ratio*: the ratio of the number of data packets received versus the number of data packets sent.

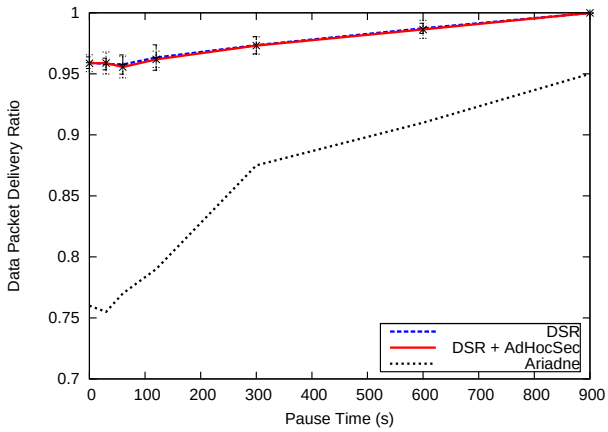


Figure 6.13: Data packet delivery ratio. AdHocSec has much lower impact on DSR performance than does Ariadne.

- *Data packet average latency*: the average latency for data packets to reach their destinations.
- *Routing packet overhead*: the number of routing packets sent during the simulation.
- *Routing byte overhead*: the number of bytes of routing packets sent during the simulation.

The simulation results are presented and discussed in the follows.

Figures 6.13 through 6.16 evaluate AdHocSec after global key agreement (i.e., primarily SF). To increase validity when comparing our findings to previously published results [13], we did not warm up the simulations before measurements.

Figures 6.13 through 6.16 also show previously reported average performance of Ariadne, a secure version of DSR, for the same scenarios [13]. Note that DSR with AdHocSec differs from Ariadne in dimensions other than performance. Compared to DSR with AdHocSec, Ariadne has the disadvantage of securing only routing, not other protocols or applications. It also has the advantages of not requiring TPM or any other hardware support and, thus, being immune to hardware attacks against it. Note that optimization might improve Ariadne's previously-reported performance.

Figure 6.13 shows the data packet delivery ratios. DSR's delivery ratios with and without AdHocSec are statistically indistinguishable, with both being above 96%. Ariadne is reported

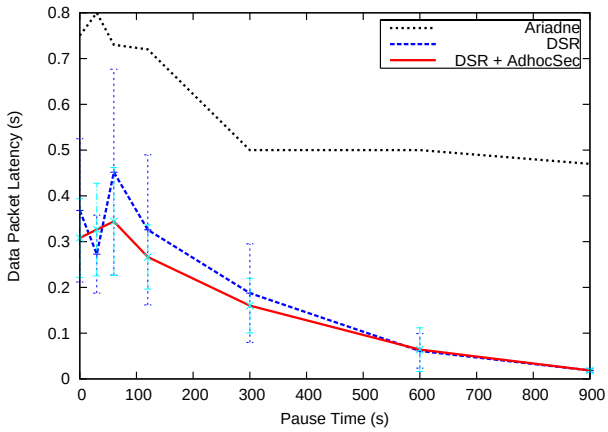


Figure 6.14: Differences in DSR data packet latency with or without AdHocSec are not statistically significant.

to have a significantly greater overhead, reducing the delivery ratio for continuously mobile nodes to 76%.

Figure 6.14 shows the data packet latencies. DSR latencies with and without AdHocSec are statistically indistinguishable. Reported latencies for Ariadne are much greater (0.75 s vs. 0.38 s for continuously mobile nodes and 0.47 s vs. 0.02 s for immobile nodes).

Figures 6.15 and 6.16 show, respectively, the number of packets and the number of bytes sent by routing protocols during the simulation. DSR's overheads with and without AdHocSec are statistically indistinguishable. Ariadne's reported overheads are much greater (118,000 vs. 34,000 packets and 26 MB vs. 2 MB for continuously mobile nodes, and 40,000 vs. nearly 0 packets and 15 MB vs. nearly 0 bytes for immobile nodes).

These figures show that AdHocSec's overhead is very small. AdHocSec can secure routing, forwarding, and higher-layer protocols and applications with little effort and overhead.

6.11.2 IMPLEMENTATION

In addition to the simulations, we implement AdHocSec in Linux as well. This enables us to evaluate our approach in real systems. Our implementation uses the same methodology as the work described in [70], in which the real implementation makes use of simulation (protocol

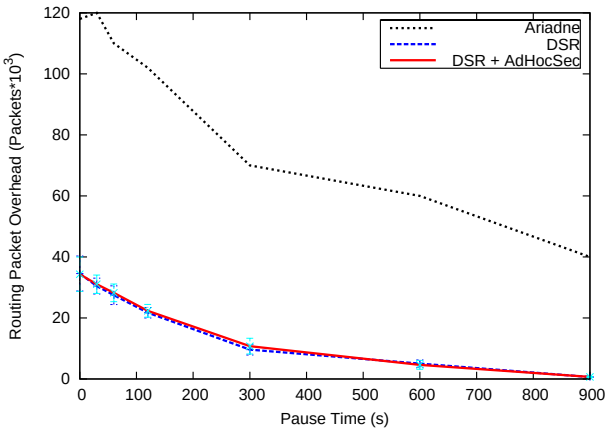


Figure 6.15: Ariadne's packet overhead is much greater than AdHocSec's.

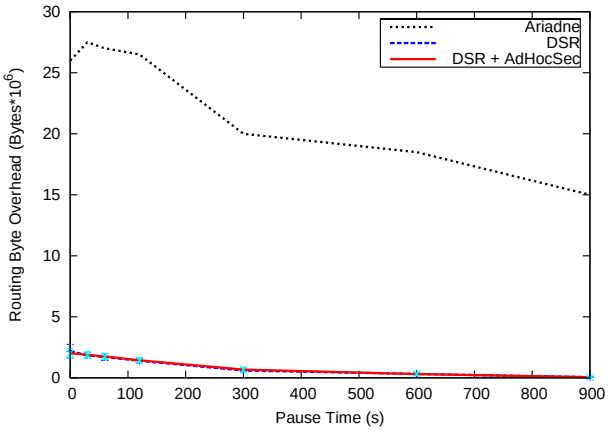


Figure 6.16: Unlike Ariadne, AdHocSec adds negligible byte overhead to DSR.

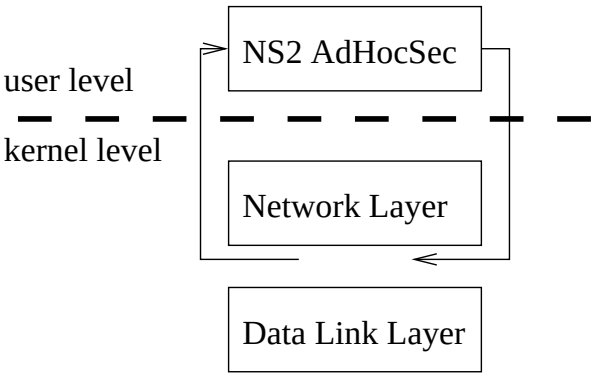


Figure 6.17: AdHocSec implementation in Linux

written for ns-2 simulator) with little modification. Therefore, the proposed protocol is not re-implemented in the OS kernel, but still uses the NS2 simulator running at user-level. The OS kernel is modified to provide some interfaces to communicate with the user-level protocol. Each time the protocol is needed, the OS kernel communicates with it through the provided interfaces. Since we leverage the protocol implemented in the NS2 simulator, all the parameters (e.g., broadcast delay, attestation request timer, new group key timer, etc), besides the real implementation (e.g., attestation quote delay, attestation and merge reply) used in the simulation are still valid.

The advantage of this implementation, as explained by Saha et al. [70] is that the same protocol can be used for both the simulation and implementation, with little modification needed. Another means of doing implementation is to actually insert the AdHocSec layer between the network and data link layers of the Linux kernel. This methodology is possible, but would likely require additional, time-intensive work unnecessarily. The advantage of this approach is a reduction in the amount of code-writing that is required, since the simulation code can be used for both the ns-2 simulator and the real system. The disadvantage is that packet processing is slow, since all of the network traffic must be diverted to the user level for processing. For an efficient real system, the implementation should be done at kernel level.

6.11.2.1 METHODOLOGY Figure 6.17 shows the framework of our implementation. We run the modified ns-2 simulator with our added AdHocSec layer at the user level and provided hooks inside the Linux kernel. These hooks reside between the network and data link layers of the Linux kernel network stack. Packets from either the data link layer or the network layer are directed to the user-level ns-2 simulator to be processed by the AdHocSec layer implementation. Output packets from the ns-2 simulator are fed back to the hooks for the Linux kernel to process. If the packets are going out, they are fed into the data link layer, otherwise the network layer.

In our implementation, packets flow through a node as follows. Any downstream packet from the network layer (a packet from an application, routing, or ARP) is intercepted by the hooks in the kernel. This packet is then diverted to the user-level ns-2 simulator with AdHocSec implemented. If the node does not have a group key, AdHocSec drops the packet. Otherwise, AdHocSec wraps it, by adding the SF header, encrypting it, and adding the MIC (Message Integrity Code), then feeds it back to the kernel through the hooks. The packet is then delivered to the lower level of the network stack.

An upstream packet is processed as follows. When the node receives a packet from its NIC (Network Interface Card), the hooks in the Linux kernel first divert the packet to the user-level ns-2 simulator. If the received packet is either a DA or AM attestation message, AdHocSec processes it and send a reply attestation message downstream through the hooks. If the received packet is a data packet (a packet from an application, routing, or ARP), AdHocSec verifies the packet and removes the SF header, then feeds it back to the kernel through the hooks. The packet is then delivered to the network layer. Therefore, the AdHocSec layer in the ns-2 simulator fulfills the functions defined as occurring between the network and data link layers in the stack, while remaining in the user level of the system.

The hooks between the Linux kernel and the ns-2 simulator are implemented by using the raw *PACKET socket* interface defined in Linux. The PACKET socket enables applications at the user level to receive and send packets directly to the NIC without going through the TCP/IP layer, thus allowing users to manipulate the packet format and protocol. The hooks must be designed to consider several different packet flows, including how to divert a receiving packet from the DLL to ns-2 and how to divert a downstream packet from the

network layer to ns-2. For the output packets from ns-2 simulator, the hooks must also be designed to consider how the ns-2 simulator transfers packets to either the network layer or the DLL.

With these concerns in mind, we begin by creating a raw PACKET socket in the ns-2 to send and receive packets to the Linux kernel. It is quite simple to divert a receiving packet from DLL to the ns-2 simulator by using the PACKET socket, which can be configured to only receive packets with the SF header. In order to divert an outgoing packet from the network layer to the ns-2 simulator, we modified the kernel file dev.c so that the packet is diverted to the created raw PACKET socket before the kernel passes it to DLL. An outgoing packet from ns-2 can be either a downstream packet heading to the DLL or an upstream packet going to the network layer. We modify the kernel file af_packet.c so that an outgoing packet from ns-2 with a SF header is delivered to DLL and a packet with just the IP or ARP header is delivered to the network layer.

Ns-2 is also modified such that it can communicate with the Linux kernel correctly. Since packets used in ns-2 are different from the real packets in a system, we have to add a packet converter in ns-2. For packets received from the hooks, the converter acts as an interface to convert packets received from the Linux kernel to ns-2 packets. Otherwise, if ns-2 is outputting a packet to the Linux kernel, the converter converts the ns-2 packet to real packet, and sends it to Linux kernel.

6.11.2.2 EXPERIMENT RESULTS After we finished the implementation, we did two sets of experiments. We first installed the implemented system on three computers, two IBM laptops and one IBM workstation, all with embedded TPM secure coprocessors, and did some analytical experiments. We tested the DA and AM algorithms on these computers, with positive results. After attestation, we run some user-level applications, such as ping, SCP (Secure CoPy), and ssh. These applications are able to run without any problems. While these applications were running, we checked the debug information displayed on the screen and could see that the packets go through the Linux kernel and the AdHocSec layer in the ns-2 simulator correctly.

We further did some experiments and collected the experiment data, by using another

two laptops: one IBM laptop with AtMel TPM version 1.1, and the other HP Compaq nw8440 laptop with Infineon TPM chip version 1.2 (note that these two laptops use different TPM chips and versions and thus require different Linux kernel drivers and user-level support utilities).

We use the following metrics to evaluate our implementation.

- Round trip time (RTT) and throughput between two nodes without AdHocSec implementation.
- RTT and throughput between two nodes with AdHocSec implementation, but no AES CCMP for packet encryption and authentication.
- RTT and throughput between two nodes with AdHocSec and AES CCMP implementation. the above two experiments are used to evaluate the impact of AES CCMP in the implementation.
- Latency, number of packets, and number of bytes for reaching key agreement between two nodes when they are within range of each other (tests distributed attestation).
- Latency, number of packets, and number of bytes for reaching key agreement between two nodes when they are within range of each other, but in different group (tests attested merger).

For the above experiments, we use *ping* (by pinging from one node to another for 10 times) to measure the RTT, and *TTCP* to measure the throughput between two nodes. When we use TTCP, we set one node as receiver, and another one as sender to transfer a file Linux-2.6.18.8.tar.bz2 (Linux kernel source) with the size of 41,842,273 bytes. The throughput was measured with the following setting: buflen=8192, nbuf=2.48, align=16384/0, port=5001, and TCP protocol. Experiment results are showed in tables 6.3, 6.4, 6.5, and 6.6. One thing that keeps in mind when we look at the experiment results here is that we basically leverage the protocol implemented in the NS2 simulator, therefore, all the parameters (e.g., broadcast delay, attestation request timer, new group key timer, etc), besides the real implementation (e.g., attestation quote delay, attestation and merge reply), used in the simulation are still valid.

Table 6.3: Round trip time (ms)

	Average	Standard deviation
W/o AdHocSec	2.14	0.40
AdHocSec w/o CCMP	2.34	0.19
AdHocSec w/ CCMP	2.47	0.39

Table 6.4: Throughput (KB/s)

	Average	Standard deviation
W/o AdHocSec	553.39	0.56
AdHocSec w/o CCMP	543.82	3.21
AdHocSec w/ CCMP	541.32	1.68

From table 6.3, we can see that the AdHocSec layer (with CCMP implemented) causes 0.3ms more latency, which is about 15%. Considering that this implementation is based on NS2 simulator and runs at user-level, we think this is a pretty good performance. Also as we can see, CCMP encryption and authentication to packets do cause a little more delay, but is very acceptable.

Table 6.4 shows the measured throughput between two nodes by using TTCP. As we can see, AdHocSec with CCMP implementation did reduce the throughput by 12KB/s (about 2.1%), compared with the measurement without AdHocSec implementation. This shows that AdHocSec really does not affect too much the network performance in term of throughput. Also, we can see that there is little difference between the one with CCMP and without CCMP implementation.

Table 6.5 shows the latency between two nodes for DA and AM. We did the experiment for twice with different setup. One was done with the number of entry in the attestation measurement log is 1, the other was node with number of entry is 67. Since the size of measurement log affects the size of attestation and merge reply, we want to see its impact to global key agreement latency. The reason we used 67 in the measurement log list is because

Table 6.5: DA and AM latency (s) with the number of entry in measurement log is 67 and 1

Number of entries in measurement log	67	1
DA avg.	1.15	1.03
DA stdev	0.21	0.04
AM avg.	0.806	0.789
AM stdev	0.002	0.004

our simulation was done with this parameter.

The latency for DA was measured at the node that receives attestation request, not the one that initiates DA. The experiment was done in the following procedure. The two nodes were started to do DA at almost the same time. Each node started to send attestation request periodically since they were in unattested state. When one node received the attestation request from another node, it treated the sender as its DA parent and recorded the starting time. DA continued between the two nodes. Finally, when the child node received the new group key from its parent, it recorded the time. The interval between the two time recorded is the value for DA in table 6.5. The latency for AM was measured by taking one of the two nodes (after they finished DA) down for a while, then restarted. Since the other node was already in attested stage, but the one that was took down in unattested stage, they started to merge. The time for AM showed in Table 6.5 is the time interval on the node that was restarted. The experiment for DA and AM latency was done for 5 times to have the average and standard deviation. As we can see, the latency in Table 6.5 and Figure 6.7 is different. This is mainly due to that, in this implementation experiment, we only had two nodes, therefore, there is not too much delay required for attestation reply and new group key dissemination. And there was no group partition. Figure 6.7 shows the global attestation latency for 50 nodes. There are always partitions due to some reason. therefore, the latency in that Figure usually is a little more than the sum of DA and AM. Section 6.12.1 analyzes the latency for DA and AM in detail.

Table 6.6 lists the packet number and bytes transferred between the two nodes. We watched that there was only one number of message for each type of messages transferred.

Table 6.6: DA and AM message size (bytes) when the number of entry in measurement log is 67 and 1

Number of entries in measurement log	67	1
ATT_REQ	176	176
ATT_RPLY	2074+2074+508	2074+96
NEWGRP_KEY	50	50
MRG_REQ	176	176
MRG_RPLY	2074+2074+470	2074+58
REKEY	74	74

As we can see, the attestation and merge reply messages are fragmented into three frames when they were sent out with 67 entries in measurement log, but two with one entry in measurement log. Together with Table 6.5, we can see that the size of measurement log did impact DA and AM. If there are lots of nodes involved in DA, size of measurement log affects the number of frames, thus causes collision and delay.

6.12 PERFORMANCE ANALYSIS

In this section, we analyze the performance of our designed algorithms, including DA, AM, and SF. The performance metrics we discuss here include the attestation latency for global key agreement, bandwidth consumption, and the impact of SF on upper level protocols and user-level applications.

6.12.1 LATENCY FOR GLOBAL KEY AGREEMENT

The attestation latency for global key agreement depends on many issues, including the DA and AM algorithms, the distribution of the nodes' physical locations, and traffic patterns among the nodes in a network. In this section, we only discuss the impact of DA and AM on it since we have no control of nodes' physical locations and traffic patterns. However,

we should know that nodes' physical partitions can cause a long latency for the global key agreement, as indicated in Figure 6.7, where the nodes' average steady speed is 1.82 seconds. Traffic pattern among nodes affects AM since AM is triggered by the communication between two nodes with different group identities; two groups may not be able to merge if they do not communicate.

We aim to give a low bound for the attestation latency and analyze the impact of DA and AM on it separately. In the following, we first analyze the latency caused by DA ($T_{DAlatency}$), then the latency caused by both DA and AM ($T_{latency}$). After that, we analyze the special case for DA and AM in which there are only two nodes involved in attestation.

For an intermediate node in the DA tree, $T_{DAlatency}$ is affected by the timers and other time variables used in DA. As we discussed in section 6.7, DA attestation proceeds in three stages. During these stages, several timers, including T_{AttReq}, $T_{AttRply}$, and T_{AttKey}, are used to regulate the DA process. Besides these timers, some other time variables and constants are also used. $t_{BrdDelay}$ is the delay before a node sends an AttReq message when it receives an AttReq from its parent. $t_{altdelay}$ is the extra time delay that a node waits for AttReq from its alternative parents before sending its AttReq message. t_{Quote} is the time consumed by the TPM quote. $t_{AttRplyDelay}$ is the random time delay that a node needs to wait before sending its AttRply message to its parent and children, in order to avoid message collision. These timers, time variables, and time constants all together affect the attestation latency for global key agreement.

$$T_{DAlatency} = t_{BrdDelay} + t_{Altdelay} + T_{AttReq} + t_{Quote} + T_{AttRply} + T_{AttKey} \tag{6.1}$$

The relationship between the DA latency $T_{DAlatency}$ for an intermediate node and these timers, variables, and constants is shown in Equation 6.1. $t_{BrdDelay}$ is a variable uniformly distributed between 0 and the maximum values. If we take its minimum value, based on Equation 6.1 and Table 6.2, we can estimate the minimum value of $T_{DAlatency}$:

$$T_{DAlatency} > 2.20 \text{ s} \tag{6.2}$$

However, the timing for the *root* of a DA tree is a little bit different from an intermediate node. The attestation root does not need to wait for a $t_{Altdelay}$, but its attestation

starting time depends on the attestation startup jitter, which is the random time uniformly distributed between 0 and 2.5 seconds (as in our simulation experiment). As we find in our simulation, most root nodes can pick up a very small value to start the DA process, therefore, we ignore this overhead. By eliminating $t_{BrdDelay}$ and $t_{Altdelay}$ in Equation 6.1, we can obtain a modified DA latency for the root as follows.

$$T_{DAlatency} > 2.19 \text{ s} \tag{6.3}$$

The analysis of attestation latency above is based only on DA, and we assume that all of a network's nodes are able to participate in DA and form one group. Another issue that is neglected in the above analysis is message loss, which may cause more latency. Therefore, the $T_{DAlatency}$ poses a low bound for attestation latency caused by DA.

AM brings little overhead to the latency for the global key agreement once it starts. The main overhead is the time used for the TPM quote (about 0.71 seconds). Therefore, if a network is partitioned after DA, the latency $T_{latency}$ should, at least, be greater than the sum of $T_{DAlatency}$ and C_{Quote}. However, two partitions may not be able to merge immediately after DA finishes since the two partitions may not reach each other at that time. This creates more delay in the attestation latency. Therefore, with AM, the total latency for the global key agreement is changed to:

$$T_{latency} \geq T_{DAlatency} + C_{Quote} = 2.90 \text{ s} \tag{6.4}$$

As we discussed before, many factors, such as physical partition, node movement, and message collision, can cause nodes in a network to be partitioned during DA. Therefore, our estimation of $T_{latency}$ only defines a lower bound for the attestation latency. Indeed, some situations in our simulations are able to reach global key agreement during DA without partitions. Their attestation latency is quite close to our analysis results reported here.

The discussion above applies to a network with many nodes. The special case is that there are only two nodes in a network just like the experiments we did in section 6.11.2. In such case, the latency for DA is about 1.15 seconds (in Table 6.5). Equation 6.2 indicates that the lower bound for DA latency is 2.20 seconds. However, that includes the timer for $T_{AttRply}$ and T_{AttKey}, which does not apply for the two-node case since the parent node knows

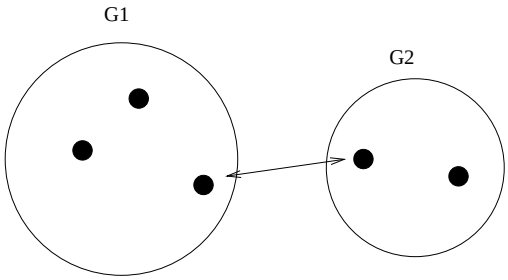


Figure 6.18: Two groups engaged in AM

it only has one child and the child node knows it is the leaf node of the tree. Therefore, they do not have to wait, but send attestation reply and new group key messages immediately. If we do not count these two timers (the value are showed in Table 6.2), the estimated DA latency from Equation 6.1 is 0.8 seconds. Considering that our implementation uses user-level NS2 simulator, not a kernel-level implementation, we think that our simulation and implementation results match well.

6.12.2 ATTESTED MERGER ANALYSIS

AM enables two groups with different group identities to merge into one group. It is composed of two processes: merge and rekey. The merge is based on groups' priorities. The rekey process is triggered by broadcast of RekeyDec messages in the group with lower group priority. In this section, we analyze AM to answer the following questions: (1) how good the group priority is, and (2) how well the rekey process works when there are many groups involved in AM at the same time.

The group priority number is an estimation of a group's size that is calculated during DA. Two groups merge based on their group priority, which has several advantages. First, the number is based on the size of the first two levels of the DA tree; therefore, it does reflect the size of the tree, to some degree. Giving priority to big groups with many nodes makes small groups merge into big groups. In addition, the way the group priority is calculated may be the best approach to estimating a group's size during DA since the root of DA tree

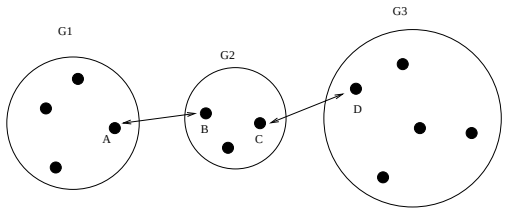


Figure 6.19: Three groups engaged in AM simultaneously with $PRI_{G2} < PRI_{G1}$ and $PRI_{G2} < PRI_{G3}$. G2 with the lowest group priority is merging with G1 and G3 at the same time, causing G2 splits and forming two new groups G1' and G3'. G1' and G3' are then merged into one group.

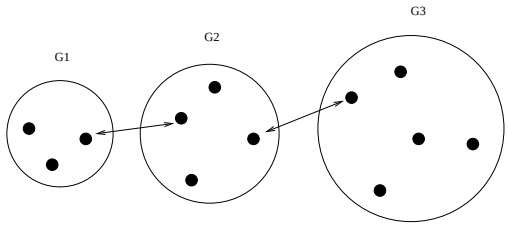


Figure 6.20: Three groups engaged in AM simultaneously with $PRI_{G1} < PRI_{G2} < PRI_{G3}$.

has no global view of all of the nodes involved. Second, once the group priority is calculated during DA, it is not changed until AM starts. If the group priority number is unique for each group, partitioned groups can quickly converge on the group with the highest group priority. There is no confusion for groups about deciding which groups they should merge with. If two or more groups have the same group priority number, the tie can be broken by picking the group with the higher group identity.

The rekey process of AM is so simple that it may cause confusion when many groups are involved in AM at the same time. We analyze this process for three simple cases first.

1. If only two groups are involved in AM, as shown in Figure 6.18, then the two groups can be merged simply by comparing their group priorities. The group with the lower group priority merges to the group with the higher priority.
2. A slightly more complicated case is the merging of three groups at the same time. The group with the lowest group priority may engage in AM with two other groups with different group identities and keys simultaneously. This may cause the group in the middle to split, as shown in Figure 6.19. In that illustration, G2 has the lowest group priority. The nodes in G2 engage in AM with G1 and G3 at the same time. Since $PRI_{G2} < PRI_{G1}$ and $PRI_{G2} < PRI_{G3}$, part of the nodes in G2 are merged with G1 and others are merged with G3 since the RekeyDec messages are broadcasted from the two groups G1 and G3 to G2 at the same time. Thus two new groups are formed. These can then be merged into one group as described above.
3. Three groups may merge with the group in the middle not having the lowest group priority. As shown in Figure 6.20, the three groups with $PRI_{G1} < PRI_{G2} < PRI_{G3}$ are merging. The merging process may vary according to the timing difference between the two merges.
    a. Suppose the merges in the two AM finish at the same time, meaning that node A in G1 and node C in G2 promiscuously unicasts the RekeyDec in their groups at the same time. Since the nodes in G1 receive the RekeyDec from node A first, they upgrade their group identity and key to G2 first. Therefore, when the nodes originally in G1 receive the RekeyDec from node B in G2, they upgrade their group identity and key to G3 again since they belong to G2 now. In this case, two AM can

merge three groups to one group.

b. Suppose that node B receives the RekeyDec first before it finishes merging with G1. Node B can upgrades its group identity and key to G3 first, and keep the merging state with node A in G1. When it is time for node B to send its merge reply to node A in G1, B then sends the new group identity and key. Therefore, the nodes in G1 can directly upgrade their group identity and key to G3.

AM becomes more complicated when there are more than three groups engaged in AM at the same time. However, the above analysis still applies in such a situation. Suppose there are n partitions with each identified by G_i. There is always one group (e.g., G_j) with the highest group priority. If the n groups are involved in AM at the same time, the groups around G_j will be merged to G_j. In addition, according to the analysis in the third case above, more groups than the number of neighbor groups of G_j can be merged to G_j during the one-time AM. Therefore, the number of nodes in G_j can keep increasing. And in the end, all the nodes in other groups will be able to merge into G_j. This has been demonstrated in our simulation experiments. In fact, a comparison of Figures 6.7 and 6.11 shows that the AM works well even when a network has many partitions.

6.12.3 BANDWIDTH OVERHEAD

The bandwidth overhead caused by AdHocSec is mainly from DA and AM messages. Of all the messages, AttRply has the greatest impact, since it contains a lot of information for attestation and its length is around 5k bytes.

Without considering message loss due to collision and retransmission, the messages used for DA and AM are proportional to the number of nodes in the network. If the number of nodes in the network is n, then the number of messages for each type of DA message, including AttReq, AttRply, AttKey, should be n. For AM, only two MrgReq messages and two MrgRply messages are needed. The number of RekeyDec messages is the same as the number of nodes in the lower priority group.

However, message collision and loss due to nodes' movement cause the use of NAK and message retransmission. That is why more messages than expected are used in Figure 6.12.

The number of MrgReq messages is usually much more than MrgRply messages, because more than one node in a group may propose to merge with nodes in another group.

From Figures 6.10 and 6.9, we can see that the bandwidth overhead demands caused by DA and AM are endurable. Furthermore, the overhead is a one-time burst. After DA and AM finish, there is no longer any such demand.

6.12.4 IMPACT OF ADHOCSEC ON HIGHER LEVEL PROTOCOLS AND APPLICATIONS

The impact of AdHocSec on higher level protocols and applications should be negligible. In fact, after DA and AM, all of the nodes in a network share a network secret key, which is used to authenticate the messages among nodes. The only overhead AdHocSec adds is the encryption, decryption, and message integrity code calculation and verification. As both the benchmark experiment in [72] and our own simulation experiments (as shown in Figure 6.13 and 6.14) indicate, AdHocSec brings little overhead to the up-layer routing protocols and network applications.

6.13 SECURITY ANALYSIS

In this section, we discuss the advantage and disadvantage of our solution for the ad hoc network security. Our approach adds a secure AdHocSec layer between the network and data link layers (DLL) of the network protocol stack, and enforces attestation in this layer. The AdHocSec layer is transparent to both its upper layer (network layer) and its lower layer (DLL). Therefore, the addition of the AdHocSec layer does not affect the function of its upper and lower layers. Network layer protocols, user-level applications, and MAC layer protocols require no change.

The advantages of AdHocSec layer to protect network's upper layers lie in three folds. First, it protects the network layer protocols, including the *original* routing protocols and ARP. This allows the network layer to use the *regular* routing protocols (e.g., DSR, AODV,

DSDV) rather than the secure ones (e.g., Ariadne, SAODV, SEAD, ARAN), yet maintains system security. Second, AdHocSec protects user-level applications. Third, AdHocSec solves the node selfishness problem.

The original routing protocols were designed without consideration of the harmful environment; thus, they are subject to many different types of attacks. As summarized in sections of other papers (e.g., ARAN [35] and Ariadne [13]), attackers can modify routing information (e.g., route sequence number, hop count), impersonate other nodes, and fabricate false routing messages, causing routing disruption and resource consumption. Attackers may engage in creating a variety of routing disruptions, including routing loops, black holes, gray holes, routing detours, network partitions, and rushing attacks. Their resource consumption attacks may involve the injection of extra data or control packets, which consume network bandwidth. All of these attacks are due to malicious or compromised nodes in the network.

AdHocSec enables the secure use of the original routing protocols, thus replacing the secure routing protocols. Compared with the secure routing protocols, AdHocSec has many advantages. First, all of the designed secure routing protocols primarily aim to prevent malicious or compromised nodes from disrupting and injecting routing messages by using cryptographic techniques authenticating the routing messages. The existence of the AdHocSec layer with attestation not only prevents such bad nodes from joining the network, but it is also able to disconnect any nodes that become compromised after attestation. Since our modified OS is able to maintain the required security policy on such nodes, any attempt to violate the required security policy will be prevented and cause the node to be excluded from the protected network. Therefore, malicious or compromised nodes cannot participate in route discovery, maintenance, and recovery. Second, AdHocSec further protects the routing messages through encryption and authentication, which guarantees their integrity and confidentiality. Third, AdHocSec is not specifically designed for any single original routing protocol; thus, it can protect all of them. However, most secure routing protocols proposed so far do not apply to all of the original routing protocol. In addition, AdHocSec has a better shared key generation method. Most secure routing protocols assume the existence of key management system.

AdHocSec provides extra protection for user-level network applications and some important network protocols, such as domain name service (DNS). Many user-level applications do not include any security approaches to protect their communications. Since AdHocSec authenticates and encrypts every data packet, it is able to provide integrity and authentication to these vulnerable user-level application packets. As a result, malicious nodes are not able to eavesdrop or inject data packets. In addition, the protection AdHocSec provides to DNS at the user level and ARP in the network layer together thwarts many possible attacks, such as MITM, on the DNS and ARP protocols.

AdHocSec is also able to solve the node selfishness problem. As discussed in section 6.2, the problem is that some nodes act selfishly and do not to forward packets on other nodes' behalf, usually in order to save their power. Because of this problem, the network cannot operate normally. Since AdHocSec enforces attestation, the ability to verify a node configuration, it can prevent such selfish nodes from joining the network. In this aspect, AdHocSec is able to replace some functions provided by earlier node-cooperation enforcement approaches such as CONFIDENT, Nuglets, etc.

AdHocSec has security limitations as well, however. Due to its location between the network and data link layers, the security features provided by AdHocSec only protect the network upper layers, and not the lower layers. Therefore, any attacks against the MAC and physical layers cannot be solved by AdHocSec. For example, AdHocSec is not able to prevent wormhole attack [68], which is a more subtle type of routing disruption attack. Two colluding nodes (e.g., A and B) linked through a private network can forward routing packets between them without modification. Thus, a node that receives a routing packet from B may think it receives the packet directly from another node near A. Wormhole attacks cause nodes in a network to have different visions of the routing topology, which disrupts routing. AdHocSec is not able to identify malicious nodes attacking networks using wormholes. Therefore, other approaches (e.g., packet leash [68]) are needed to prevent wormhole attacks. In addition, AdHocSec is not able to prevent some denial of services attacks at the MAC or physical layers, such as the injection into the network of packets by a node with powerful transmitters. Even though such nodes cannot pass attestation, attested nodes may busily receive such packets and not be able to process packets from attested nodes. This kind of attack is a hard problem

to solve.

There also could be some attacks directly against AdHocSec. One possible attack is the disruption of the initial attestation among the nodes in a network. At the beginning of attestation, nodes send AttReq messages to solicit attestation. Since the network shared secret key is not derived yet at this point, attackers can use malicious nodes to send AttReq messages into the network. As a result, malicious nodes can be included in the attestation graph. Though malicious nodes cannot pass attestation by other nodes, their existence could lead to the partition of nodes in the network after DA. AdHocSec provides some approaches to overcome this problem. First, the use of alternative parents in DA may help reduce partitions caused by malicious nodes, since alternative parents cause nodes to have more connections during DA. Furthermore, each node maintains a blacklist, which can help trusted nodes identify malicious nodes. Although malicious nodes cannot be identified during the first time, they can be recognized later on. Finally, AM can help the partitions to be merged later. However, such attacks could cause the latency of the global key agreement to be increased, since malicious nodes may cause partitions.

Other limitations to AdHocSec are the use of TPM, bugs found in the TCB list, and possible hardware attacks. First, as we discussed before, AdHocSec replies on TPM. Therefore, a node without TPM cannot take its advantage. Furthermore, the TPM has to be certified by a known certificate authority since the TPM quote needs a certified identity key to be authenticated by another party. This may cause some limitation for its use. Second, the attestation requires a TCB list. We assume that each component in the TCB list is trustworthy. However, a bug in one component of the TCB list may breach the security. Therefore, the TCB list needs to re-certified and updated as soon as a bug is found. Third, we assume that the TPM is not breakable and hardware attack is nearly impossible. However, if an attacker breaks the PC platform and uses some ways to steal the confidential in the TPM, the attacker will be able to attack the network.

6.14 CHAPTER SUMMARY

Many attacks against ad hoc networks, such as have been discussed in this chapter, require a maliciously configured node to join the attacked network. AdHocSec is a novel defense against such attacks. AdHocSec uses secure coprocessors and operating system mechanisms to guarantee that only nodes with trusted configurations can join a secure network and that any possible compromise of a node configuration causes the node to leave the network. Implemented between the data link and network layers, AdHocSec can transparently secure routing, forwarding, and other protocols layered on top of it, without modifying them. In this project, we contribute several efficient algorithms to help nodes perform attestation in ad hoc networks. Simulations demonstrate that AdHocSec imposes little overhead–much less, for example, than Ariadne, an efficient secure ad hoc routing protocol. Since AdHocSec's nodes authenticate each other's configuration, not identity, it is not as susceptible to fraud by malicious nodes. Innovative use of certain operating system features greatly simplifies AdHocSec's key management. In the vast majority of simulated cases conducted with 50 nodes, global trust in each others' configuration and key agreement was achieved in less than 8 s. After key agreement, AdHocSec imposes negligible overhead. We believe that AdHocSec is a valuable new approach for securing ad hoc networks.

7.0 CONCLUSION

Many attacks on network security are due to malicious or compromised nodes inside a network. This dissertation investigates such problems in enterprise and ad hoc networks and presents solutions for defending against them.

Enterprises increasingly suffer from damage caused by insider attacks. Existing secure solutions (e.g., firewalls, IDSs, anti-virus software) are generally biased to defend against attacks which originate outside the network; thus, they are not able to effectively detect and defend against insider attacks. Attackers can get into enterprise networks by compromising computers inside a network or by using compromised or malicious computers brought into the network by employees. Existing access control technologies, such as 802.1x and IKE, can only authenticate users' identities and not their client systems. Newer access control approaches, such as NAP and NAC, try to validate the configuration of client computers when users are first accessing the network. However, such approaches are weak against malicious users who may modify the initially-verified configuration without being detected.

Ad hoc networks have even more security problems, most of which are caused by malicious or compromised nodes. Most secure routing protocols (e.g., SAODV, Ariadne, SEAD, SRP, ARAN) were designed to prevent routing disruption and resource consumption caused by malicious or compromised nodes. They are insufficient for dealing with many security issues, including the node cooperation enforcement problem, which has been studied by many researchers and is another type of attack by malicious or compromised nodes.

This dissertation provides a hardware-based secure solution, which relies on attestation to verify a node's configuration before allowing the node to access a protected network. Attestation, provided by the TPM secure coprocessor, is the process of vouching for the accuracy of information. Therefore, this solution is able to detect and defend against malicious

or compromised nodes.

We apply such solutions to improve the security of both enterprise and ad hoc networks. Our solution integrates operating systems and network protocols. With modifications to existing OS and the differing integration of attestation in the network access control technologies of enterprise and ad hoc networks, our solution provides strong protection for both kinds of networks. Our approach can effectively protect the two networks by preventing attackers or compromised nodes from gaining network access, and enforcing such security policies constantly on nodes that already have access. Therefore, any nodes that gain network access initially, but violate the required security policy later, can be detected and excluded from the protected network.

While implementing our solutions, this dissertation made several contributions to operating systems. The OS is enhanced to be able to support attestation and enforce network access policies on client nodes. We contribute several techniques, including TCB prelogging, secure association root-tripping, and sealing-free attestation. These techniques convert a traditional OS into a system that can support attestation and help enforce security policies on the network.

This dissertation also makes many contributions to network protocols. First, a secure attestation protocol BKA is designed, which can adapt to different network environments. Second, for enterprise networks, we integrate BKA with existing access control technologies (802.1x and IKE) seamlessly, thus preventing malicious or compromised nodes using different network access methods from getting into a protected network. We believe our work with ad hoc networks is the first approach to apply attestation to enhance network security. In order to add attestation to the network, we designed several algorithms and protocols, including BBKA, DA, AM, and the proposed AdHocSec layer, which is added into a node's network stack to enforce attestation.

In order to demonstrate and evaluate the proposed solution, this dissertation has required many simulations and implementations. We implement the designed OS enhancement techniques in FreeBSD and modify the existing network access control protocols in both the client and server sides. We run real experiments to evaluate the impact of our work on enterprise networks. The experiment results show that our solution has little overhead. We

run many simulations of ad hoc networks and compare our solution with existing secure routing protocols. We also implement our work for ad hoc network in a real system with a Linux kernel. Both simulations and implementation demonstrate that our solution performs very well.

8.0 FUTURE WORK

Network and system security are important fields of exploration. In this dissertation, we provide a hardware-based solution, using TPM secure coprocessors, to improve the security of both enterprise and ad hoc networks by hardening both the operating system and network protocols. In the following, we discuss some promising extensions of our work to enhance network and system security.

TPM-based hardware solution can be applied in other networks or applications (e.g., peer-to-peer networks, distributed computing), besides enterprise and ad hoc networks, to improve their security. Attestation of nodes' configurations can help prevent malicious or compromised nodes from harming these networks or applications as well. In peer-to-peer networks or distributed computing, BBKA can be used to improve the performance of attestations.

One extension of our work in operating systems is to continue researching operating systems in terms of security, privacy, and flexibility. In this dissertation, we modified an OS such that it is able to support attestation and enforce network-required access policies. This requires the client system to expose all of the important information to the network, which may violate the privacy issue concerns of many users. In addition, users have no control over the inflexible system they are using. Therefore, a better approach may be needed to enable a system to have the required features to fulfill attestation and policy enforcement, while providing flexibility and maintaining privacy. We are going to investigate how such a system can be built. One of the approaches may use the virtual machine technology (e.g., xen [73]).

Another extension of our work in operating systems is to investigate how to enhance OS security by combining the power of TPM with other system access control approaches, such

as mandatory access control or role-based access control [74]. DTE [75] (Domain and Type Enforcement) is one kind of mandatory access control that divides a system into different domains and mandates the access control in each domain. The use of TPM may further consolidate DTE and provide strong protection for domains. DTE has Linux implementations (e.g., [76], [77]), which may facilitate the research in this direction.

This dissertation did not provide a solution to how the network can help attested nodes inside an enterprise network to update software when a newer version or a security patch is released. We also did not address how a client should update its configuration to conform to the attestation requirement. Therefore, as the continuation of this dissertation research, we may investigate a patch management system [78], [79] for enterprise networks.

As the continuation of our work for ad hoc network security, we will research how to defend against attacks at the MAC layer. As we discussed in section 6.13, AdHocSec is not able to defend against attacks such as wormholes [68], since they attack the layer lower than the AdHocSec. One possible approach is to investigate putting attestation at the MAC layer.

In the field of security, many topics can be explored. In the future, besides working on network and system security, I am also interested in research on software vulnerability and security. I would like to explore how attackers exploit software vulnerabilities and use them for attacking and, correspondingly, to identify what approaches are best for defending against such attacks. In addition, any research that relates to my past projects (e.g., [54], [80], [81], [82], [83], [84], [85]) may be investigated.

BIBLIOGRAPHY

[1] S. Saroiu, S. D. Gribble, and H. M. Levy, "Measurement and analysis of spyware in a university environment," in *First Symposium on Networked Systems Design and Implementation (NSDI'04)*, San Francisco, CA, 2004.

[2] Microsoft, "Network access protection." [Online]. Available: http://www.microsoft.com/windowsserver2003/technologies/networking/nap/default.mspx

[3] CISCO, "Network admission control." [Online]. Available: http://www.cisco.com/en/US/netsol/ns466/networking_solutions_sub_solution_home.html

[4] IETF, "Mobile ad-hoc networks (MANET)." [Online]. Available: http://www.ietf.org/html.charters/manet-charter.html

[5] S. Buchegger and J.-Y. L. Boudec, "Performance analysis of the CONFIDANT protocol (Cooperation Of Nodes: Fairness In Dynamic Ad-hoc NeTworks)," in *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, 2002, pp. 226–236.

[6] P. Michiardi and R. Molva, "CORE: a COllaborative REputation mechanism to enforce node cooperation in mobile ad hoc networks," in *CMS'2002, Communication and Multimedia Security 2002 Conference*, September 2002.

[7] Trusted Computing Group, "Trusted computing platform alliance (TCPA) main specification version 1.1b." [Online]. Available: https://www.trustedcomputinggroup.org/downloads/Main_TCG_Architecture_v11b.zip

[8] Trusted Computing Group. [Online]. Available: https://www.trustedcomputinggroup.org/home

[9] T. Dierks and C. Allen, "The TLS protocol version 1.0," IETF RFC 2246, January 1999. [Online]. Available: ftp://ftp.rfc-editor.org/in-notes/rfc2246.txt

[10] IEEE, "Port-based network access control, 802.1x Std," [Online] Available: http://standards.ieee.org/getieee802/download/802.1X-2001.pdf, 2001.

[11] S. Kent and R. Atkinson, "Security architecture for the Internet protocol," IETF RFC 2401, November 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2401.txt

[12] D. Harkins and D. Carrel, "The Internet key exchange (IKE)," IETF RFC 2409, November 1998. [Online]. Available: http://www.faqs.org/rfcs/rfc2409.html

[13] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne:: a secure on-demand routing protocol for ad hoc networks," in *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM Press, 2002, pp. 12–23.

[14] Y.-C. Hu, D. B. Johnson, and A. Perrig, "Sead: Secure efficient distance vector routing for mobile wireless ad hoc networks," in *WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*. IEEE Computer Society, 2002, pp. 3–13.

[15] Trusted Computing Group, "TNC architecture." [Online]. Available: https://www.trustedcomputinggroup.org/downloads/specifications/TNC_Architecture_v1_0_r4.pdf

[16] Microsoft, "Next generation secure computing base," July 2003. [Online]. Available: http://www.microsoft.com/technet/security/news/ngscb.mspx

[17] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: a virtual machine-based platform for trusted computing," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003, pp. 193–206.

[18] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and implementation of a tcg-based integrity measurement architecture," in *Proceedings of the 13th Usenix Security Symposium*, August 2004.

[19] J. Marchesini, S. Smith, O. Wild, J. Stabiner, and A. Barsamian, "Open-source applications of tcpa hardware," in *Proceedings of 20th Annual Computer Security Applications Conference*, December 2004.

[20] Trusted Computing Group, "TNC IF-IMC specification." [Online]. Available: https://www.trustedcomputinggroup.org/downloads/specifications/TNC_IFIMC_v1_0_r3.pdf

[21] Trusted Computing Group, "TNC IF-IMV specification." [Online]. Available: https://www.trustedcomputinggroup.org/downloads/specifications/TNC_IFIMV_v1_0_r3.pdf

[22] Trusted Computing Group, "Trusted network connect to ensure endpoint integrity." [Online]. Available: https://www.trustedcomputinggroup.org/downloads/whitepapers/TNC_NI-collateral_10_may_(2).pdf

[23] Trusted Computing Group, "Open standards for integrity-based network access control." [Online]. Available: https://www.trustedcomputinggroup.org/downloads/whitepapers/Open_Standards_for_IntegrityBased_AccessControl.pdf

[24] NIST, "Secure hash standard," April 1995. [Online]. Available: http://www.itl.nist.gov/fipspubs/fip180-1.htm

[25] B. balacheff, liqun chen, siani pearson, david plaquin, and graeme proudler, *Trusted Computing Platforms: TCPA Technology In Context*, S. Pearson, Ed. Prentice Hall PTR, July 2002.

[26] Trusted Computing Group, "TCG architecture overview." [Online]. Available: https://www.trustedcomputinggroup.org/groups/TCG_1_0_Architecture_Overview.pdf

[27] R. Anderson, "Cryptography and competition policy: issues with 'trusted computing'," in *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*. New York, NY, USA: ACM Press, 2003, pp. 3–10.

[28] R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn, "Attestation-based policy enforcement for remote access," in *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*. New York, NY, USA: ACM Press, October 2004, pp. 308–317.

[29] A. Palekar, D. Simon, J. Salowey, H. Zhou, G. Zorn, and S. Josefsson, "Protected eap protocol (peap) version 2," IETF Internet Draft. [Online] ftp://ftp.rfc-editor.org/in-notes/internet-drafts/draft-josefsson-pppext-eap-tls-eap-10.txt, October 2004.

[30] IEEE, "P802.1ae/d2.01-draft standard for local and metropolitan area networks: Media access control (mac) security," IEEE standard, October 2004.

[31] "Wi-Fi alliance." [Online]. Available: http://www.wi-fi.org

[32] IEEE, "P802.11i/d10.0-part11: Wireless medium access control (mac) and physical layer (phy) specifications: Amendment 6: Medium access control (mac) security enhancements," IEEE standard, April 2004.

[33] M. G. Zapata and N. Asokan, "Securing ad hoc routing protocols," in *WiSE '02: Proceedings of the ACM workshop on Wireless security*, 2002, pp. 1–10.

[34] P. Papadimitratos and Z. Haas, "Secure routing for mobile ad hoc networks," in *Proceedings of CNDS*, 2002.

[35] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields, and E. M. Belding-Royer, "A secure routing protocol for ad hoc networks," in *ICNP '02: Proceedings of the 10th IEEE International Conference on Network Protocols*. IEEE Computer Society, 2002, pp. 78–89.

[36] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," in *ACM Conference on Communications Architectures, Protocols and Applications, SIGCOMM '94, London, UK*, ACM. ACM, August 1994, pp. 234–244. [Online]. Available: http://people.nokia.net/charliep/txt/sigcomm94/paper.ps

[37] D. B. Johnson, D. A. Maltz, and J. Broch, "DSR: the dynamic source routing protocol for multihop wireless ad hoc networks," pp. 139–172, 2001.

[38] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*. Washington, DC, USA: IEEE Computer Society, 1999, pp. 90–100.

[39] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "The tesla broadcast authentication protocol," *RSA CryptoBytes*, vol. 5, no. Summer, 2002.

[40] S. Buchegger and J.-Y. L. Boudec, "Nodes bearing grudges: Towards routing security, fairness, and robustness in mobile ad hoc networks," in *Proceedings of the Tenth Euromicro Workshop on Parallel, Distributed and Network-based Processing*. IEEE Computer Society, January 2002, pp. 403–410.

[41] J. R. Douceur, "The sybil attack," in *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. London, UK: Springer-Verlag, 2002, pp. 251–260.

[42] L. Buttyán and J.-P. Hubaux, "Nuglets: a virtual currency to stimulate cooperation in self-organized ad hoc networks, Tech. Rep. DSC/2001, 2001.

[43] H. Yang, X. Meng, and S. Lu, "Self-organized network-layer security in mobile ad hoc networks," in *WiSE '02: Proceedings of the 3rd ACM workshop on Wireless security*. New York, NY, USA: ACM Press, 2002, pp. 11–20.

[44] Trusted Computing Group, "TPM software stack (TSS) specifications." [Online]. Available: https://www.trustedcomputinggroup.org/specs/TSS

[45] E. Felten, "Understanding trusted computing: will its benefits outweigh its drawbacks," *IEEE Security and Privacy*, vol. 1, no. 03, pp. 60–62, 2003.

[46] IBM, "TCPA resources." [Online]. Available: http://www.research.ibm.com/gsal/tcpa/

[47] GNU GRUB, "GRand Unified Bootloader." [Online]. Available: http://www.gnu.org/software/grub/

[48] Trusted Computing Group, "TCG PC client specific implementation specification for conventional BIOS." [Online]. Available: https://www.trustedcomputinggroup.org/specs/PCClient/TCG_PCClientImplementationforBIOS_1-20_1-00.pdf

[49] D. McDonald, C. Metz, and B. Phan, "PF_KEY key management API, version 2," IETF RFC 2367, July 1998. [Online]. Available: ftp://ftp.rfc-editor.org/in-notes/rfc2367.txt

[50] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman, "Linux security modules: General security support for the linux kernel," in *Proceedings of the 11th USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, August 2002, pp. 17–31.

[51] Tripwire.org. [Online]. Available: http://www.tripwire.org/

[52] S. D. Schoen, "EOF - give TCPA an owner override," accessed on July 26, 2006. [Online]. Available: http://www.linuxjournal.com/article/7055

[53] S. D. Schoen, "Trusted computing: Promise and risk," accessed on July 26, 2006. [Online]. Available: http://www.eff.org/Infrastructure/trusted_computing/20031001_tc.php

[54] H. Xia and J. C. Brustoloni, "Hardening web browsers against man-in-the-middle and eavesdropping attacks," in *WWW 2005*, Chiba, Japan, May 14-18 2005.

[55] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, November 1976.

[56] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-hashing for message authentication," RFC 2104, IETF, 1997. [Online]. Available: ftp://ftp.rfc-editor.org/in-notes/rfc2104.txt

[57] S. Berger, R. Caceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn, "vTPM: Virtualizing the trusted platform module," IBM, Tech. Rep., Feb 2006.

[58] IEEE, "Wireless lan medium access control (mac) and physical layer (phy) specification. 802.11 std." IEEE standard, 1999.

[59] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, "Extensible authentication protocol (EAP)," IETF Intternet Draft, [Online] http://www.ietf.org/rfc/rfc3748.txt, June 2004.

[60] B. Aboba and D. Simon, "Ppp eap tls authentication protocol," RFC 2716. [Online] http://www.ietf.org/rfc/rfc2716.txt, October 1999.

[61] P. Funk and S. Blake-Wilson, "Eap tunneled tls authentication protocol (EAP-TTLS)," IETF Internet draft, [Online] http://www3.ietf.org/proceedings/02jul/I-D/draft-ietf-pppext-eap-ttls-01.txt, February 2002.

[62] D. Maughan, M. Schertler, M. Schneider, and J. Turner, "Internet security association and key management protocol (ISAKMP)," IETF RFC 2408, November 1998. [Online]. Available: http://rfc.net/rfc2408.html

[63] KAME, "Racoon IKE daemon." [Online]. Available: ftp://ftp.kame.net/pub/kame/misc/

[64] FreeRADIUS, "The FreeRADIUS server project." [Online]. Available: http://www.freeradius.org/

[65] Open1x, "Open source implementation of IEEE 802.1x." [Online]. Available: http://www.open1x.org/

[66] J. Moy, "OSPF version 2," IETF RFC 2328, [Online] http://http://www.faqs.org/rfcs/rfc2328.html, April 1998.

[67] C. Hedrick, "Routing Information Protocol," IETF 1058, June 1998. [Online]. Available: http://www.faqs.org/rfcs/rfc1058.html

[68] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Packet leashes: A defense against wormhole attacks in wireless networks," in *Proceedings of IEEE Infocomm 2003*, April 2003.

[69] NS-2, "The network simulator - ns-2." [Online]. Available: http://www.isi.edu/nsnam/ns/

[70] A. K. Saha, K. A. To, S. PalChaudhuri, S. Du, and D. B. Johnson, "Physical implementation and evaluation of ad hoc network protocols using unmodified simulation models," in *ACM SIGCOMM Asia Workshop*, Beijing, China, April 2005.

[71] J. LeBoudec and M. Vojnovic., "Perfect simulation and stationarity of a class of mobility models," in *Proc. INFOCOM'2005*. IEEE, March 2005.

[72] Crypto++ Library 5.2.1. [Online]. Available: http://www.eskimo.com/~weidai/benchmarks.html

[73] "The xen virtual machine monitor." [Online]. Available: http://www.cl.cam.ac.uk/Research/SRG/netos/xen/

[74] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, 1996.

[75] L. Badger, D. F. Sterne, D. L. Sherman, K. M. Walker, and S. A. Haghighat, "Practical domain and type enforcement for unix," in *SP '95: Proceedings of the 1995 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, June 1995, p. 66.

[76] S. E. Hallyn and P. Kearns, "Domain and type enforcement for linux," in *4th Annual Linux Showcase and Conference*, October 2000.

[77] S. Smalley, C. Vance, and W. Salamon, "Implementing selinux as a linux security module." [Online]. Available: http://www.nsa.gov/selinux/info/docs.cfm#papers

[78] J. Chan, "Essentials of patch management policy and practice." [Online]. Available: http://www.patchmanagement.org

[79] F. M. Nicastro, "Security patch managment," internet Network Services. [Online]. Available: http://www.ins.com

[80] H. Xia and J. C. Brustoloni, "Improving the usability of web browser security," in *Symposium on Usable Privacy and Security (SOUP)*, Pittsburgh, PA, July 6-8 2005.

[81] H. Xia and J. C. Brustoloni, "Secure and flexible support for visitors in enterprise wi-fi networks," in *GLOBECOM 2005*. St. Louis, MO: IEEE, Nov. 29 - Dec. 1 2005.

[82] H. Xia and J. C. Brustoloni, "Detecting and blocking unauthorized access in Wi-Fi networks," in *NETWORKING 2004*, Athens, Greece, May 9-14 2004.

[83] H. Xia and J. C. Brustoloni, "Virtual prepaid tokens for Wi-Fi hotspot access," in *LCN 2004*, Tampa, Florida, November 2004.

[84] H. Xia, J. Kankana, and J. C. Brustoloni, "Using secure coprocessors to protect the access to enterprise networks," in *NETWORKING 2005*, Waterloo, Canada, May 2-6 2005.

[85] H. Xia, J. Kankana, and J. C. Brustoloni, "Enforcement of security policy compliance in virtual private networks," in *SSI 2005*, Sao Jose dos Campos, Brazil, Nov. 8-11 2005.