The Pennsylvania State University

The Graduate School

Department of Information Sciences and Technology

A STUDY OF SELECTED SECURITY ISSUES IN WIRELESS NETWORKS

A Thesis in

Information Sciences and Technology

by

Qijun Gu

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2005

The thesis of Qijun Gu has been reviewed and approved* by the following:

Chao-Hsien Chu
Associate Professor of Information Sciences and Technology
Thesis Co-Adviser
Co-Chair of Committee

Peng Liu
Assistant Professor of Information Sciences and Technology
Thesis Co-Adviser
Co-Chair of Committee

Lee Giles
Professor of Information Sciences and Technology

Tracy Mullen
Assistant Professor of Information Sciences and Technology

Guohong Cao
Associate Professor of Computer Science and Engineering

Joseph M. Lambert
Associate Professor of Information Sciences and Technology
Associate Dean, Graduate Programs in Information Sciences and Technology

*Signatures are on file in the Graduate School

Abstract

Wireless networks are becoming more and more important in our lives. However, due to the fact that wireless communication is on air, an adversary can easily eavesdrop on packets, imitate other wireless devices, forge packets, and so on. Hence, enhancing the security in wireless networks has become of vital importance. In this thesis, we mainly study two security aspects of wireless networks. One is service confidentiality and access control, that is to ensure only legitimate users can access service data according to their privileges. The other is the prevention of denial-of-service (DoS) attacks, that is to prevent injection of junk packets in wireless networks. To this end, we investigate the following two topics: key management in wireless broadcast services and hop-by-hop source authentication in wireless ad hoc networks.

Wireless broadcast is a convenient and effective approach for disseminating data to a number of users. To provide secure access to broadcast data, key-based encryption ensures that only users who own valid keys can decrypt the data. Regarding various subscriptions in broadcast services, a key management system for distributing new keys efficiently and securely is in great demand. Hence, we propose a key management scheme, namely KTR, to address this need. KTR uses a shared key structure which exploits the overlapping relationships among different subscriptions and allows multiple programs to share a single key tree so that users who subscribe to these programs can manage less keys. KTR further reduces rekey cost by identifying the minimum number of keys that must be changed to ensure broadcast security.

Wireless ad hoc networks have very limited network resources and are thus susceptible to attacks that focus on resource exhaustion, such as the injection of junk packets. These attacks cause serious denial-of-service via wireless channel contention and network congestion. Although ad hoc network security has been extensively studied, most previous work focuses on secure routing and cannot prevent attackers from injecting a large number of junk data packets into a route that has been established. We propose an on-demand hop-by-hop source authentication protocol, namely SAF, to defend against this type of packet injection attacks. The protocol can either immediately filter out injected junk packets with very high probability or expose the true identity of an injector. Unlike other forwarding defenses, this protocol is designed to fit in the unreliable environment of ad hoc networks and incurs very lightweight overhead in communication and computation.

In summary, this study presents two security aspects of wireless networks. First, a key management scheme is proposed to address secrecy and efficiency in broadcast services, where keys are used for service confidentiality and access control. The proposed approach performs better in terms of communication overhead, computation load and storage in mobile devices. Second, a hop-by-hop source authentication approach is proposed to prevent the packet injection attack, which is a type of denial of service attack based on resource exhaustion. This approach provides source authentication in the unreliable environment in ad hoc networks without interfering with the delivery of legitimate packets.

Table of Contents

List of Tables

List of Figures

Acknowledgments

The thesis cannot be finished without my wife. She shared my happiness and sadness throughout the 4-year hard work. She gave me great supports, and made my life colorful and meaningful.

I am indebted to my thesis advisors, Dr. Chao-Hsien Chu and Dr. Peng Liu for the guidance and the encouragement. They have greatly helped me to overcome various difficulties and frustrations, and developed my academic thinking and research skills. It is really fortunate to work with them during my time here at Pennsylvania State University.

I am grateful to my doctoral committee members, Dr. Lee Giles, Dr. Tracy Mullen, and Dr. Guohong Cao for spending time on my committee and providing insightful suggestions on the thesis. I am also grateful to Dr. Wang-Chien Lee and Dr. Sencun Zhu for their constructive comments on my research.

I thank my lab mates, Kun Bai, Yoon-Chan Jhi, Kameswari Kotapati, Fengjun Li, Lunquan Li, Udaiyanathan Muthukumar, Chi-Chun Pan, Hai Wang, and Tao Yang, for the discussion on research problems. I thank my friends as well as anyone not in this list that gave me help on my work and life.

Finally, I thank my parents and siblings. They gave me unconditional supports, even though they are far away from me.

The thesis was partially supported by NSF ANI-0335241, NSF CCR-0233324, Department of Energy Early Career PI Award, and NSA H98230-04-1-0224.

Chapter 1

Introduction

Wireless networks are widely used in a broad range of real life applications: military war field operations, emergency search-and-rescue deployments, data collection sensor networks, instantaneous meeting room applications and so on. Because of this, wireless networking is an extremely hot topic for research, with work covering the mobile applications and services to the physical signal processing technologies. One main characteristic of wireless networks is that its users share the same physical media, that is, they transmit and receive signals at the same frequency band. Because of this, wireless networks are subject to various misbehaviors and attacks. For example, a user can directly listen to other users communications which are within his radio range via wireless links, , A user can also occupy the frequency band for the purpose of suppressing transmissions from other wireless devices. Our study targets the security perspectives of wireless networks.

In this study, two types of wireless networks are considered: infrastructure networks and ad hoc networks. An infrastructure wireless network generally consists of a base station and mobile users. In the network, mobile users receive signals directly from the base station. If one user wants to communicate with another user, the base station will relay the data. An example of such a network is an 802.11 wireless LAN, where the access point acts as the base station. In an ad hoc wireless network, mobile users have a peer-to-peer relationship with the base station. When a user wants to communicate with another user, he generally asks some intermediate users

to relay his data to the destination, instead of relying on the base station. Hence, users are capable of communicating with each other through multiple hops without the support of a fixed infrastructure.

Due to the facts that all communication in a wireless network is carried on open air and that the mobile devices carried by users are not easy to manage, a wireless network is subject to a variety of threats. An attacker may illegally obtain access privileges and information of the services provided in the networks or may cause denial-of-service (DoS) attacks designed to prevent legitimate users from accessing the service. An attacker may abuse some special service or protocols to gain an advantage or cause disruption to the network. An attacker may exploit their privileges to obtain unauthorized services or information. In the following, we summarize the major threats of particular importance to this thesis.

First, wireless communication can be *eavesdropped* by attackers. When a packet is on air, an attacker with the proper equipment can overhear the packet without being detected. When the packet is delivered over multiple hops, the attacker can overhear the packet if he stays nearby the forwarding nodes, and thus has no need to stay in the route in order to obtain the packet. It is also hard for two wireless devices to trust each other. An attacker may *masquerade* as a legitimate system (a server or the base station) to get trust from a user and obtain confidential information. He can also masquerade as a legitimate user in order to obtain access privileges or confidential information from wireless services. *Violation of service integrity* could also be frequent in a wireless network. In a wireless ad hoc network, a packet delivered over multiple hops may be modified by an attacker if he is a node in the forwarding path of the packet. Users in wireless networks can also *repudiate* actions that they have committed. For example, an attacker can massively replay overheard packets to disrupt network traffic, but claim that these packets

are forwarded from their sources or retransmitted due to transmission failures. Also, wireless networks have limited resources, and thus are susceptible to *resource exhaustion* attacks. For example, the frequency band in a wireless network is much less than that in a wired network. Radio jamming is a simple but effective approach to exhaust bandwidth and block communication. Furthermore, a mobile device has limited energy and computational capability. Hence, an attacker may exhaust a mobile device by asking it to process large amounts of information. All of these threats draw our attention on wireless security and bring many challenges to defending wireless networks. In the following, we overview the major issues addressed in this study.

1.1 Motivations of this Study

In this thesis, we investigate two crucial security issues in wireless networks. The first is the key management for service confidentiality and access control in wireless broadcast services. The second is the hop-by-hop source authentication in ad hoc networks to prevent packet injection DoS attacks. In short, these two techniques are used to protect a wireless network from the security threats discussed above. They are crucial in providing secure and smooth services for wireless networks.

1.1.1 Key Management in Broadcast Services

Wireless broadcast is an effective way to deliver data to multiple users in a wireless network. However, since all data is broadcast on air in a wireless environment, all users can listen to the broadcast channel and log the broadcast data, regardless of which data a user subscribes to or whether the user subscribes to the service. If data is not encrypted, the service is open to the public, and a legitimate user can access data to which he does not subscribe. Hence, a secure

access control mechanism is required to protect the service and the users' privileges. Symmetric-key-based encryption provides secure data dissemination and service access. The broadcast data can be encrypted in a way that only users with valid keys can decrypt them. Thus, decryption keys are an effective means for access control in wireless broadcast services.

Key management for all users in broadcast services is crucial and complex due to several factors. First, a service may have many users. Sending a message of a new key to each user separately would bring tremendous traffic to the network if the number of users in the network is large. We can take the advantage of broadcast to distribute new keys in wireless channels to address this issue of scalability in key management. Second, the service may provide many programs, and each user may subscribe to an arbitrary set of programs. For the purpose of security, each program should have its own keys to encrypt program data, and each user should have only the keys for the subscribed programs. In this way, each user has their individual privileges to access the broadcast data. The challenge then is how to distribute new keys efficiently to different users in the broadcast environment while ensuring that no user can access keys and data beyond their privileges. Third, keys need to be changed when a user joins or leaves the wireless network. A user's privilege is not only which data the user can access but also how long the user can access specific data. At the same time, other users should not be interrupted due to the change of keys. Hence, a key management system should be flexible to adapt user subscription activities. Fourth, a user may subscribe to many programs. If the user holds different keys for different programs, it might be demanding for a user to handle many keys, especially when the storage, computational capability and power of mobile devices are limited. Hence, we need a system where a user handles as few keys as possible.

These challenging issues raise the critical problem of *how we can efficiently manage the keys when a user joins/leaves/changes the service without compromising security and interrupting the operations of other users.* In this study, we propose a key management approach that addresses this problem.

1.1.2 Packet Injection in Ad Hoc Networks

Because wireless bandwidth is limited and shared, resource exhaustion is an effective means of causing DoS in wireless networks. In ad hoc networks, an attacker may inject junk packets into legitimate routes. This can cause serious network congestion at the destination side as well as lead to severe wireless channel contention in the nearby nodes of the legitimate routes. Compared with other types of DoS attacks in ad hoc networks, the injection attack in general is easier to enforce since attackers can simply impersonate other nodes by spoofing the source addresses in junk packets. Hence, an analysis of injection attacks is needed for a good understanding, and a defense against junk packet injection is desired as well.

When injecting junk packets, an attacker may impersonate another user in order to hide himself from being detected and traced. Hence, the solution to prevent junk packet injection attacks is to enforce source authentication when forwarding data packets. When the source sends a data packet after discovering a route, it appends authentication information to the packet. An en route node only forwards those packets that are authenticated. In this way, only the data packets from the real source can go through the route and reach the destination. In particular, hop-by-hop source authentication is adopted as a necessary measure to ensure that an injected junk packet can be filtered out immediately. Otherwise (i.e., if end-to-end authentication is used), junk packets can affect nodes along the route until the destination discards it. If some

nodes do not verify the packets when forwarding them, an attacker is able to inject packets into the network via these nodes.

Although source authentication is effective in the prevention of packet injection attacks, it faces many practical challenges when being applied in an ad hoc network, of which the unreliability of these types of networks is the most critical. For example, packet loss may cause the loss of critical information for source authentication and thus make it impossible for en route nodes to verify the packets they receive. When a route in the network is changed, the authentication information carried in packets may not be useful because it is not designated to the new forwarding nodes in the new routes and thus authentication is not sustainable. The consequence of these problems of unreliability is that good packets will be discarded when source authentication approaches are adopted for defense purposes.

Hence, the problem we face is *how we can provide hop-by-hop source authentication in unreliable ad hoc networks to prevent attackers from injecting junk packets*. To address this problem, we propose a hop-by-hop source authentication protocol in forwarding data packets.

1.2 Contribution of this Study

1.2.1 Key Management in Broadcast Services

There are a great number of existing studies on key management in the literature. Many of them target multicast services and ask the users who belong to one group to handle a set of keys for their group. In broadcast services, a program is equivalent to a multicast group, and the users who subscribe to one program form a group. Because all programs are simultaneously broadcast on air, a user who subscribes to one program can easily listen to another program.

Applying the approaches in the literature, the broadcast server can assign a separate set of keys for each program and ask a user to handle only the sets of keys for the subscribed programs, thereby preventing a user from accessing any program to which he has not subscribed.

However, these approaches overlook a main feature in broadcast services: that many users subscribe to multiple programs simultaneously. In another words, programs overlap with each other in terms of users. Because existing approaches manage keys by separating programs, they turn to be demanding for the users who subscribe to many programs. Hence, this study contributes to the literature a new scheme (namely KTR) to better support subscriptions of multiple programs by exploiting the overlapping among programs. KTR let multiple programs share the same set of keys for the users who subscribe to these programs. KTR thus inherently enables users to handle fewer keys and reduces the demands of storage and processing power on resource-limited mobile devices.

Since multiple programs are allowed to share the same set of keys, a critical issue is how to manage shared keys efficiently and securely. We find that when keys need to be distributed to a user, it is unnecessary to change all of them. In many circumstances, when a user subscribes to new programs or unsubscribes to some programs, a large portion of keys that he will hold in his new subscription can be reused without compromising security. KTR is a novel approach for determining which keys need to be changed and for finding the minimum number of keys that must be changed. Hence, KTR efficiently handles the rekey of the shared keys and minimizes the rekey costs associated with possible subscriptions. Our simulations show that critical keys can be employed in logical key hierarchy schemes [53, 50] to improve their performance.

We notice that there are similar works for multi-group key management in the field of multicast security. As far as we know, the most similar work is in [49], where Sun *et al.* propose

a key tree structure similar to ours. However in [49], the focus is on reducing the redundancy of key trees given an organizational structure. Because users may subscribe to an arbitrary set of programs in broadcast services, the key structure in KTR is more flexible and depends on users subscriptions. In addition, KTR is unique in that it uses an approach to decide whether a key needs to be changed.

1.2.2 Packet Injection In Ad Hoc Networks

Ad hoc network security has been extensively studied. However, most of the previous work [18, 20, 62] focuses on secure routing. After a route is established, there is no authentication in forwarding data packets. As a result, an attacker can exploit this weakness to inject junk packets into routes. Although this type of attack has been recognized as a threat to ad hoc networks, it has not been systematically studied in terms of its impact on a network in regards to various attacking strategies and factors. Hence, we contribute to the literature a clear picture on the impact of packet injection attacks based on empirical studies and data analysis.

The attack we studied is different from packet dropping attacks and false data injection attacks in the literature. The intention of this attack is to disrupt traffic and cause denial of service by consuming network resources when normal nodes forward junk packets. Its attack impact is not only on the nodes nearby the attackers, but also on the areas that junk packets go through. In contrast, packet dropping attacks mainly disrupt the traffic that goes through the attackers. Packet injectors can also exploit flaws in defenses against false data injection attacks which mainly target influencing the decision making by providing false information. Since most authentication approaches only target data integrity, attackers can replay or retransmit authenticated packets for injection purposes.

The unreliability feature of ad hoc networks also brings many difficulties to defense systems. The forwarding protocol proposed in this study contributes to the literature a defense scheme against packet injection attacks. This protocol is especially designed to handle various problems in the forwarding procedure in an unreliable ad hoc network. It not only provides security against packet injection attacks, but also ensures the smooth delivery of legitimate data packets. The main difference between our protocol and the secure routing protocols lies in the design goals. The secure routing protocols are designed to secure the route discovery, whereas our protocol focuses on filtering and dropping junk packets injected into the routes that are established by those routing protocols. Therefore, our scheme and the secure routing protocols are complementary to each other.

Finally, we systematically analyzed and summarized various problems when applying source authentication in forwarding data packets in ad hoc networks. We also studied how attackers can misuse source authentication to cause other security issues. These discussions contribute to the literature a clear picture on the practical problems that current as well as future source authentication approaches need to take into consideration in their designs.

1.3 Thesis Outline

The thesis is organized as follows. In Chapter 2, we present the main research related to our studies. In Chapter 3, we propose the key management scheme for wireless broadcast services. In Chapter 4, we study the consequence of packet injection attacks under various factors. In Chapter 5, we propose the source authentication forwarding protocol for ad hoc networks. Finally, in Chapter 6, we summarize the thesis, and overview future works.

Chapter 3 is organized as follows. In Section 3.2, we describe the service model, present the related approaches on group key management, and discuss our design goals. In Section 3.3, KTR is proposed to fully exploit the service structure, and address the updating and distribution of shared keys. In Section 3.5, we present the results of simulations to illustrate the performance improvements of KTR. Finally, we conclude in Section 3.6.

Chapter 4 is organized as follows. In section 4.2, we overview packet injection attacks in ad hoc network, and overview the features of packet injection attacks targeting network wide congestion. In Section 4.3, the remote attacks are analyzed and simulated. In Section 4.4, the local attacks are analyzed and simulated. Finally, we conclude this study in section 4.5.

Chapter 5 is organized as follows. Section 5.2 presents the background on attack and defense approaches. In Section 5.3, we present the problems that an authentication protocol will face. Section 5.4 introduces the design overview, the detail, and security analysis of SAF. SAF is evaluated in Section 5.6 and this chapter concludes with Section 5.7.

Chapter 2

Related Works

In this chapter, we summarize the related research in the literature, and give an overview on the different perspectives these works have addressed. A more detailed discussion on the most related approaches and the rationale on why we need new ideas are presented in the corresponding chapters.

2.1 Group Key Management

Secure key management for wireless broadcast is closely related to secure group key management in networking. Mittra [31] proposed partitioning a group into multiple subgroups and organizing them into a hierarchy, in which users are the leaf nodes and group security agents are the non-leaf nodes. A more popular approach, i.e. logical key hierarchy (LKH), is proposed in [53, 50]. In LKH, a key tree is applied for each group of users who subscribe the same program. The root (top node) of the tree is the data encryption key (DEK) of the program. Each leaf (bottom node) in the tree represents an individual key of a user that is only shared between the system and the user. Other keys in the tree, namely key distribution keys (KDKs), are used to encrypt new DEKs and KDKs. A user in the tree only knows the keys along the path from the leaf of the user to the root of the key tree. When a user adds or quits a program, the system changes corresponding DEKs and KDKs, and other users use their known keys to decrypt new

keys (see examples in [53]). The operation to distribute new keys is called rekey. In our system, data and rekey messages are broadcast in the same broadcast channel to the users.

LKH is an efficient and secure key management for multicast services in that each user only needs to hold $O(log_2(n))$ keys for his group, and the size of a rekey message is also $O(log_2(n))$, where n is the number of group users. It is also a flexible key management approach that allows a user to join and leave the multicast group at any time. Many variations of LKH have been proposed. [26] proposes a combination of key tree and Diffie-Hellman key exchange to provide a simple and fault-tolerant key agreement for collaborative groups. [7] reduces the number of rekey messages, while [55] reduces message loss and thus lost keys. Balanced and unbalanced key trees are discussed in [53] and [32]. Periodic group re-keying is studied in [45, 59] to reduce the rekey cost for groups with frequent joins and leaves. Issues on how to maintain a key tree and how to efficiently place encrypted keys in multicast rekey packets are studied in [32, 59]. Moreover, the performance of LKH is also thoroughly studied [59, 46].

There are some other key management schemes in the literature for multicast and broadcast services. [5] used arbitrarily revealed key sequences to do scalable multicast key management without any overhead on joins/leaves. [56] proposed two schemes that insert an index head into packets for decryption. However, both of them require pre-planned eviction, which contradicts the fact that in pervasive computing and air data access a user may change subscriptions at any moment. In addition, [56] only supports a limited combination of programs. [29] proposed a scheme to yield maximal resilience against arbitrary coalitions of non-privileged users. However, the size (entropy) of its broadcast key message is large, at least $O(n)$ [24]. Zero-message scheme [11, 3] does not require the broadcast server to disseminate any message in order to generate a common key. But it is only resilient against coalitions of k non-privileged users, and requires

every user to store $O(klog_2(k)log_2(n))$ keys. Naor *et al.* [33] proposed a stateless scheme to facilitate group members to obtain up-to-date session keys even if they miss some previous key distribution messages. Although this scheme is more efficient than LKH in rekey operations, it mainly handles revocation when a user stops subscription. It does not efficiently support joins, which are crucial in our system. Finally, [41, 48] proposed self-healing approaches for group members to recover the session keys by combining information from previous key distribution information. Since this study does not focus on the recovery of lost keys, these schemes can be incorporated in our system in future works.

2.2 Source Authentication

In order to ensure data integrity, source authentication is mostly used. When the source sends a packet to the destination, it puts authentication information into the data packet. A receiving node only accepts the packet if it is authenticated. In this way, only the data packets from the real source can go through the route and reach the destination. Other than public key based digital signature which has unbearable computational demand on mobile nodes, several source authentication approaches exist in the literature. In the field of multicast source authentication, multiple receivers can verify whether the received data originated from the claimed source and was not modified en route. The recent research efforts [37, 54] in multicast source authentication focused on amortizing the cost of a digital signature over multiple packets. The source amortizes signature over the message digests of a block of packets, and the receiver can verify all the packets in a batch. Although this scheme has the lowest amortized packet overhead, it needs an en route node to receive the whole block for verification and does not tolerate any packet losses. Some researches proposed techniques that do tolerate packet loss by using expanded

graph [47], authentication chain [13], distillation code [25] or erasure code [38]. In general, they can only tolerate the loss of a few packets. However, a node may discard all packets in its routing buffer when it is turned down in an ad hoc network. Previous approaches are unable to sustain authentication in this situation.

Perrig *et al.* proposed TESLA [40] based on one-way key chain, which is also used in securing routing protocols [18, 20]. A one-way key chain is a sequence of keys $k_0, k_1, ..., k_n$, where $k_i \leftarrow H(k_{i+1})$ and $H(*)$ is a hash function. When the source sends data packets, it first computes such a chain, and then sends k_i with the i^{th} packet. A receiver can verify k_i by computing whether $k_{i-1} = H(k_i)$, where k_{i-1} is the key in the previous packet. To start the scheme, a sender uses a regular signature scheme to sign the initial key. All subsequent packets are authenticated through the one-way key chain. TESLA is efficient in computation and can tolerate the loss in the following data packets. However, this scheme requires that the source computes the chain in advance and that the nodes buffer the packets in order to verify them later. However, this scheme cannot guarantee to filter a forged packet instantly before other nodes to forward it to the next hop. Therefore, it is not suitable for this study.

Hop-by-hop source authentication [60, 65] has been considered as the necessary measure to ensure that an injected data packet can be filtered out quickly. The defense takes three steps. First, the source and the destination need to establish a route. Then, the source node needs to set up pairwise keys with the en route nodes. The literature provides many novel key management schemes with better performance. For example, in random key schemes [10, 9, 27, 66], any two nodes can establish a pairwise key with a sufficiently high probability, and only $O(n)$ memory is needed. Recently, Chan *et al.* [8] proposed a non-probabilistic scheme with a memory requirement of $O(\sqrt{n})$. In addition, Zhang *et al.* [63] provided a rekeying scheme based on

the collaboration among neighboring nodes to counteract compromised nodes in filtering false data. The source computes the authentication header that consists of several tokens. Each token is computed with one pairwise key so that only the node that has the pairwise key can verify the token. Due to the unreliability in ad hoc networks, a forwarding node may not be able to verify a received packet when a route is changed. Hence, we propose a scheme to improve these schemes.

2.3 DoS in Wireless Networks

There are many approaches to launching DoS attacks in an ad hoc network in the network layer. Congestion is recognized as a simple and effective DoS attack approach. In the physical layer, jamming [43] can disrupt and suppress normal transmission. In the MAC layer, the defects of MAC protocol messages and procedures of a MAC protocol can exploited by attackers. In the 802.11 MAC protocol, Bellardo *et al.* [2] discussed vulnerabilities on authentication and carrier sense, and showed that the attackers can provide bogus duration information or misuse the carrier sense mechanism to deceive normal nodes to avoid collision or keep silent. For example, the flooding nodes can forge RTS or CTS packets with false and large duration values. The flooding nodes can also disobey the contention process to leverage the channel opportunity toward attackers. They can discard the backoff procedure so that they always get the first chance to send RTS right after the end of last transmission. They can always choose the minimum contention window, so that the random defer time is likely to be shorter than others. Gu *et al.* [14] analyzed how the attackers can exhaust bandwidth by using large junk packets without breaking the MAC protocol. The attackers can use certain packet generation and transmission behavior to obtain more bandwidth than other normal nodes. Wullems *et al.* [57] identified

that the current implementation of the MAC protocol in the commodity wireless network cards enables an attacker to deceive other nodes to stop transmission. The attackers can exploit the CCA function of the 802.11 PHY protocol to suppress other nodes with the illusion of a busy channel. Borisov *et al.* [4] discovered several security flaws in WEP, which can lead to practical attacks against link security. For example, the attackers can modify the content of a message and its WEP checksum without being detected. Thus, the victim cannot get correct information from its service provider.

Previous research [1, 18, 19, 34] also found that attackers can manipulate routing procedures to break valid routes and connections. For example, Hu *et al.* [18] summarized a set of attacks which misuse routing messages. If attackers change the destination IP address in the route request message, the victim cannot establish a route to its destination and thus cannot access services. In order to prevent attackers from exploiting the security flaws in routing protocols, several secure routing protocols have been proposed to protect the routing messages, and thus prevent DoS attacks. Dahill *et al.* [44] proposed to use asymmetric cryptography for securing ad hoc routing protocols: AODV and DSR. Yi, Naldurg, and Kravets [61] presented a security-aware routing protocol which uses security (e.g., trust level in a trust hierarchy) as the metric for route discovery between pairs. Papadimitratos and Hass [36] proposed a routing discovery protocol that assumes a security association (SA) between a source and a destination, whereas the intermediate nodes are not authenticated. Hu, Perrig and Johnson designed SEAD [20] which uses one-way hash chains for securing DSDV, and Ariadne [18] which uses TESLA and HMAC for securing DSR.

Aad *et al.* identified the JellyFish attacks that drop, reorder or delay TCP packets to disrupt TCP connections [1]. They believed that the DoS resilience relies on end-to-end detection

mechanisms, because current intrusion detection approaches cannot effectively identify the attackers in ad hoc networks. For example, in [64], Zhang *el al.* proposed a general architecture to have all nodes participate in intrusion detection. Each node takes two roles. A node needs to monitor transmission in its neighborhood in order to detect misbehavior in its nearby nodes. Then, each node can cooperate with its neighboring nodes to exchange intrusion detection information in order to detect the malicious node. In [30], Marti *et al.* proposed to use watchdog to detect the attacking nodes. Basically, a good node overhears its next hop to check whether its next hop forwards the packets that are received from the good node. After detecting malicious nodes, the good node uses a pathrater to exclude the malicious node from its routes. In a clustered ad hoc network, a cluster head is elected for monitoring data traffic within the transmission range [21]. All these intrusion detection approaches needs nodes to monitor the transmission in their neighboring areas. However, a malicious node may use a directional antenna for transmission in order to avoid monitoring. Also, a malicious node may ask other malicious nodes to circumvent its transmission area. Hence, monitoring of nearby transmission may not be realistic in this kind of adversary environment. Furthermore, the detection relies on trusted neighboring nodes. A trusted node will honestly report misbehavior. However, a malicious node can ask another neighboring node to lie and deceive defenders.

Chapter 3

Key Management in Wireless Broadcast Services

3.1 Introduction

With the ever growing popularity of smart mobile devices along with the rapid advent of wireless technology, there has been an increasing interest in wireless data services among both industrial and academic communities in recent years. There are two fundamental information access approaches for wireless data services: point-to-point access and periodic broadcast [58]. Point-to-point access employs a basic client-server model, where the server is responsible for processing a query and returning the result to the user via a dedicated point-to-point channel. However, this model is inefficient given the resource limitations of current wireless technology. Periodic broadcast, on the other hand, has the server actively pushing data to the users. The server determines the data and its broadcast schedule. A user listens to the broadcast channel to retrieve data based on his queries. Compared with point-to-point access, broadcast is a very attractive alternative because it allows simultaneous access by an arbitrary number of mobile clients and thus allows very efficient usage of the scarce wireless bandwidth.

Wireless data broadcast services have been available as commercial products for many years. In particular, the recent announcement of the MSN Direct Service has further highlighted the industrial interest in and feasibility of utilizing broadcast for wireless data services. Previous studies on wireless data broadcast services have mainly focused on performance issues such as reducing data access latency and conserving battery power of mobile devices. Unfortunately,

the critical security requirements of this type of broadcast service have not yet been addressed. In the wireless broadcast environment, any user can monitor the broadcast channel and record the broadcast data. If the data is not encrypted, the content is open to the public and anyone can access the data. In addition, a user may only subscribe to a few programs. If data in other programs are not encrypted, the user can obtain data beyond his subscription privilege. Hence, data should be encrypted in a proper way to ensure that only subscribing users can access the broadcast data, and subscribing users can only access the data to which they subscribe.

Symmetric-key-based encryption is a natural choice for ensuring secure data dissemination and access. The broadcast items can be encrypted so that only those users who own valid keys can decrypt them. Thus, the decryption keys can be used as an effective means for access control in wireless data broadcast services. Nevertheless, a critical issue remains, i.e. *how can we efficiently manage keys when a user joins/leaves/changes the service without compromising security and interrupting the operations of other users?*

Regarding several unique features of broadcast services, we are interested in new key management schemes that are more convenient for users. First, a broadcast service generally provides many programs. Intuitively, we could manage a separate set of keys for each program. Thus, if a user subscribes to m programs, we ask the user to hold m sets of keys. However, this approach is inefficient for users subscribing to many programs. If users could use the same set of keys for multiple programs, there would be fewer requirements for users to handle keys. Second, we envision that a user should be able to flexibly subscribe/unsubscribe to programs of interests and make changes to his subscription at any time. Hence, one important feature of any key management in broadcast services is to support these types of dynamics in user subscriptions. The key management should ensure that a user can only access data during his

subscription period. Third, when a user changes his subscription (i.e. subscribes or unsubscribes to some programs), we argue that it is unnecessary to change keys for the programs to which the user is still subscribing, as long as security can be ensured. In this way, a lower rekey cost is needed for user activities and fewer users will be affected. Hence, the primary design goal of our proposal for key management in wireless data broadcast services is to provide security, flexibility and efficiency simultaneously. Although there are a great number of existing studies on key management in group communication literature, they do not fulfill all these requirements. Therefore, we propose a new key management scheme, namely key tree reuse (**KTR**), based on two important observations: (1) users who subscribe to multiple programs can be captured by a shared key tree, and (2) old keys can be reused to save rekey cost without compromising security.

The rest of the chapter is structured as follows. In Section 3.2, we show the service model, the most related work, and our design goals. In Section 3.3, KTR is proposed to fully exploit the service characteristics. In Section 3.4, we address the updating and distribution of shared keys. In Section 3.5, we present the results of simulations to illustrate the performance improvements in KTR. Finally, we conclude in Section 3.6.

3.2 Preliminaries

3.2.1 Service Model

A wireless data broadcast system consists of three parts: (1) the broadcast server; (2) the mobile devices; and (3) the communication mechanism as depicted in Figure 3.1. The server broadcasts encrypted data and rekey messages on air. A user's mobile device receives the broadcast information, and filters the subscribed data according to user's queries and privileges. The

specialty of this broadcast system is that (a) the server schedules to broadcast all data on air, and (b) the mobile devices, instead of the server, directly process user's queries.

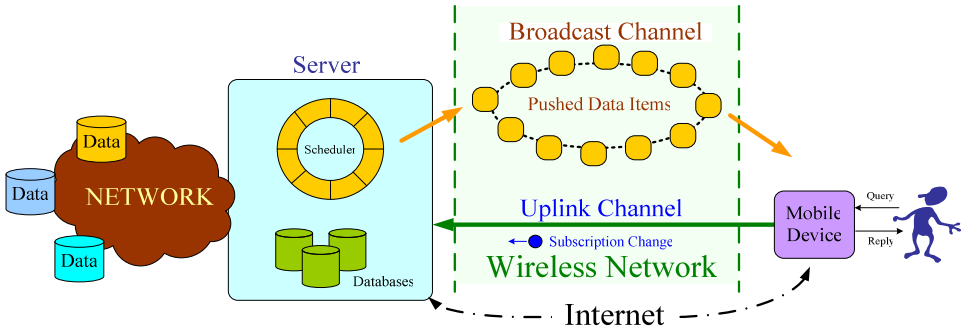


Fig. 3.1. A wireless data broadcast system

The communication mechanism includes wireless broadcast channels and (optional) uplink channels. Broadcast channel is the main mechanism for data and key dissemination in our system. Data and keys may be broadcast periodically so that users can recover lost or missed data items and keys. Since a user only has keys for his subscription, he cannot decrypt broadcast data and rekey messages designated to other users. At the same time, a data item can be decrypted by an arbitrary number of users who subscribe to it. This allows many users to receive the data at the same time and addresses the scalability problem. The uplink channels, which have limited bandwidth, are reserved for occasional uses to dynamically change subscriptions or request lost or missed keys.

In this study, data items are grouped into **programs** and a user specifies which programs he would like to access. For simplicity, we assume that each program covers a set of data items,

and that programs are exclusively complete. A user may subscribe to one or more programs. The set of subscribed programs is called the user's **subscription**. Users can subscribe via Internet or uplink channels to specify the programs that they are interested in receiving (monitoring). Each program has one unique key to encrypt the data items. The key is issued to the user who is authorized to decrypt and receive the data items. If a user subscribes to multiple programs, it needs an encryption key for each program. When a user joins or leaves a program, the encryption key needs to be changed to ensure that the user can only access the data in his subscription period.

3.2.2 Logical Key Hierarchy

LKH is an efficient and secure key management for broadcast services with one program, since each group member only needs to hold $O(log(n))$ keys, and the size of a broadcast rekey message for each new DEK is also $O(log(n))$. An example of LKH is depicted in Figure 3.2. A group of users use such a key tree for key management. The root (top node) of the tree is the *data encryption key* (**DEK**) of the program, i.e. k_1, which is used to encrypt group data. At the bottom are the leaf nodes, each of which is an *individual key* (**IDK**) of a particular user. The individual key is set up when the user registers in the group, and only shared between the server and the user. In the middle of the key tree are keys, namely *key distribution keys* (**KDK**s), which are used to encrypt new DEKs and KDKs. Each user has a unique path from his leaf node to the root node, and the user only knows the keys along his own path.

When a user joins or leaves the group, the server needs to change and broadcast the corresponding new keys, and this operation is called *rekey*, and the broadcast message of new keys is called *rekey message*. In our system, data and rekey messages are broadcasted in the same broadcast channel to the users. Assume user u_1 leaves the group. The server needs to change

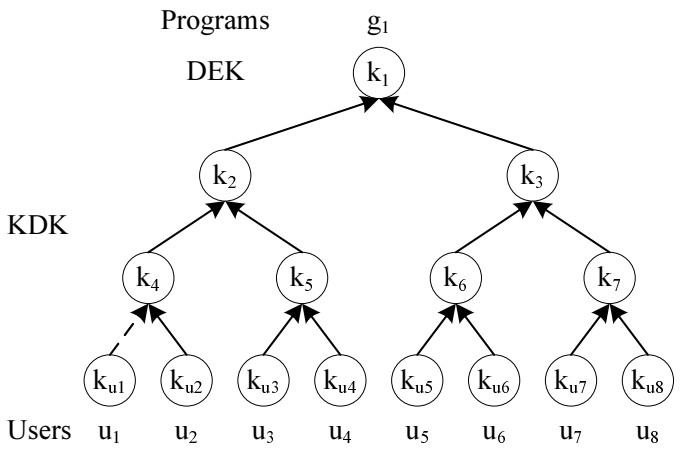


Fig. 3.2. Logical key hierarchy

k_4, k_2 and k_1 so that u_1 will no longer receive any data for this group, which is encrypted by $k1$. The rekey message is

$$\{k_4'\}_{k_{u_2}}, \{k_2'\}_{k_4'}, \{k_2'\}_{k_5}, \{k_1'\}_{k_2'}, \{k_1'\}_{k_3}$$

where k_i' is the new key of k_j and $\{k_i'\}_{k_j}$ means k_i' is encrypted by k_j. When u_2 receives this message, u_2 first decrypts $\{k_4'\}_{k_{u_2}}$ based on his individual key k_{u_2} to obtain k_4', then uses k_4' to decrypt $\{k_2'\}_{k_4'}$ and so on to obtain k_2' and k_1'. Similarly, other users can obtain the new keys in their own paths. It is obvious u_1 cannot obtain any new keys from this message, and thus the broadcast data in the future will not be decrypted by u_1.

Now assume u_1 joins the group, and the server needs to change k_4, k_2 and k_1 so that u_1 cannot use the old keys to decrypt old broadcast data. Note that u_1 may have already eavesdropped on some broadcast data before he joined the group. If the server gives u_1 the old keys,

u_1 can decrypt the eavesdropped broadcast data. The rekey message is

$$\{k_4'\}_{k_{u_1}}, \{k_2'\}_{k_{u_1}}, \{k_1'\}_{k_{u_1}}, \{k_4'\}_{k_4}, \{k_2'\}_{k_2}, \{k_1'\}_{k_1}$$

The first three components are for u_1 to use his individual key to decrypt the new keys, and the last three are for all existing users to use their old keys to decrypt the new keys. In this way, u_1 will not obtain any old key.

Compared with LKH-based approaches, key management schemes in broadcast encryption are less flexible regarding possible subscriptions. Hence, we select LKH as the basis to present our scheme. In the future, we would investigate whether the integration of broadcast encryption techniques into our scheme can be further optimized by exploiting overlaps between groups and programs.

Nevertheless, directly applying LKH to broadcast services is not the best option. A broadcast service provides many programs, and users subscribing to one program form an equivalent multicast group. Hence, if a user subscribes to multiple programs, he is in multiple multicast groups, and thus need to hold multiple sets of keys. Letting multiple programs share the same key tree is more efficient [49]. However, we find that only reducing the redundancy of key trees is still not enough to address issues of security and efficiency in broadcast services. In fact, new problems are raised when a user changes his subscription. We need to determine whether keys for the programs to which the user is still subscribing should be changed to ensure security, instead of simply renewing keys as in traditional LKH-based approaches. Consequently, KTR differs from the key sharing approach [49] in its handling of the shared keys and provides

a minimum rekey cost without compromising security, which can be seen from our extensive simulations .

3.2.3 Design Goals

Given the broadcast service, we want to design a secure, flexible and efficient key distribution scheme.

- Security.

  Each user subscribes to a few programs for certain periods in broadcast services. The scheme should restrict users from accessing data out of their subscription periods and data belonging to other programs. In another words, a user owns valid keys only for his subscription and only during his subscription period. Therefore, a user cannot access any data not in his subscription, and also cannot access any data before he joins the service or after he quits the service. Furthermore, in our applications, the scheme should deal with new security considerations, such as the handling of shared keys when a user keeps some programs while changing other programs in his subscription.

- Efficiency.

  A mobile device is resource-limited in terms of storage, energy and computation. Hence, it would be efficient if the scheme allows a user to hold as few as possible keys for his subscription to save storage, and change as few as possible keys in a rekey event to save energy and computation. Therefore, instead of holding separate sets of keys for all programs, the scheme should only give one set of keys to a user. Instead of changing all keys

along “paths” as in conventional LKH approaches, the scheme should inspect every key until the minimum number of must-be-changed keys are found.

- Flexibility.

  A user may join, quit or change subscriptions at any time. Instead of asking users to pre-plan their subscriptions or only permitting subscriptions according to fixed broadcast sessions, the scheme needs to support user demands flexibly. We expect a good key management scheme will distribute new keys immediately at the time when users change subscriptions. In this way, a user only pays for what was subscribed.

3.3 Shared Key Tree

Because directly applying LKH is not efficient, we use a shared key structure to address the key management in broadcast services. In the following, we first describe how a shared key structure is applied and then raise the security and efficiency problems of this scheme. We then present an approach for shared key management that ensures security and minimizes rekey cost, and also address major issues when applying KTR in a broadcast server.

3.3.1 Key Forest

To address scalability and flexibility in key management, LKH is used as the basis to present our scheme. An intuitive solution is to use a key tree for each program as shown in Figure 3.3(a), where the broadcast server manages two separate key trees for two programs g_1 and g_2. If a user subscribes to a program, he needs to handle keys along the path from the user's IDK via KDKs to the DEK in the program's key tree. As depicted, when user u_1 subscribes to

two programs simultaneously, he needs to manage two sets of keys in both trees which is not very efficient (see Figure 3.3(a)). Hence, shared key tree (**SKT**) is proposed to reduce this cost in key management. As shown in Figure 3.3(b), we let the two programs share the same sub key tree as represented by the gray triangle. We regroup users so that users subscribing to both programs only need to manage keys in the gray triangle.

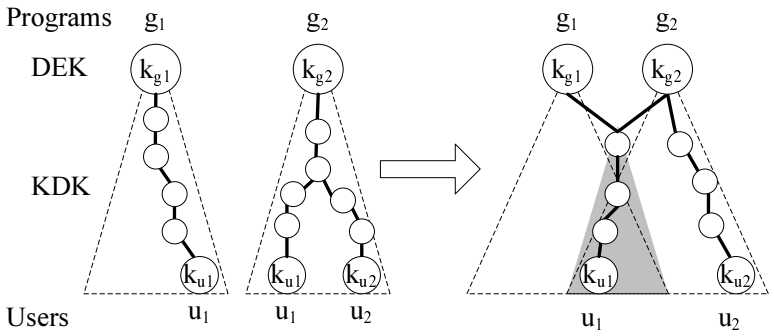


Fig. 3.3. Shared key tree

The advantage of shared key tree is clear: any user subscribing to both g_1 and g_2 only needs to manage one set of keys for both programs. Moreover, when a user joins or leaves a tree shared by multiple programs, the encryption and communication cost for rekey operations can be significantly less than conventional LKH approaches. Our approach is also very general in that it does not require two programs that share a sub-key-tree to have any semantic relationship.

In this study, shared key tree is modeled as a key forest (see Figure 3.4), in which all keys form a directed and acyclic graph. The top keys in the forest are the DEKs of the programs

provided in the broadcast service. All other keys (KDKs) form trees. In the forest, each unique subscription is represented by a tree structured as LKH. As discussed in Section 3.2.1, a subscription is a set of programs. Hence, the root of a tree is connected to the DEKs of the programs belonging to a subscription. As in Figure 3.4, the broadcast service provides three programs, g_1, g_2 and g_3. There could be 7 trees for 7 unique subscriptions. However, in this example, only 6 subscriptions have users, and thus 6 trees exist: tree tr_1 stands for the subscription with only one program g_1; tr_2 for g_2; tr_3 for g_3; tr_4 for g_1 and g_2; tr_5 for g_2 and g_1; tr_6 for g_1, g_2 and g_3. No user in this example subscribes only g_1 and g_3.

Users are placed in trees according to their subscriptions. In another words, a tree represents not only a unique subscription, but also a group of users subscribing to the same set of programs. If a tree is connected to multiple programs, the keys in the tree are in fact shared by the connected programs. A user in a tree only needs to handle the keys in the tree and the DEKs of the connected programs. For example, in Figure 3.4, since tr_4 represents a subscription of g_1 and g_2, its root k_{r_4} is connected to both k_{g_1} and k_{g_2}. The keys in tr_4 is shared by both g_1 and g_2. A user u_s in tr_4 subscribes g_1 and g_2 and needs to handle keys in the path from his leaf node to the DEKs of the subscribed programs: k_{u_s}, k_{n_L}, k_{r_4}, k_{g_1} and k_{g_2}. Finally, k_{g_1}, k_{g_2} and k_{g_3} are DEKs to encrypt broadcast data, and all other keys are KDKs.

In order to ensure that a user will not pay for subscribed programs multiple times, the key forest obviously should have the following properties, which are guaranteed in any directed and acyclic graph.

- PROPERTY 1. *Only one path exists by following the upward links from the root of a tree tr_s to the DEKs of the programs that share tr_s;*

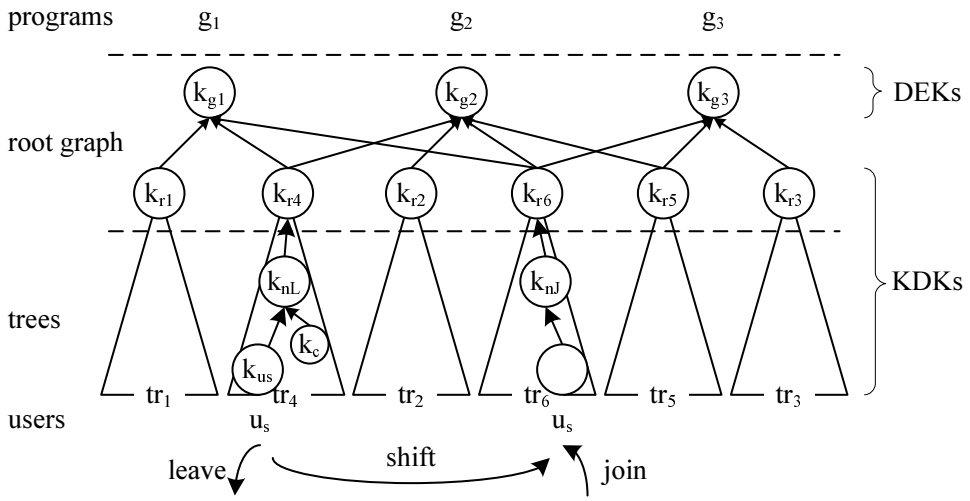


Fig. 3.4. Key forest

- PROPERTY 2. *Only one path exists by following the upward links from any leaf node in a tree to the root;*

- PROPERTY 3. *Each user belongs only to one tree in the key forest, and his individual key is the leaf node of the tree.*

3.3.2 Root Graph

The root graph depicts how programs share keys. In Figure 3.4, the root graph has two layers. The top layer consists of the DEKs of the programs, and the bottom layer consists of the roots of the trees. Since the number of possible shared key trees grows exponentially with the number of programs, m programs could generate $2^m - 1$ unique subscriptions. Hence, when the number of programs is large, such a two-layer structure in fact brings two major problems in terms of rekey overheads.

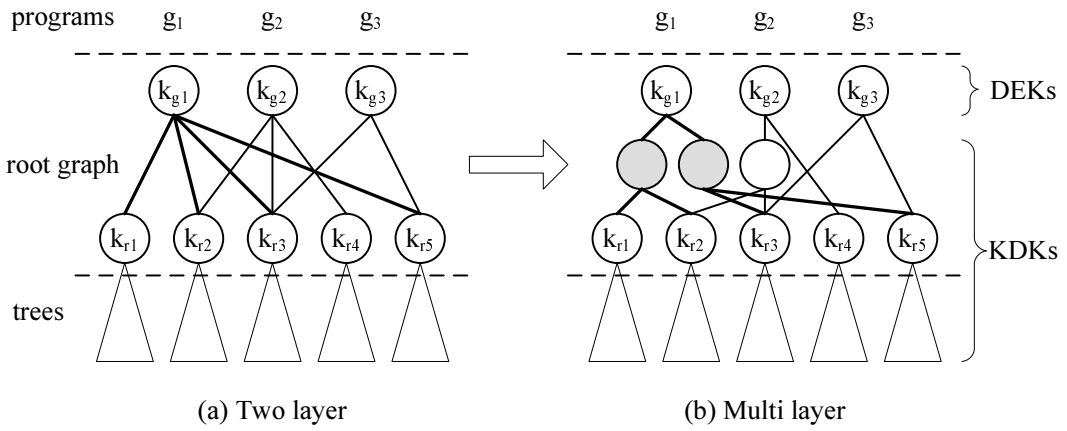


Fig. 3.5. Multi-layer root graph

First, a program may be included in many subscriptions, which means the DEK of the program is connected with many roots. Assume the DEK is connected with n trees. When a user stops subscribing the program, the DEK needs to be updated and distributed to users in n trees. Because the new DEK is encrypted with the roots of the n trees in rekey, $O(n)$ rekey items are generated. Obviously, if n is large, a leave event results in a huge rekey message. For example, in Figure 3.5(a), 3 programs are included in 5 unique subscriptions. Program g_1's DEK k_{g_1} is connected with 4 roots k_{r_1}, k_{r_2} k_{r_3} and k_{r_5}. Hence, when k_{g_1} is updated due to a leave event, 4 rekey items are needed.

To solve this problem, we use a multi-layer structure to connect the DEK with the roots of the shared trees. As in Figure 3.5(b), k_{g_1} is connected (bold lines) with k_{r_1}, k_{r_2} k_{r_3} and k_{r_5} via two intermediate key nodes (gray circles). Such a multi-layer structure inherently exploits the advantages of LKH. For a leave event, the number of rekey items in the root graph is reduced to $O(log_2(n))$. Note that, for different programs, the number of intermediate nodes and the number of layers may be different, which is obviously determined by the number of trees to

which the program is connected. In Figure 3.5, program 2 is connected with 3 trees, and thus 1 node is needed. No node is needed for program 3, because it is only connected with 2 trees.

Second, m programs could generate $2^m - 1$ unique subscriptions. Managing keys for all possible subscriptions would overload the server. Now, assume the service has n users and $2^m \gg n$. Although $2^m - 1$ unique subscriptions exist, at most n subscriptions are valid, since the number of valid subscriptions cannot be more than the number of users. Hence, this problem can be easily solved by letting the server only manage the valid subscriptions that have at least one user.

Assuming n users are distributed to e trees ($e \leq n$), and each tree is shared by d programs on average ($d \geq 1$), then each tree would have $\frac{n}{e}$ users, and each program would have $\frac{ed}{m}$ trees. As discussed in Section 3.5.4, KTR requires that the broadcast server manage $O((2\frac{n}{e} - 1)e + (\frac{ed}{m} - 1)m) = O(2n - e + ed - m)$ keys. In the worst case where $e = n$, the server needs to manage only $O(n(1 + d) - m)$ keys. In comparison, LKH requires that the server manage $O((2\frac{ed}{m}\frac{n}{e} - 1)m) = O(2nd - m)$ keys. Obviously, KTR allows the server to manage much fewer keys than conventional LKH-based approaches.

3.3.3 Rekey Operations

In a key forest, rekey happens when users join, quit or change subscriptions, and new keys are distributed in the broadcast channel as the data items. In this study, we consider user activities of **joining/leaving/shifting** among trees, instead of joining/quiting/changing among programs. Table 3.1 lists the mapping between tree-oriented operations and the corresponding user events. Consider the example in Figure 3.4, where a user u_s shifts from tr_4 to tr_6. When u_s was in tr_4, u_s subscribed g_1 and g_2. After he shifts to tr_6, he subscribes g_1, g_2 and g_3. Hence,

the shift in fact means the user adds g_3 into his current subscription. Note that the discussion of rekey operations in this study only considers individual user events. The batch rekey operations for simultaneous user events can be found in [45, 59], and can be incorporated in our scheme in the future.

Table 3.1. Rekey operations

Tree-oriented rekey events	Program-oriented user events
Join a tree	A user has not subscribed to any program. Then, • He subscribes to only one program. • He subscribes to multiple programs simultaneously.
Leave a tree	A user has subscribed to several programs. Then, • He unsubscribes to all current programs at the same time.
Shift among trees	A user has subscribed to several programs. Then, • He subscribes to one more programs. • He subscribes to a few more programs. • He unsubscribes to one of the current programs. • He unsubscribes to a part of the current programs. • He changes a part of the current programs to another set of programs.

To issue new keys upon a user event, the main task is to identify the keys that need to be changed. We use two types of paths in the key forest to represent the to-be-changed keys. When a user leaves a tree, we say, a **leave path** is formed, which consists of keys that the user will no longer use. When a user joins a tree, we say, an **enroll path** is formed, which consists of keys that the user will use in the future. Similarly, when a user shifts from one tree to another, a leave path and an enroll path are formed. Similar to LKH, a path in KTR includes all keys inside a tree from the leaf node of the user to the root of the tree. But unlike LKH, a path in KTR also includes the keys in the root graph. Hence, a complete path starts from the leaf node

and ends at the multiple DEKs of the subscribed programs that share the tree. For example, in Figure 3.4, when u_s shifts from tr_4 to tr_6, the leave path consists of k_{n_L} and k_{r_4}, and the enroll path consists of k_{n_J}, k_{r_6}, k_{g_1}, k_{g_2} and k_{g_3}. Note that in this example, k_{g_1} and k_{g_2} are the keys that u_s already has and still needs in the future. Hence, k_{g_1} and k_{g_2} are not in the leave path, although u_s leaves tr_4.

Knowing these paths, we need to change the corresponding keys. However, the rekey operations and messages for leave path and enroll path are different. In LKH, keys in both paths must be changed regardless of circumstance. However, in KTR, this is not necessarily the case. As described by the two rules below, only the keys in a leave path are changed as in LKH, while only a few keys in an enroll path are changed as long as security is ensured. In addition, a new approach is proposed to identify an enroll path with the minimum number of to-be-changed keys to reduce rekey cost without compromising security. The detailed approach is presented in Section 3.4.

(1) If a key is in a leave path, change it.

(2) If a key is in an enroll path, change it only if it is critical according to Theorem 3.1.

To broadcast the new keys, the server should first compose rekey packets. The rekey packets are broadcast before encrypting the data with the new keys. The server also needs to provide key indices so that the users can know which key needs to be received. With a simple index approach, a rekey packet is composed as a sequence of **rekey items**

$$\underline{(i_1, j_1)\{k'_{i_1}\}_{k_{j_1}}}, \underline{(i_2, j_2)\{k'_{i_2}\}_{k_{j_2}}}, \cdots$$

Each underlined unit is a rekey item, which consists of a pair of key indices and an encrypted key, where k_i' is the new value of k_i, and k_j is the KDK to encrypt k_i'. They are indexed as (i, j). When a user receives a rekey item $(i, j)\{k_i'\}_{k_j}$, he first check whether he holds k_i and k_j according to the indices. If no, he discards the rekey item. Otherwise, he uses k_j to decrypt $\{k_i'\}_{k_j}$ and thus obtains the new k_i'.

In this study, we take the standard LKH approach to encrypt the new key k_i' with k_j. If k_i' is in an enroll path, k_j is k_i, i.e. $\{k_i'\}_{k_j} \equiv \{k_i'\}_{k_i}$. If k_i' is in a leave path, k_j is a child key of k_i'. The order of the key items is bottom up in the key forest so that a receiver changes the keys from leaves to roots following the paths. Readers can refer to [53, 50] for examples of rekey packets.

3.3.4 Challenges in Shared Key Management

If keys are shared by multiple programs, it is not always cost saving, especially when a user shifts to a tree in which some programs have already been subscribed to by the user. Still considering the above example in Figure 3.4 where u_s shifts from tr_4 to tr_6. In general, two sets of keys need to be changed to ensure security: k_{n_L} and k_{r_4} in the leave path, and k_{n_J}, k_{r_6}, k_{g_1}, k_{g_2} and k_{g_3} in the enroll path.

When keys are not shared (i.e. each program has one individual key tree), less keys are involved in the example shift event. As in Figure 3.4, tr_4 is shared by g_1 and g_2, while tr_6 is shared by g_1, g_2 and g_3. The shift from tr_4 to tr_6 in fact indicates that u_s adds the program g_3 to his current subscription of g_1 and g_2. Hence, if keys are not shared and each program has one individual key tree, only keys in g_3's tree need to be changed. Thus, the shared key scheme has more rekey cost than conventional LKH in this example.

Nevertheless, it is noticed that after shifting to tr_6, u_s is still subscribing programs g_1 and g_2 as he was in tr_4. Obviously, some keys in the enroll path (e.g. k_{n_J}, k_{r_6}, k_{g_1} and k_{g_2}) were used for users in tr_4 and tr_6, regardless of whether u_s is in tr_4 or tr_6. We find that it might be secure to reuse some keys in tr_6 after u_s shifts to tr_6, because these keys may only reveal to u_s what he has already known. For example, no matter whether u_s was in tr_4 or shifts to tr_6, u_s always knows k_{g_1} or k_{g_2}. Hence, these two keys can be **reused** in this shift event without compromising security. With an in-depth analysis, we find that under certain conditions k_{r_6} and k_{n_J} can be reused as well.

Deciding whether or not a key in an enroll path can be reused without any change depends on whether or not the key can reveal the programs' DEKs that u_s is not supposed to know. In LKH, a key k_i is used to encrypt its parent key k_j, when k_j is in a leave path and needs to be changed. The encryption takes the form $\{k_j\}_{k_i}$ [53]. Accordingly, k_i reveals k_j when a user records $\{k_j\}_{k_i}$ in broadcast and obtains the reused k_i, because the user can decrypt k_j from $\{k_j\}_{k_i}$.

Now, let's assume, in the example shift event, k_{r_6} has never been changed since t_0, u_s joins tr_4 at t_1 and shifts to tr_6 at t_2, and $t_0 < t_1 < t_2$. There are at least two situations where k_{r_6} must be changed at t_2.

- Either k_{r_6} was used to encrypt k_{g_3} as $\{k_{g_3}\}_{k_{r_6}}$ during t_0 and t_2,
- Or k_{r_6} was used to encrypt k_{g_1} or k_{g_2} as $\{k_{g_1}\}_{k_{r_6}}$ or $\{k_{g_2}\}_{k_{r_6}}$ during t_0 and t_1.

In the first situation, if k_{r_6} is reused, u_s can decrypt k_{g_3} from $\{k_{g_3}\}_{k_{r_6}}$, and then decrypt g_3's data that was broadcast before u_s shifts to tr_6 at t_2. Hence, the reused k_{r_6} reveals k_{g_3} before u_s adds g_3 into his subscription. In the second situation, if k_{r_6} is reused, u_s can decrypt

k_{g_1} or k_{g_2} from $\{k_{g_1}\}_{k_{r_6}}$ or $\{k_{g_2}\}_{k_{r_6}}$, and then decrypt g_1's or g_2's data that was broadcast before u_s joins tr_4 at t_1. Hence, the reused k_{r_6} reveals k_{g_1} or k_{g_2} before u_s joins the service. In summary, k_{r_6} must be changed in both situations. If k_{r_6} is reused without any change, it can reveal DEKs that u_s should not know and thus enable u_s to obtain data from programs before his subscription period started. Except these two situations, if k_{r_6} has never been used in the encryptions as discussed above, k_{r_6} can be reused without any change.

A similar but more complicated inspection is required on other keys in tr_6. The principle is to check whether a key in an enroll path can reveal the previous DEK or another program's DEK. The difficulty is that a key may indirectly reveal DEKs, because its parent may not be a DEK. For example, although k_{n_J}'s parent key is k_{r_6}, k_{n_J} can still indirectly reveal the DEK k_{g_3}. Assume that k_{n_J} was used to encrypt k_{r_6} and then k_{r_6} was used to encrypt k_{g_3}. In this situation, a sequence of rekey items $\{k_{r_6}\}_{k_{n_J}}$,$\{k_{g_3}\}_{k_{r_6}}$ exist in all broadcast messages. If k_{n_J} is reused without any change, u_s can first decrypt k_{r_6} and then k_{g_3}. Thus, k_{n_J} must be changed. If no such sequence of rekey items exist, k_{n_J} can be reused. Hence, the challenge in reusing keys lies in how to find out which sequences of rekey items may compromise security and which keys in such sequences may reveal DEKs.

In the following section, we propose a special approach in KTR to efficiently address the security issue in reusing keys. Since rekey cost is determined by the number of must-be-changed keys, the cost can be minimized if we can find the minimum number of must-be-changed keys when the user joins or shifts to the tree. We name the must-be-changed keys in an enroll path as **critical keys**. In other words, KTR only changes all keys in a leave path and the critical keys in an enroll path, while leaving all the other keys unchanged. In this way, the rekey cost can be

minimized. Note that keys in a leave path always need to be changed to ensure forward secrecy [50].

3.4 Shared Key Management

In the section, we first present some important concepts in Section 3.4.1 and 3.4.2, which are used for identifying critical keys . Then, we present the condition under which a key is critical in Section 3.4.3 and 3.4.4.

3.4.1 Rekey Spots

KTR basically logs how a key was used in rekey messages. We can always find two operations in any rekey message: 1) a key's value is changed or 2) a key is used to encrypt its parent key when the parent key's value is changed. Accordingly, we define two types of spots to log the time points when either operation is committed.

DEFINITION 1. *Renew spot of a key k_i: the time point t when k_i's value is changed. k_i's new value starting from t is denoted as $k_i(t)$.*

DEFINITION 2. *Refresh spot of a key k_i: the time point t when k_i is used to encrypt its parent key k_j's new value in a refreshment $\delta(k_j, t; k_i, t')$.*

DEFINITION 3. *Refreshment, $\delta(k_j, t; k_i, t')$: a rekey message broadcast at t in the form of $\{k_j(t)\}_{k_i(t')}$, and $t' \leq t$.*

At a refresh spot t, we say k_i is refreshed when it is used to encrypt its parent key k_i's new value as in the $\delta(k_j, t; k_i, t')$. At a renew spot t, we say k_i is renewed when its valued is

changed. k_i is renewed only when it is in a leave path or critical in an enroll path. Accordingly, the rekey message to renew k_i has two possible forms. If k_i is critical in an enroll path, k_i's new value is encrypted with its old value. Hence, the renew message is $\{k_i(t)\}_{k_i(t')}$. If k_i is in a leave path, k_i's new value is encrypted with its child key k_c. Hence, the renew message of k_i is also the refresh message of k_c, i.e. $\{k_i(t)\}_{k_c(t')}$.

When a user changes his subscription, the server needs to change certain keys according to the algorithm presented in Section 3.4.3 and broadcast corresponding rekey messages. Then, refresh and renew spots are logged to the keys that are used in the rekey messages. The sequence of refresh and renew spots thus forms spot series in the time order. If a key is shared by multiple programs, we let the key have multiple spot series, each of which is associated with one program (see examples in Section 3.4.2).

Algorithm 1 describes the procedure to log refresh and renew spots upon a user event. According to Definition 1, renew spots are logged to a key when the key is changed to a new value. Hence, when a key is in a leave path or critical in an enroll path, the key must be changed and renew spots must be logged to that key. Furthermore, according to Definition 2, refresh spots must be logged to a key when the key's parent is in a leave path, because the key is used to encrypt its parent in order to renew its parent.

As previously discussed, refreshments may contain information that threaten past confidentiality. For example, assume a user joins a tree shared by program g_m at t_c, and k_1 is in the enroll path. From all previous broadcast rekey messages, a dangerous rekey sequence is defined as,

DEFINITION 4. *A dangerous rekey sequence for k_i is a sequence of refreshments*

$$\delta(k_2, t_2; k_1, t_1), \delta(k_3, t_3; k_2, t_2), \cdots, \delta(k_m, t_m; k_{m-1}, t_{m-1})$$

Algorithm 1 Update of refresh and renew spots

Assume k_i is used in the rekey messages upon a user event.

1: **if** k_i is in a leave path **then**
2: renew spots must be added to all k_i's spot series;
3: **end if**
4: **if** k_i is critical in an enroll path **then**
5: renew spots must be added to all k_i's spot series;
6: **end if**
7: **if** k_i's parent key k_j is in a leave path **then**
8: refresh spots must be added to k_i's spot series that are associated with the programs sharing k_j;
9: **end if**

satisfying

- k_i *is* k_{i-1}*'s parent key,*
- $t_1 \leq t_2 \leq \cdots \leq t_m < t_c$*,*
- $k_1(t_1)$ *has never been changed since* t_1*,*
- $k_m(t_m)$ *is the value of* g_m*'s DEK.*

If the server sends k_1 to the user without any change, the user can decrypt $k_2(t_2)$ and then iteratively decrypt $k_3(t_3)$ to $k_m(t_m)$. Past confidentiality at t_m is thus compromised, if the user uses $k_m(t_m)$ to decrypt the data items that are broadcast during t_m and t_c, which should otherwise be inaccessible to that user. Past confidentiality at t_m is preserved if the user either cannot find such a sequence of refreshments to obtain $k_m(t_m)$ or has already legitimately obtained $k_m(t_m)$ and the data items during t_m and t_c.

Therefore, to identify whether a key is critical in a program, we use a third type of spot to mark the key based on its renew and refresh spots. Similarly, the sequence of revive spots forms

Algorithm 2 Update of revive spots

```
1: let k be k_v;
2: let t be t_v;
3: UPDATEREVIVE(k, t, t_v, g_v);
4: function UPDATEREVIVE(k, t, t_v, g_v)
5:     let V be the set of all child keys of k;
6:     for k_i ∈ V do
7:         find t_i the renew spot of k_i that satisfies t_i ≤ t;
8:         if δ(k, t; k_i, t_i) exists then
9:             if t_i is the latest renew spot of k_i then
10:                add t_v to k_i's revive spot series associated with g_v
11:            end if
12:            UPDATEREVIVE(k_i, t_i, t_v, g_v);
13:        end if
14:    end for
15: end function
```

revive spot series in the time order. If a key is shared by multiple programs, the key has multiple revive spot series, each of which is associated with one program (see examples in Section 3.4.2).

DEFINITION 5. *Revive spot of a key: the time point t when (1) the DEK of this key's associated program has changed or (2) a dangerous rekey sequence exists as defined in Definition 4.*

After the update of refresh and renew spots, the server updates revive spots according to Algorithm 2. Assume k_v (the DEK of program g_v) is renewed at t_v. Algorithm 2 updates the revive spots of corresponding keys iteratively, starting from k_v. At the beginning, let $k = k_v$ and $t = t_v$. The algorithm iteratively uses the function UPDATEREVIVE(k, t, t_v, g_v) to update revive spots of related keys. Assuming that the algorithm checks a key k which was renewed at t and was refreshed at sometime later than t, it first selects a k_i from all k's child keys. According to Definition 5, if $\delta(k, t; k_i, t_i)$ does not exist, there is no need to log a revive spot to k_i and no need to further check k_i's child keys. Also, if $\delta(k, t; k_i, t_i)$ does exist but t_i is not k_i's latest

renew spot, there is still no need to log a revive spot to k_i but the algorithm continues to check k_i's child keys as they may have revive spots. If $\delta(k, t; k_i, t_i)$ does exist and t_i) is k_i's latest renew spot, a revive spot is logged to k_i and the algorithm continues to check k_i's child keys.

3.4.2 Examples of Spots

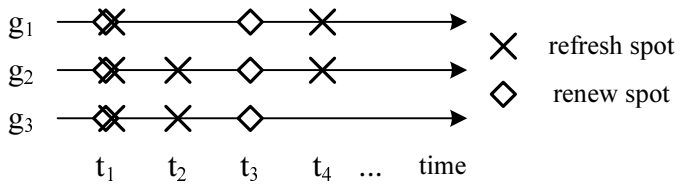


Fig. 3.6. Spot series of key k_{r_6}

In this part, we demonstrate the spot series from two different dimensions. First, a key has multiple spot series associated with its programs. Figure 3.6 depicts the spot series of k_{r_6} in the key forest of Figure 3.4. Because k_{r_6} is shared by three programs (i.e. g_1, g_2 and g_3), it has three spot series, and each series is represented by a line in Figure 3.6. In this example, at t_1, a user leaves tr_6. k_{r_6} is first renewed, and all its spot series get a renew spot. Right after it is renewed, k_{r_6} is used in the refreshments $\delta(k_{r_1}, t_1; k_{r_6}, t_1), \delta(k_{r_2}, t_1; k_{r_6}, t_1), \delta(k_{r_3}, t_1; k_{r_6}, t_1)$. Because these refreshments are related to all the programs, refresh spots are added to all k_{r_6}'s spot series. At t_2, a user leaves tr_5. Because only k_{r_2} and k_{r_3} need to be changed, k_{r_6} is used in the refreshments $\delta(k_{r_2}, t_2; k_{r_6}, t_1), \delta(k_{r_3}, t_2; k_{r_6}, t_1)$. Hence, only two refresh spots are added to the series associated with g_2 and g_3. Readers can find that the other spots are for the events

where a user joins tr_6 at t_3, and another user shifts from tr_4 to tr_3 at t_4. In brief, renew spots of a key are the same in all of its series, while refresh and revive spots are different regarding their corresponding programs.

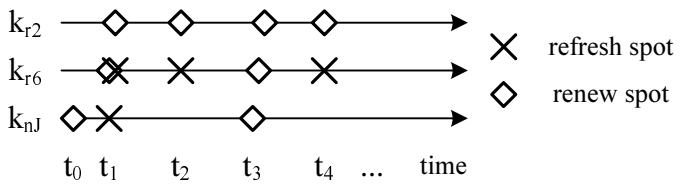


Fig. 3.7. Spot series regarding program g_2

The second dimension of spot series is illustrated in Figure 3.7, where we draw spot series of k_{n_J}, k_{r_6} and k_{r_2} associated with only one program g_2. At t_1, a user leaves tr_6. Assume k_{r_6} and k_{r_2} are in the leave path, but k_{n_J} is not. Hence, the broadcast server composes the refreshments $\delta(k_{r_6}, t_1; k_{n_J}, t_0), \delta(k_{r_2}, t_1; k_{r_6}, t_1)$ to change k_{r_6} and k_{r_2}. At t_2, a user leaves tr_5. The refreshment $\delta(k_{r_2}, t_2; k_{r_6}, t_2)$ is broadcast to change k_{r_2}. At t_3, a user joins tr_6. Assume k_{n_J}, k_{r_6} and k_{r_2} are in the enroll path, these keys are changed. At t_4, a user shifts from tr_4 to tr_3, and $\delta(k_{r_2}, t_4; k_{r_6}, t_4)$ is broadcast to change k_{r_2}. Note that at a refresh spot (for instance, t_1 of $\delta(k_{r_6}, t_1; k_{n_J}, t_0)$, the symmetric key k_{n_J} is refreshed simultaneously as its parent key k_{r_6} is renewed.

In Figure 3.8, we draw the revive spot series of the three keys associated with g_2 based on Figure 3.7. At t_2, k_{r_2} is changed and $\delta(k_{r_6}, t_1; k_{n_J}, t_0), \delta(k_{r_2}, t_2; k_{r_6}, t_1)$ can be recorded by any user. Revive spots are added to both k_{r_6} and k_{n_J}, since the refreshments can expose k_{r_2}

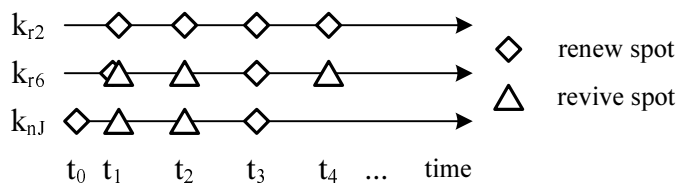


Fig. 3.8. Revive spots regarding program g_2

at t_2 to a new user if either k_{r_6} or k_{n_J} is given to the user without any change. However, at t_4, a revive spot is only added to k_{r_6}, because only $\delta(k_{r_2}, t_4; k_{r_6}, t_4)$ is potentially insecure. No revive spot is added to k_{n_J} at t_4, since it is not used in the rekey operation.

3.4.3 Critical Keys

By logging spots to keys, the server can inspect a key's past usage. We introduce the concepts of key age and subscription age to decide whether a key is a critical key.

DEFINITION 6. *Age of a key: (1) if the key is a DEK, its age is the time interval between the current time to its latest renew spot; (2) if the key is a KDK, its age is the time interval between the current time to the revive spot that is located between the current time and the latest renew spot and is closest to the latest renew spot. Note that a key may have multiple ages if it is shared by multiple programs, and each age is associated with one program.*

According to the above definition, the age of a KDK is 0 if and only if there is no revive spot between the current time and the latest renew spot. Otherwise, the age of the key is greater than 0.

DEFINITION 7. *Age of a subscription: the time interval between the current time to the latest beginning time the user is in a program. Note thatif a user subscribes to multiple programs, he has one subscription age for each program.*

According to the above definition, the subscription's age is 0 if and only if the user is not in the program. Otherwise, the user is in the program, and his subscription age is greater than 0. If a user stops subscribing a program, the subscription age associated with this program turns to 0. If a user shifts from a tree to another tree while staying in a program, his subscription age with this program continues. Finally, a user can have different subscription ages for different programs.

The traditional LKH approach to protecting past confidentiality is to always have the server change keys in the enroll path so that no previous refreshment can be exploited. Although this ensures past confidentiality, this approach is costly. There are some situations where the old refreshments only contain secrets that the user already knows. In these situations, the user can use the old symmetric key to decrypt the old refreshments, thus keeping past confidentiality intact. That being the case, the server can directly distribute the old symmetric key without compromising the past confidentiality.

In the following, we give a generic method to identify critical keys in the enroll path and reduce the rekey cost. Assuming key k is shared by m programs and will be distributed to user u, we can get all k's ages and all u's subscription ages associated with these programs, denoted as $[ka_1, ..., ka_m]_k$ and $[ua_1, ..., ua_m]_u$. Program g_i is thus associated with a pair of ages, denoted as $(ka_i, ua_i)_{k,u}$.

THEOREM 3.1. *Theorem of Critical Key (**TCK**): k in the enroll path is a **critical key**, i.e. k must be changed before being distributed to u to ensure past confidentiality, if and only if at least one pair of $(ka_i, ua_i)_{k,u}$ satisfies $ka_i > ua_i$ at current time t, i.e. the key is older than the user's subscription regarding program g_i.*

Proof of the sufficient condition: If k is the DEK of g_i, the proof is obvious. If k's age is older than u's subscription age, there are some data items encrypted by k before the user joins the program. Hence, k needs to be changed so that the user cannot decrypt those data items.

If k is not a DEK, let k_i be the DEK of program g_i, and the latest renew spot of k is t_k. Assume a pair of $(ka_i, ua_i)_{k,u}$ that satisfies $ka_i > ua_i$ at current time t. According to Definition 6, k's value has never been changed since $t-ka_i$ and was revived at $t-ka_i$. According to Definition 5, u can find such a sequence of refreshments from all previous broadcast rekey messages at $t - ka_i$: $\delta(k_\alpha, t_\alpha; k, t_k), ..., \delta(k_i, t - ka_i; k_\beta, t_\beta)$, where $t_k \leq t_\alpha \leq ... \leq t_\beta \leq t - ka_i$. Hence, if k is sent to u without any change, u can derive k_i at $t - ka_i$ from these refreshments.

According to Definition 7, u joined g_i at $t - ua_i$, which means u is only allowed to decrypt data items of g_i broadcast after $t - ua_i$. Because $ka_i > ua_i$, $t - ka_i < t - ua_i$. If k is not changed, u can decrypt data items of g_i broadcast between $t - ka_i$ and $t - ua_i$, and thus past confidentiality at $t - ka_i$ is compromised.

Therefore, if at least one pair of $(ka_i, ua_i)_{k,u}$ satisfies $ka_i > ua_i$ at current time t, k in an enroll path needs to be changed before being distributed to user u to ensure past confidentiality.

Proof of the necessary condition: The necessary condition is that if all pairs of $(ka_i, ua_i)_{u,k}$ satisfy $ka_i \leq ua_i$, k does not need to be changed. If k is the DEK of g_i, the proof is obvious. If k's age is younger than user's subscription age, the user has already known all data items encrypted by k. Hence, k does not needs to be changed.

If k is not a DEK, select any program g_i that shares k. Let k_i be the DEK of g_i, and the latest renew spot of k is t_k. We use reduction to absurdity to prove. The opposite of the necessary condition is that past confidentiality for program g_i will be broken if all pairs of $(ka_i, ua_i)_{k,u}$ satisfy $ka_i \leq ua_i$ at current time t, and k is sent to u without any changed.

According to Definition 7, u joined g_i at $t - ua_i$ and is allowed to decrypt data items of g_i broadcast after $t - ua_i$. If past confidentiality for program g_i is compromised, u must have derived k_i's value before $t - ua_i$. Because $ka_i \leq ua_i$, $t - ka_i \geq t - ua_i$. u must have derived k_i's value at a time point t' before $t - ka_i$, i.e. $t' < t - ka_i$. Hence, u must have found the refreshments from all previous broadcast rekey messages: $\delta(k_\alpha, t_\alpha; k, t_k), ..., \delta(k_i, t'; k_\beta, t_\beta)$.

According to Definition 5, t' is a revive spot of k. If $ka_i = 0$, no revive spot exists after t_k, and thus t' cannot exists. If $ka_i > 0$, t' is a revive spot after k's latest renew spot t_k, and thus $t_k < t' < t - ka_i$. However, according to Definition 6, there cannot be any revive spot of k between t_k and $t - ka_i$. Hence, t' cannot exist, and the opposite of the necessary condition is false. Consequently, past confidentiality for any program g_i will not be broken if the pair of $(ka_i, ua_i)_{k,u}$ satisfies $ka_i \leq ua_i$ at current time t, and k is sent to u without any changed.

Therefore, to ensure past confidentiality, k in an enroll path needs to be changed, if and only if at least one pair of $(ka_i, ua_i)_{k,u}$ satisfies $ka_i > ua_i$ at current time t. By applying Theorem of Critical Key (**TCK**), KTR in a broadcast server works as in Algorithm 3 upon a user event.

Algorithm 3 Algorithm of KTR in Broadcast Server

1: **if** a join or shift event happens **then**
2: according to TCK, find all critical keys in the tree the user wants to join or shift to;
3: select the best enroll path that has the minimum number of critical keys;
4: change all critical keys in the best enroll path, and broadcast corresponding rekey messages;
5: **end if**
6: **if** a leave or shift event happens **then**
7: change all keys in the leave path, and broadcast corresponding rekey messages;
8: **end if**
9: update renew, refresh and revive spots according to the latest rekey messages;

3.4.4 Examples of Critical Keys

COROLLARY 3.1. *When a user joins a tree, a key in the enroll path is a critical key if and only if one of the key's ages is greater than 0.*

Before a user joins the tree, his subscription ages for all of the programs sharing this tree are 0. Hence, if the age of a key in the enroll path for this program is greater than 0, the key is older than the user's subscription. According to Theorem 3.1, the key needs to be changed before being distributed to the user.

For example, consider a user event where a user u_2 joins tree tr_4 at time t_2 in Figure 3.4. Assume that before t_2, another user u_1 shifts from tree tr_4 to tr_6 at t_1, and no other event happens between t_1 and t_2. At t_1, assume k_{n_L} and k_{r_4} are in the leave path. At t_2, assume k_{n_L} and k_{r_4} are in the enroll path. Now, we determine which keys are critical at t_2 according to Corollary 3.1.

At t_1, because u_1 is still in g_1 and g_2, the broadcast server does not need to change k_{r_1} and k_{r_2}, and only needs to send the refreshments $\delta(k_{n_L}, t_1; k_c, t_{k_c}), \delta(k_{r_4}, t_1; k_{n_L}, t_1)$, where k_c is a child key of k_{n_L} and not known by u_1. According to Definition 5, no revive spot is added

to k_{n_L} and k_{r_4}, and these two keys are renewed after t_1. Consequently, according to Definition 6, at t_2, the ages of both k_{n_L} and k_{r_4} are 0. The server can give k_{n_L} and k_{r_4} to u_2 without any change according to TCK, since u_2 cannot derive k_{r_1} and k_{r_2} before t_2 from the previous refreshments.

In this example, k_{n_L} and k_{r_4} are not critical keys, although they are in the enroll path at t_2. However, k_{r_1} and k_{r_2} are critical keys at t_2 because their ages are greater than 0. The server needs to change k_{r_1} and k_{r_2} before distributing them to u_2. This shows that it is not necessary to change all keys in the enroll path when a user subscribes the broadcast data services as would be required in the traditional LKH approach.

COROLLARY 3.2. *After a user enrolls in a tree, all keys in the enroll path are not older than the user.*

According to Theorem 3.1, if a key is older than the user's subscription regarding a program, the key needs to be changed. Hence, at the time when the user enrolls in a tree, the keys, whose ages are older than the user, are renewed and their ages turns to be 0. If the key is not older than the user's subscription regarding any program, the key does not need to be changed. Hence, the key continues to be not older than the user. Therefore, after a user enrolls in a tree, all keys in the enroll path are not older than the user.

COROLLARY 3.3. *When a user shifts from a tree to another tree, the keys overlap both trees do not need to be changed.*

Assume the user shifts from tree tr_α to tree tr_β. According to Corollary 3.2, after the user enrolls in tr_α, all keys in the enroll path cannot not be older than the user. Hence, when the

user shifts to tr_{β}, the overlapped keys, which were in the enroll path when user enrolled in tr_{α}, do not need to be changed according to Theorem 3.1.

For example, in Figure 3.4, user u shifts from tree tr_4 to tr_6 at t_1. Assume user u_1 is in tr_4 since t_0, i.e. u can decrypt data items of g_1 and g_2 since t_0. k_{r_1} and k_{r_2} are the overlapped keys with tr_4 and tr_6. Because u is still in g_1 and g_2 after he shifts to tr_6, k_{r_1} and k_{r_2} do not need to be changed.

3.4.5 Other Issues

In this part, we address major issues when implementing KTR in a broadcast server. One concern lies in the fact that wireless communication is inherently unreliable. Moreover, users may move out of service areas or intentionally shut down their mobile devices to save power. To simplify our discussion and analysis, we assume that the broadcast channel is reliable and that users will not miss keys. Nevertheless, much research has been conducted on helping users recover their keys [59, 41, 33, 28] which can also be added into our schemes. Furthermore, in our broadcast model, a user can occasionally use the uplink channel (or via wirelines) to request lost keys. Recovering lost keys may introduce overhead into the system, but this overhead is more relevant to the frequency that keys are lost rather than the security of KTR, and hence is out of scope of our research.

We also notice that the definition of refreshment is semantically related to the form of a rekey message when a user leaves a tree. In the situation where a rekey method [45, 59, 41] other than the LKH is used, the definition of refreshment needs to be changed. Note that, since a refreshment of a key always helps a user to figure out keys in the key's upper-level keys,

the refreshments in different rekey methods are equivalent regarding to whether they contain information that potentially compromises past confidentiality.

3.5 Performance Evaluation

3.5.1 Comparison Targets

Shared key and *critical key* are two important ideas we developed to improve the performance of key management in wireless broadcast systems. In the evaluation, we compare KTR with three other representative schemes: SKT, eLKH, LKH, as listed in Table 3.2. SKT is the approach in [49] where only shared key tree is applied. eLKH is an approach where only critical key is applied to enhance LKH. Neither shared key tree nor critical key is adopted in LKH. If shared key tree is adopted, key management is based on the key forest as illustrated in Figure 3.4; otherwise, a key tree is created for each program and a user is assigned to all trees corresponding to the programs he subscribes. If critical key is used, a key in an enroll path is changed if and only if it is a critical key; otherwise, all keys in the enroll path need to be changed (as in the other old schemes).

Table 3.2 lists four schemes representing different solutions which may or may not adopt these two ideas. The names of the schemes are self-explanatory. If key tree reuse is adopted, key management is based on the key forest as illustrated in Figure 3.4; otherwise, a key tree is created for each program and a user is assigned to all trees corresponding to the programs he subscribes. If critical key is used, a key in an enroll path is changed if and only if it is a critical key; otherwise, all keys in the enroll path need to be changed (as in other schemes). Note that the well-known LKH is used as a base line (i.e. neither key tree reuse nor critical key is adopted).

Table 3.2. Key management schemes

Schemes	*shared key*	*critical key*
KTR	Y	Y
SKT	Y	N
eLKH	N	Y
LKH	N	N

3.5.2 Simulation Factors

Two categories of factors might influence the performance of the proposed key management approach. The first is the structure of subscription, which includes the number of different programs, the number of different subscriptions, the number of users, the distribution of users in the programs, etc. For example, some programs may be very attractive and popular and thus be subscribed by many users. The second category is the patterns of user behavior. For example, the frequency of a user joining/leaving/shifting trees varies over time. When a new program is published, join will dominate user behavior. Then, as the program matures, the join and the leave rates will even out stabilizing the number of users.

3.5.3 Performance Metrics

On the server side, we can measure the main performance metrics for computation, energy consumption and memory space: *total rekey message size per event*, *average number of encryption per event*, and *total number of keys to be stored*. Since a server generally is abundant in energy and memory, its computation capacity could be the first important factor that can affect the performance of the whole system. If the processing time for each event is long, this would delay user's request.

At the client side, three main performance metrics need to be measured: *average rekey message size per event*, *average number of decryption per event per user*, and *maximum number of keys to be stored*. The average rekey message size per event, measured as the number of encryptions $\{*\}_k$ in the rekey message, is used to represent the communication cost. The average number of decryptions per event per user measures the computation cost and power consumption on a mobile device, since the device needs to decrypt new keys from rekey messages. The maximum number of keys to be stored obviously represents the memory requirement on the mobile devices. Based on these three metrics, we can infer other metrics that are more directly related to the mobile devices. For example, if we decide a particular encryption algorithm, we know the length of a key, the time to compute a key, and the energy consumption to execute the algorithm. Consequently, we can obtain metrics, such as the communication overhead in the rekey messages, etc.

3.5.4 Server Side

3.5.4.1 Analysis

We define the following parameters for the analysis.

- The service provides m programs;
- n users subscribes the service, and $2^m \gg n$;
- n users are distributed to e trees, and $e \leq n$;
- Each tree is shared by d programs on average, and $d \geq 1$;
- The leave rate is λ_l;

- The join rate is λ_j;
- The shift rate is λ_s;
- The total event rate $\Lambda = \lambda_l + \lambda_j + \lambda_s$

It is obvious that the overhead of KTR at the server side is determined by the number of keys the server needs to manage. Since a tree has $\frac{n}{e}$ users in average, there are $O(2\frac{n}{e}-1)$ keys to be handled, including the root. For e trees, the server needs to manage $O(e(2\frac{n}{e}-1)) = O(2n-e)$ keys. Because a key in any tree needs to keep a separate spot series for each program that shares the tree and each tree is shared by d programs on average, the server needs to keep $O((2n-e)d)$ spot series. In the multi-layer root graph, the server needs to manage DEKs and other non-root keys. Because one program is connected with $O(\frac{ed}{m})$ trees, the server needs to manage $O((\frac{ed}{m}-1)m) = O(ed-m)$ keys (including DEKs and non-root keys) for m programs. We notice that any one of these keys is connected with only one program. Hence, each key requires only one spot series, and thus the server needs to keep $O(ed-m)$ spot series. In total, the server needs to manage $O(2n-e+ed-m)$ keys with a storage requirement of $O((2n-e)d+ed-m) = O(2nd-m)$.

In a rekey operation, the server needs to inspect keys in the leave path and the enroll path, and then update renew, refresh and revive spots for affected keys. The leave path simply consists of keys that the user will no longer hold, and thus the server can easily find out the leave path. Nevertheless, the server needs to inspect the to-be-joined tree and find out the enroll path with the minimum number of critical keys. Hence, according to Algorithm 3, the computation

complexity on the server side is mainly determined by the number of keys to be inspected and the number of updates to be committed[1].

The number of keys to be inspected is determined by the number of users in the to-be-joined tree and the number of programs that share that tree. Inside the tree, there are $O(2\frac{n}{e})$ keys. The root of the tree is connected to DEKs of the programs that share the tree. As in Figure 3.5, the path between the root and one DEK has $O(log_2(\frac{ed}{m}))$ keys on average. Hence, there are $O(dlog_2(\frac{ed}{m}))$ keys between the root and the connected DEKs. In total, the server needs to inspect $O(2\frac{n}{e} + dlog_2(\frac{ed}{m}))$ keys in order to find the best enroll path and the critical keys.

The number of updates to be committed is determined by the number of DEKs that must be changed. In the worst case, all DEKs need to be changed and all keys under the DEKs need to be changed. Because each tree is shared by d programs on average, one DEK is connected with $\frac{de}{m}$ trees. Hence, in the worst case, a changed DEK will require the server to update all keys in the trees that are connected with the DEK. Hence, for each DEK, $O(2\frac{de}{m}\frac{n}{e}) = O(2\frac{dn}{m})$ keys need to be updated. In the worst case (i.e. m DEKs need to be changed), the server needs to commit $O(2dn)$ updates.

3.5.4.2 Simulation

We did simulations on a server (a computer with a 2GHz CPU and 2GB RAM running Linux). On average, the server uses tens of milliseconds for one rekey operation in the KTR scheme. Table 3.3 shows the simulation results for two services with 10000 users. The first row is a small service that provides only 5 programs and 31 valid trees[2], and each tree is shared

[1] One update means the server logs a spot to a key.

[2] A valid tree is a tree that has at least one user.

by 2.5 programs on average. The second row is a large service that provides 50 programs and 300 valid trees, and each tree is shared by 12 programs on average. Column "inspection" is the estimate of the number of keys to be inspected in the worst case. Column "update" estimates the number of updates to be committed in the worst case. Column "computation" measures the average computation time for one rekey operation in simulation.

Obviously, the computation time on the server side is mainly determined by the number of updates to be committed in the simulations. First, since the number of users is fixed in these simulations, when the service has more trees, the average number of users in one tree decreases. Accordingly, the server takes less time to inspect keys in the large service than in the small service. Hence, the server spends the most time on updating spots for affected keys. Since a program is connected with more trees in the large service, a changed DEK will affect more keys. According to the estimation, in the worst case, the server needs to commit updates in the large service $\frac{240000}{50000} = 4.8$ times as much as in the small service. Overall, as measured, the server's computation time in the large service is $\frac{68}{16} = 4.3$ times as much as in the small service.

Table 3.3. Computation at the sever side

service	users n	programs m	trees e	degrees d	inspection	update	computation (ms)
small	10000	5	31	2.5	655	50000	16 ± 2
large	10000	50	300	12	141	240000	68 ± 16

3.5.5 Client Side

3.5.5.1 Analysis

The main goal of this evaluation is to examine the overhead of KTR on resource limited mobile devices in terms of communication, storage, power consumption, computation, etc, which can be captured by three metrics:

- Average rekey message size per event,
- Average number of decryptions per event per user,
- Average number of keys hold per user.

The first metric, measured as the number of encryptions $\{*\}_k$ in the rekey message, represents communication cost and power consumption in a mobile device. In a leave event, the rekey message consists of rekey items for keys in the leave path. Hence, the number of encryptions is $O(2(log_2(\frac{n}{e}) + dlog_2(\frac{de}{m})))$. In a join event, the rekey message consists of rekey items for critical keys in the enroll path. Hence, the number of encryptions is $O((1+\rho)(log_2(\frac{n}{e}) + dlog_2(\frac{de}{m})))$, where ρ is the ratio of critical keys over all keys in the enroll path. Our simulation shows that ρ is around $22.9 \pm 1.8\%$ in an enroll path. In a shift event, the rekey message consists of rekey items for keys in the leave path and critical keys in the enroll path. Hence, the number of encryptions is $O((3+\rho)(log_2(\frac{n}{e}) + dlog_2(\frac{de}{m})))$. Considering user activities, the average rekey message size would be $O(\frac{1}{\Lambda}(2\lambda_l + (1+\rho)\lambda_j + (3+\rho)\lambda_s)(log_2(\frac{n}{e}) + dlog_2(\frac{de}{m})))$.

The second metric measures computation cost and power consumption in a mobile device, since the device needs to decrypt new keys from rekey messages. Define the height of a key as its distance to the bottom of a tree. For example, if a tree has n users, the height of its root

key is $log_2 n$. Obviously, if a key's height is h, 2^h users needs to decrypt it when it is updated in a rekey event. Hence, in a tree of n users, when a path of keys need to be changed, the total number of decryptions is $O(\sum_{i=1}^{log_2^n} 2^i) \approx O(2n)$. In a leave event, because there are $O(\frac{n}{e}\frac{de}{m})$ users under a to-be-changed DEK, the total number of encryptions for this DEK and its downward keys is $O(2\frac{nd}{m})$. Because d DEKs need to be change in a leave event, $O(2\frac{nd^2}{m})$ decryptions are committed by affected users. Similarly, in a join event, the total number of decryptions is $O(2\rho\frac{nd^2}{m})$. In a shift event, the number of decryptions is $O(2(1+\rho)\frac{nd^2}{m})$. Considering user activities, the average number of decryptions would be $O(\frac{1}{\Lambda}(2\lambda_l + 2\rho\lambda_j + 2(1+\rho)\lambda_s)\frac{nd^2}{m})$. Hence, on average, each user decrypts $O(\frac{1}{\Lambda}(2\lambda_l + 2\rho\lambda_j + 2(1+\rho)\lambda_s)\frac{d^2}{m})$ rekey items upon receiving a rekey message.

The third metric represents the storage demand which is proportional to the number of keys a user needs to hold. Inside a tree, a user needs to hold keys along the path from its leave node to the root of the tree. Because a tree has $O(\frac{n}{e})$ users, a user holds $O(log_2(\frac{n}{e}))$ keys inside a tree. In the root graph, a user hold keys from the root of the tree to all DEKs of the programs that share the tree. Because a root is connected with d DEKs and each DEK is connected with $O(\frac{ed}{m})$ trees, a user holds $O(dlog_2(\frac{de}{m}))$ keys in the root graph. In total, a user should hold $O(log_2(\frac{n}{e}) + dlog_2(\frac{de}{m}))$ keys.

3.5.5.2 Simulation

Settings The analysis gives estimates on major performance metrics. We notice that some factors could bring more insightful results. For example, users may not be evenly distributed in trees due to the fact that some programs may be more popular than other programs. Also, users may be more or less likely to stay in current subscriptions than to frequently change their

subscriptions. Hence, in the following, we conduct simulations to examine the impacts of these user behaviors. We assume that the server provides 50 programs. In our experiments, the key forest consists of 300 trees when shared key tree is adopted. Each tree represents a unique option of subscriptions. We also assume that there are 10000 users (on average) subscribing the services. All three user events (i.e. join, shift and leave) are modeled as independent Poisson processes. We let $\lambda_l = \lambda_j$ so the total number of users remains constant (i.e. around 10000). We vary λ_l and λ_s separately in order to observe their impacts on the rekey performance. The result of our performance comparison is obtained by averaging the rekey cost over 3000 random user events. Here, a user event is referred to an event in schemes that adopt shared key tree. Such an event is mapped into several user events in schemes that do not adopt shared key tree. For example, a user joins a tree of multiple programs in KTR is mapped as a sequence of events in LKH (each is corresponding to the user joining a tree of these programs).

Table 3.4. Cases in key management

Case	Major subscriptions	Major events
Case I	Multiple	Join and leave
Case II	Single	Join and leave
Case III	Multiple	Shift
Case IV	Single	Shift

Four test cases are generated for evaluation based on major subscriptions and major events (summarized in Table 3.4). In Case I and Case III, 80% of the users subscribe to multiple programs and the other 20% only subscribe to one of the programs. In Case II and Case IV, 20% of the users subscribe to multiple programs and the other 80% subscribe to only one program.

Furthermore, in Case I and Case II, the major events are joins and leaves; while in Case III and Case IV, the major events are shifts. In the simulations, we vary the rates for the major events while keeping the other rates at 1.

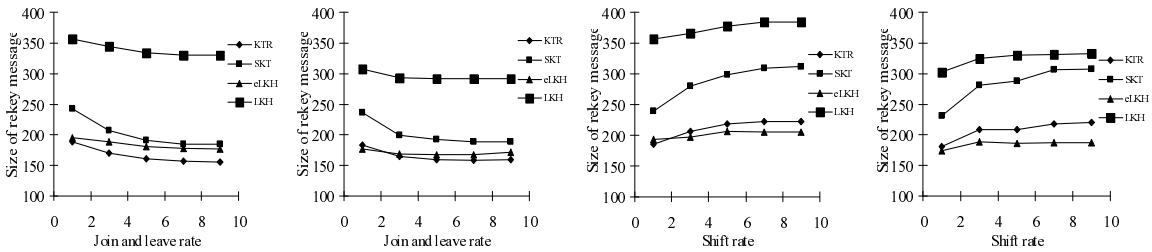


Fig. 3.9. Average rekey message size per event

Average Rekey Message Size Per Event We first evaluate performance of the key management schemes in terms of average rekey message size, by fixing $\lambda_s = 1$ and varying λ_l (x-axis) from 1 to 9 as shown in Figure 3.9(a) and (b). KTR and SKT that adopt shared keys significantly outperforms LKH, Because the major user events are join and leave, a user only needs to join or leave one tree when he subscribes or unsubscribes to multiple programs. In contrast, LKH that does not have shared keys requires a user needs to join or leave multiple trees for multiple programs. Furthermore, KTR is better than SKT, because critical key in KTR allows the server to change less keys.

Then, we evaluate performance of the key management schemes by fixing $\lambda_l = 1$ and varying λ_s (x-axis) from 1 to 9 as shown in Figure 3.9(c) and (d). Obviously, when the shift rate

grows, the performance of SKT turns worse because of the extra overhead that is introduced in managing shared keys when a user quits or adds some but not all of his subscribed programs. Obviously, the adoption of critical keys incurs the major improvement in terms of rekey message size. KTR and eLKH are the best solutions.

By comparing CASE III and CASE IV (corresponding to the subscriptions of multiple programs and single program, respectively), we found that the extra overhead of shift is higher in CASE III. Figure 3.9 also shows that KTR and STK are more sensitive to these types of major user events than eLKH and LKH. The average rekey message size in KTR and STK grows as the shift rate increases and shrinks as the join/leave rate increases. On the contrary, the average rekey message size in eLKH and LKH remains almost flat regardless of the rate of user events

Based on our experimental results, by adopting only shared key tree, SKT reduces the rekey message size to around 60% to 90% of LKH. By adopting critical keys alone, eLKH reduces the rekey message size to around 55% to 65% of LKH. This result also validates our claim that many keys in the enroll path do not need to be changed. In fact, over all the experiments, only around 23% keys in the enroll path need to be changed.

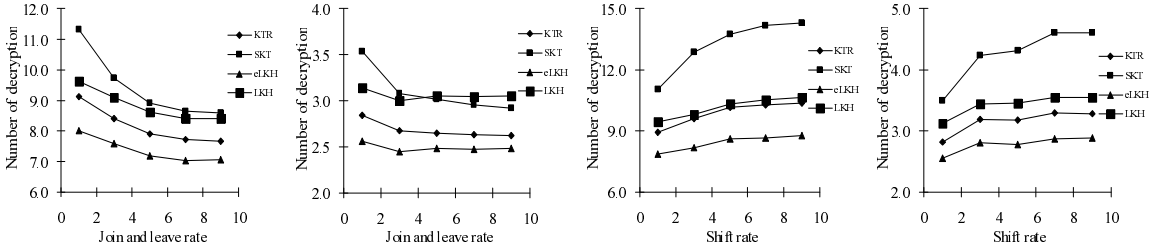


Fig. 3.10. Average number of decryption per event per user

Average Number of Decryption Per Event Per User Power consumption and computation cost are two primary concerns for mobile users. We use the average number of decryptions to measure these costs. Similar to the experiments in the previous section, we vary the rates of major events to observe their impacts on decryption overhead. Figure 3.10 shows that the schemes adopting critical keys (i.e. KTR and eLKH) are better than SKT and LKH. The number of decryption in eLKH is around 80% of that in LKH, and KTR's is around 80% of SKT's too. In all schemes, the number of decrptions drops as the join/leave rate increases and rises as the shift rate increases.

However, the adoption of shared key has negative impacts on reducing user's decryption cost. In Figure 3.10, LKH is better than SKT, especially when the shift rate is high. So does eLKH in comparison with KTR. This can be explained as follows. In shared key schemes, when a user shifts from a tree to another, some users will be affected by the changed keys in both the leave and the enroll paths if they subscribe to the programs that the user keeps subscribing to. As previously discussed, shared key schemes introduce extra rekey cost because they change keys in two paths. On the contrary, in non-shared key schemes, no key needs to be changed for the programs the user keeps subscribing to, Hence, those users who are affected by keys in two paths in shared key schemes only need to decrypt keys in the leave path in non-shared key schemed. As a consequence, the decryption cost per user is less in non-shared key schemes that that in shared key schemes.

Figure 3.10 also shows that the user subscription pattern has a great impact on the average number of decryptions. The average number of decryptions in Case II and Case IV (where only 20% of users subscribe multiple programs) is around 55% of that in Case III and Case IV (where

80% of users subscribe multiple programs). Obviously, if a user subscribes to more programs, it is more likely that he will be affected by other user activities.

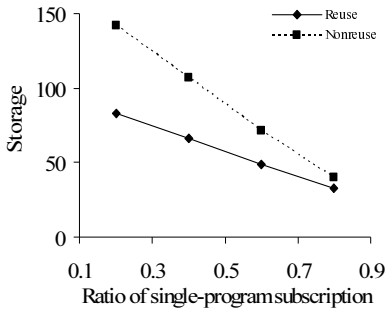


Fig. 3.11. Average number of keys hold per user

Average Number of Keys Held Per User Finally, we evaluate the storage demand on mobile devices, which is measured as the average number of keys held by each user. One goal is to save storage by reducing the number of keys each user needs to hold. Since KTR makes programs share keys, KTR saves storage for a user when the user joins a tree shared by more programs. As analyzed before, the structure that programs share trees determines the number of keys that can be saved. However, since users may favor some subscriptions, users may concentrate in some trees. For users subscribing to single programs, KTR has no advantage over LKH.

The average storage demand is also affected by how users are distributed in trees. In Figure 3.11, we vary the ratio of users who only subscribe single programs and illustrate the average storage demand. Obviously, when most users subscribe to multiple programs (i.e. only 20% users subscribe single programs), KTR can save 45% storage on average for each user

compared to LKH which assigns a separate set of keys to each program. When more users (80% users) subscribe to single programs, the storage reduction in KTR is only 10%.

In summary, KTR combines the advantages of both shared key tree and critical key. Among all schemes, it has a light communication overhead (i.e. its average rekey message size per event is the least or close to the smallest), incurs less computation and power consumption on mobile devices than the other schemes (i.e. its average number of decryption per event per user is the smallest), and requires least storage in mobile devices (i.e. its average number of keys held per user is the smallest). Because a mobile receiver generally only has limited resources, such an overhead saving can greatly benefit the receivers so that they can have a longer working duration and more computation capacity to process broadcast data.

3.6 Conclusion

In this chapter, we investigated the issues of key management in support of *secure* wireless data broadcast services. To the best knowledge of the authors, this is the first research conducted in the field of subscribe-publish based wireless data broadcast services. We proposed KTR as a scalable, efficient and secure key management approach in the broadcast system. We used the key forest model to formalize and analyze the key structure in our broadcast system. KTR let multiple programs share a single tree so that the users subscribing these programs can hold less keys. In addition, we proposed an approach to further reduce rekey cost by identifying the minimum set of keys that must be changed to ensure broadcast security. This approach is also applicable to other LKH-based approaches to reduce the rekey cost as in KTR. Our simulation showed that KTR can save about 45% of communication overhead in the broadcast channel and about 50% of decryption cost for each user, compared with the traditional LKH approach.

Chapter 4

Analysis of Packet Injection in Ad Hoc Networks

4.1 Introduction

Many researches have been done on security problems in ad hoc networks [1, 57, 43, 18, 2, 4, 19]. They have identified various attack approaches in transport, network, MAC and physical layers, which can be exploited for denial-of-service (DoS) attacks in ad hoc networks. Due to the unique features of ad hoc networks, DoS attacks in an ad hoc network may present new properties and thus need more researches. First, ad hoc nodes are peers. If a node is compromised, it can attack the network from inside. Second, every node in an ad hoc network is not only a host but also a router. It is harder to determine whether a suspicious packet is from an attacker or relayed from a good node. These features indicate that there may be "easier" ways to cause DoS in ad hoc networks.

In this study, we investigate packet injection attacks, that target at network wide congestion. Congestion has already been recognized as a simple and effective DoS attack approach. In the physical layer, jamming [43] can disrupt and suppress normal transmission. In the MAC layer, Wullems *et al.* [57] identified a congestion approach by deceiving normal nodes that the channel is busy. Due to the limited wireless bandwidth, congestion can also be easily caused by high volume of packets in other attacks. When routing request packets or TCP SYN packets are flooded in an ad hoc network, the network can be congested by these packets. However, it is still not clear whether and how these attack approaches can be exploited by distributed attackers in

an ad hoc network, how bad the packet injection attacks could be in an ad hoc network, and what the unique characteristics of packet injection attacks in ad hoc networks are.

Previous studies on DoS attacks are not enough to show the real impacts of DoS attacks on an ad hoc network. First, most studies focused on individual attackers and the attack impacts on individual nodes and traffic flows. However, in a DoS attack, the attackers may group together to flood in the network simultaneously. If the attackers are distributed across the whole network, most normal traffic and nodes will be affected. Second, most attack approaches in ad hoc networks need specialized hardware, which may not be realistic for DoS attackers. However, packet injectors simply control some nodes to flood the network by sending spoofed packets. They may not be able to modify the network cards of the controlled nodes, and thus cannot launch such attacks as dropping, modifying or reordering selected TCP, routing or MAC packets, as described in [1, 18, 2, 4, 19].

In this study, we investigate some new attack approaches that can be easily enforced in compromised nodes or special nodes that are intentionally deployed in the network. These approaches follow the network and communication protocols and do not require specialized hardware or modifications of hardware. We also analyzed various important factors that may affect the attack impacts. We found that it is light-weight for attackers to launch a packet injection attack. This study contributes to the literature by demonstrating the consequences of network wide packet injection under various factors.

The rest of the chapter is organized as follows. In section 4.2, we overview packet injection attacks in ad hoc network, and overview the features of packet injection attacks targeting at network wide congestion. In Section 4.3, the remote attacks are analyzed and simulated. In

Section 4.4, the local attacks are analyzed and simulated. Finally, we conclude this study in section 4.5.

4.2 Background and Problem Statements

4.2.1 Packet Injection Attacks

The literature has showed many DoS approaches as previously discussed in Section 2.3. This study focuses on another type of DoS attack in the network layer where a malicious forwarding node injects junk packets in a route by impersonating the source or the other forwarding nodes of the route. Differing from the packet dropping attacks, the injected packets can congest the nodes in the area nearby the route. Even when they target only a single server, the attacking traffic can still have DoS impact on the areas the traffic goes through instead of only the vicinity of the target server. Hence, we name these attacks as area-congestion-based packet injection attacks in this study. Furthermore, since the attacking nodes will largely follow the network and communication protocols, it is relatively more difficult to detect and block such an attack. We believe that from the point of view of many (not-so-sophisticated) attackers, the identified attacks can be more feasible and effective. In this section, we first give an overview of area-congestion-based packet injection attacks. Then, in the following sections, we give the analysis in detail.

4.2.2 Attack Topologies

The packet injection attacks are illustrated in Figure 4.1, where the big ellipse area is an ad hoc network, and a1, a2 and a3 are the attackers. The attackers send traffic toward nodes n1, n2 and n3. The dashed lines stand for the attack traffic through multiple hops, and the solid lines

show that the attackers send traffic to their nearby nodes in one hop. The shadowed areas are possible congested areas. Note that all nodes in the figure are inside a network and function as both hosts and routing nodes. They are interconnected and able to control network protocols. As a contrast, in the Internet, attackers have very limited ability to launch attacks in the routing and the link layers, because they are at the perimeter of the core network and access the Internet through their access networks.

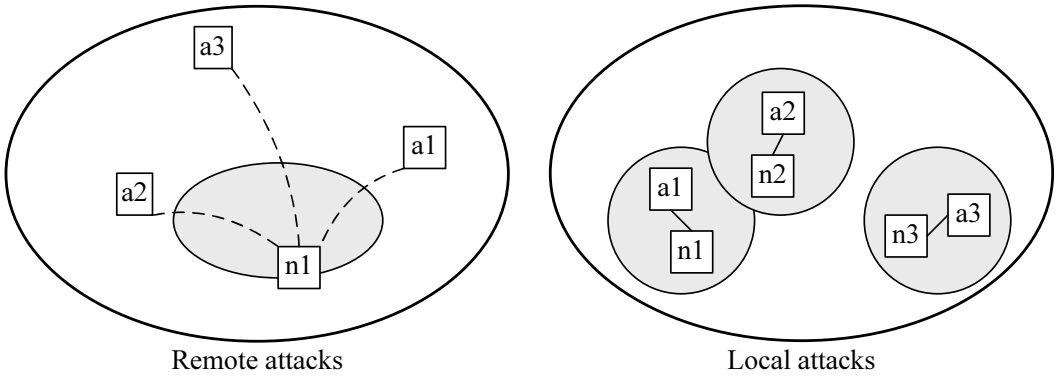


Fig. 4.1. Packet injection attacks

The depicted packet injection attacks are classified as *remote attacks* and *local attacks*. The approaches for each type of attacks will be addressed in later sections. The remote attacks are different from flooding in the Internet. In the Internet, a congested link keeps its maximum throughput during each attack period. However, in ad hoc networks, because the communication channel is open and shared, packets in a small area can collide with each other. Hence, different attack streams interfere with each other when they go through the same area. In addition, the

transmission of a routing node in an attack stream may interfere with another routing node. The attack stream may experience self-congestion and its route may break before it reaches the target. As a consequence, it is largely unpredictable which routing nodes may forward the flooding packets or how many flooding packets can reach the target through multiple hops. It is thus largely unpredictable which node is interfered and how bad the target is congested. Based on the simulations, it is found that more flooding nodes and higher attack load may in fact reduce the attack impacts in a remote packet injection attack.

The local attacks directly affect the nodes near the attackers. Compared with remote attackers, local attackers may suffer less self-congestion and better congest their nearby targets. Local attackers are competing for the channel with all other nearby nodes. To cause the damage, they need to use appropriate values of traffic parameters to effectively exhaust the channel. Furthermore, attack packets can impact all nodes near the attacker, since all nearby nodes should avoid an in-transmission packet. Consequently, to affect a nearby target, a local attacker can send the traffic to another nearby node instead of directly sending to the target. If two or more attackers are neighbors, they may only flood to each other. In this way, more traffic can be generated to congest the target. Based on the simulation, the impact of a local packet injection attack is increased with more flooding nodes and higher attack loads. If the number of flooding nodes and the attack loads are same, a remote packet injection attack can bring more damage to the network than a local packet injection attack.

4.2.3 Problem Statements

In this study, we mainly investigate attack constraints, attack strategies and attack impacts of packet injection attacks. Although it is claimed that such a packet injection attack can disrupt

the network traffic, it is not clear how serious the attack could be. Most previous studies on DoS attacks focused on the attack impacts on individual nodes and traffic flows. In terms of the whole network, it is not well studied how the attacking traffic can influence the legitimate traffic.

In addition, most previous studies focused on individual attackers. It is obvious that a single attacker may only affect a small part of the network, and thus its attack impact is limited. In a DoS attack, the attackers may group together to flood in the network simultaneously to affect most legitimate traffic and nodes. However, different attacking flows may interfere with each other, and thus the real attack impact may not be as serious as expected by attackers. In fact, various factors may affect the attack impacts. We found that it might be less devastating when attackers simply inject as many packets as possible.

To thoroughly study the attack impacts, we first present analysis for the understanding of the nature of attacks. Then, we use extensive simulation to illustrate and compare the attack consequences under various factors.

4.3 Remote Attacks

4.3.1 Attack Approaches

In the remote attack, the attackers flood a huge number of junk packets toward the service node over multiple hops. The attackers may risk exposing their true identities, since the victims in the congested area can easily trace back to the flooding sources in some routing protocols. For example, DSR [23] puts the route in the head of a data packet. A routing node follows the route in the head to forward the packet. It can also use the route in the head to trace where the packet comes from. Some other routing protocols, such as AODV [39], can also be improved to record

the source information. For example, during the routing discovery phase, a routing node can set up a routing entry pointing back to the source of the route after it gets the route reply message for the valid route. Then, the routing node can know the previous hop where the packet comes from and thus trace back to the source hop by hop. Hence, it is more likely that the attackers will impersonate other nodes and flood packets with spoofed source addresses in order to hide themselves.

To flood a network with spoofed source addresses, the attackers (or the compromised nodes) can take the following two steps. First, the attackers need to find valid routes toward the targets in the network. The attackers may follow the routing protocols to discover good routes by themselves. The attackers may also eavesdrop their nearby traffic, from which they can identify whether their neighbor nodes have valid routes. The attackers can also check their own routing entries and may find that they are already in valid routes. Then, the attackers inject junk data packets into the valid routes. In this step, the attackers spoof the source addresses in the junk packets, but they can claim that they are forwarding these packets from the claimed sources.

When a routing node receives the injected packets, it checks its routing table, finds the routing entry according to the destination addresses, and then forwards them, although the route is actually not set up for the claimed sources. If the routing node traces back according to the source address, it may trace back along another route, trace to the claimed source instead of the flooding source, or find the claimed source is invalid. The reason that the attackers can succeed in flooding is that discrepancies exist between routing and forwarding. Even if secure routing protocols [18, 17] are enforced in ad hoc networks, no further source verification is enforced in packet forwarding. If the victim can identify the flooding sources with some intrusion detection systems, he may not figure out where the packets come from.

The approach to flood packet as described above has no special requirement in a network card. The flooding node simply composes packets with spoofed source after obtaining a valid route. If the flooding node is more specialized than a compromised node, the attack can be worse. For example, the attacker can deploy nodes with special hardware and software. These nodes can not only flood by themselves, but also help to forward and redirect packets from other nodes. In particular, these nodes can function as the routing nodes to forward packets from a compromised node in case its packets are denied by other routing nodes.

4.3.2 Attack Constraints

4.3.2.1 Self Congestion

In the remote attack, attackers may not flood as fast as they want due to self congestion. Because a routing node shares the channel with another routing node in the same route, their transmission interferes with each other. The consequence is the transmission of a routing node can break the links in its upstream. If an attacker generates packets very quickly, most packets will be buffered in upstream nodes and dropped later due to link failures. Our simulation illustrates that attackers need to control the speed of packet generation to achieve the maximum throughput. The generation speed is measured as *generation gauge*, which is the multiplication of the average period to generate one bit and the total channel bandwidth. In our simulation, the channel bandwidth is $1Mbps$. If a node generates attack load at $50Kbps$, i.e. it generates one bit every $2\mu s$ in average, its generation gauge is 2. The quicker a node generates packets, the smaller the generation gauge is.

Figure 4.2 shows the relation between the achieved throughput of UDP traffic and the generation gauge in chain-like paths of different lengths. It depicts the curves for 5-hop, 10-hop

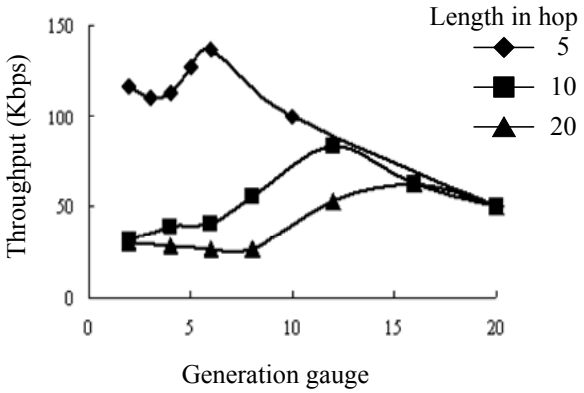


Fig. 4.2. UDP throughput in a chain-like path

and 20-hop paths respectively. Now, consider the curve for a 10-hop path. Its right-side slope illustrates a normal situation that slower packet generation, i.e. bigger generation gauge, results in less throughput. However, its left slope is special in that faster packet generation can reduce its throughput. Obviously, the maximum throughput is achieved at the best generation gauge, and they are inverse to each other. In addition, for paths of different lengths, the best generation gauge is different.

Based on extensive simulations, we find an empirical rule that, for a single UDP path, the best generation gauge is

- approximately 1.2 times of the hop number, if the length of the path is less than 12 hops;
- or around 15, if the length of the path is greater than 12 hops.

Due to the self congestion problem, the longer a path is, the less maximum throughput the UDP traffic has. As illustrated in Figure 4.2, if the path has 5 hops, the maximum throughput is around $145Kbps$. If the path has 10 hops, the maximum throughput is reduced to $80Kbps$. If the path has 20 hops, the maximum throughput is further reduced to $60Kbps$. Consequently,

if one attacker is flooding toward a target from a very far distance, the traffic that can actually reach the target is less than $60Kbps$ no matter how fast it generates packets. If an attacker wants less flooding nodes to cause the same damage on the target, he needs to let the flooding nodes be close to the target.

An attacker may also use TCP traffic for flooding. As stated in [12], if an optimum TCP window is adopted, the maximum throughput of TCP traffic can reach $200Kbps$, and it is less affected by the length of the path. Because the UDP flow has no congestion control mechanism, a single UDP flow seems to suffer from more self congestion and have less throughput than a single TCP flow. However, this does not mean a TCP flow can be better than a UDP flow for flooding. The current observations can only be applied to a single path which is not interfered by any other traffic. When numerous flooding streams aggregate, they will definitely interfere with each other. A UDP flow may further reduce its throughput because more packets are congested by other traffic. A TCP flow may also reduce its throughput because it may need to reestablish its connection frequently due to route failure, ACK packet loss, etc. We use simulations to study their different impacts later in Section 4.3.3.

4.3.2.2 Cross Congestion

Another constraint is that different traffic flows can interfere or congest with each other. Two traffic flows do not interfere only when any node in one flow cannot sense any node in the other flow. It is possible that two attack flows collide with each other before reaching the target. If both flows carry the maximum throughput, they will congest. If the change of routes in congestion is not considered, several attack flows may merge into one outgoing flow when they

collide in an area. Then the outgoing traffic may collide and merge with other traffic flows. Note that traffic mergence does not equal to throughput mergence.

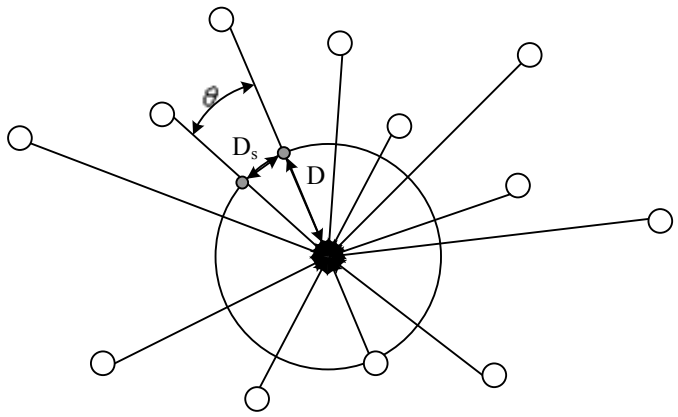


Fig. 4.3. Attack traffic collision

Consider Figure 4.3 where all attackers send traffic toward the target in the center. Assume that all attack sources are far away from each other, and able to find the best routes which directly point to the target. If the sensing distance is D_s and the average angle between every two closest routes is θ, at least one collision takes place at a location whose distance to the target satisfies $D \geq \frac{D_s}{2sin\frac{\theta}{2}}$. In another word, at the distance D from the target, maximally $N_D = \frac{\pi}{arcsin(D_s/2D)}$ flows can go through toward the target without collision. Accordingly, in remote attacks, the maximum number of non-interfered traffic flows entering the target's sensing range is 6 (where $D = D_s$), which is a lower bound on the number of attackers to circumvent the target from all directions. This lower bound indicates that (a) most attack traffic will be congested before reaching the target and (b) many areas, besides the target, are congested during the

attack. Hence, a remote attack can actually affect the whole network instead of the target node, although attackers may not know which area is actually congested.

This bound also indicates the specialties in remote attacks if attackers only target one service node. Fewer attackers are required to congest the target in an ad hoc network than in the Internet. Since nodes are peers in the channel, each node can exhaust the channel as much as the service node. Consider the situation where 6 flooding nodes are close to the target, for example, 3 hops away from the target. If each node floods at $150 - 200Kbps$, the total flooding traffic toward the service node is at a rate around the channel capacity $1Mbps$. If the flooding nodes are far away from the target, for example, more than 15 hops away, we need to consider the maximum throughput of a single UDP flow discussed in Section 4.3.2.1. Assume that the attackers are smart enough to select good flooding topology so that the flooding flows do not interfere with each other before reaching the target. 16 flooding nodes may be needed, since each of them can only get $50 - 70Kbps$ of flooding traffic to reach the target. However, in reality, because ad hoc routes are random, the attackers can hardly select such a good topology to avoid cross congestion. We use the simulations to study the impact of the number of flooding nodes on the target.

4.3.3 Simulations

In this section, we present details of our simulations in NS2 [35]. We study the attack impacts with different attack factors, and the attack impacts on different traffic of good nodes. Then, we summarize the attack properties of the remote packet injection attacks.

4.3.3.1 Simulation Settings

Communication model In the physical layer, the two-ray ground reflection model is used to model the signal propagation in the simulations. We choose the widely used IEEE 802.11 as the MAC and PHY protocols for the communications among mobile nodes. The CSMA and DCF functions are used to avoid transmission collision among nearby nodes. Each node has a sensing range of 550 meters and a transmission range of 250 meters. The maximum bandwidth of the channel is $1Mbps$. For communications over multiple hops, AODV is used as the routing protocol.

Network topology We simulate the attacks in a $4200m \times 4200m$ network. The network is divided into 441 grids, each of which is a $200m \times 200m$ square area. 441 nodes are put into these grids one by one. Inside a grid, a node is randomly placed. Under these conditions, the network topology is randomly generated for each simulation. We do not consider the movement of nodes in the simulations. In general, the motion of nodes is much slower than the dynamics of the network under attack, such as the frequent changes of routes during a packet injection attack.

Traffic model The node in the middle of the network is the service node, also referred as the server in our discussion. In each simulation, we use CBR agents to generate good traffic and flooding traffic. In each simulation, we randomly select 10, 20 or 40 nodes as the flooding nodes to send traffic toward the service node. The flooding traffic starts 5 seconds after the good traffic, and continues for 30 seconds. The load of a flooding flow is $20Kbps$, $50Kbps$, $100Kbps$ or $200Kbps$. In each simulation, all flooding streams have the same attack load. We compare the attack impacts under various traffic parameters and patterns. However, if it is not mentioned, the following default traffic setting is applied.

Default traffic setting The good traffic is generated by 20 good nodes and the service node. The good nodes are randomly selected. 10 good nodes communicate with the service node, and the other 10 randomly communicate with other nodes. 80% of the good traffic use TCP connections, and the remaining 20% use UDP packets. Each good traffic flow has a load of $20Kbps$. The flooding nodes are randomly put in the network. The flooding traffic use UDP packets.

Metrics We measure the throughput loss of the good traffic. The higher the throughput loss is, the less good traffic can reach the destination and thus the more damage the attacks cause. All good packets at the transport layer are recorded during the attack period. The throughput loss is measured as the percentage of the total bits in all dropped good packets over the total bits in all good packets during the attacks. Note that the throughput loss is related to many impacts in the application layer. It may cause extra delay of the service due to the retransmission of the lost packets. It may result in the disconnection to the service node if the service request packet is lost. We compare the throughput loss in different scenarios under different factors. In each comparison, we measure the throughput loss with different numbers of flooding nodes and different attack loads. Each point of the throughput loss in the comparison figures is the average of 4 independent simulations.

4.3.3.2 Positions of Flooding Nodes: random or ring

In ad hoc networks, the positions of flooding nodes may affect the attack impacts. The flooding nodes can gather in a certain area or be distributed across the whole network. In this comparison, we simulate two types of flooding positions. First, the flooding nodes may be randomly distributed in the network. This is a typical situation when some normal nodes are

compromised by the attackers for flooding. These compromised nodes may stay anywhere in the network and the attackers cannot control their positions. Second, the flooding nodes may be located in a ring circling the service node. The attackers can intentionally deploy some flooding nodes to circle the service node. In this comparison, the ring is a circle centered at the service node and has a radius of $1300m$. The flooding nodes are selected from the nodes on or close (within $200m$) to the ring. The ring is equally divided to these flooding nodes.

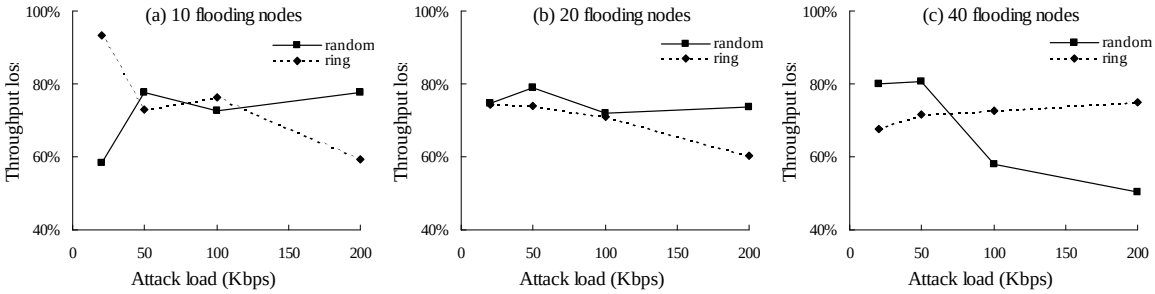


Fig. 4.4. Positions of the flooding nodes: random or ring

Figures 4.4(a), (b) and (c) illustrate the attack impacts as a function of the node position, the number of flooding nodes and the attack loads. In each figure, the solid lines stand for the throughput loss of the good traffic when the flooding nodes are randomly distributed in the network, and the dashed lines for the ring positions. For example, Figure 4.4(a) shows the throughput loss of the good traffic when 10 nodes flood in the network with two types of positions.

It is shown that the trend of throughput loss over attack load varies with different number and position of flooding nodes. First, let us look at the trend of throughput loss when the flooding

nodes are randomly put in the network. Figure 4.4(a) shows that the throughput loss grows from 59% to 77% as the attack load increases in the network of 10 flooding nodes. When more flooding nodes are put in the network, the trend of throughput loss turns flat around 74% as in Figure 4.4(b) (20 flooding nodes), and decreases from 80% to 50% as in Figure 4.4(c) (40 flooding nodes). This indicates that the density of flooding nodes changes the attack impacts when flooding nodes are randomly positioned. When the density is low, flooding nodes are more likely to be far away from each other, while a few nodes may be close to the service nodes. Since the flooding flows are far apart from each other, they may have less interfere with each other. As the attack load increases, more flooding traffic can exist in the network and affect the good traffic. On the contrary, when the density is high, the flooding nodes are close to each other. Based on the simulation results, self congestion and cross congestion have a significant impacts on the flooding traffic that many flooding packets are dropped before they can have impacts on good traffic and good nodes.

The ring type position shows another trend of the throughput loss. When the number of flooding nodes is small (10 in Figure 4.4(a)), the throughput loss drops from 93% to 77% as the attack load increases. The drop trend turns flat or a little reverse as the number of flooding node increases. When 20 flooding nodes are in the network as in Figure 4.4(b), the throughput loss drops from 75% to 60%. When the number of flooding nodes is 40 in Figure 4.4(c), the throughput loss grows a little around 70% as the attack load increases. Since all flooding nodes are put in the ring, the density of the flooding nodes only affects the ring area. Since the ring is not close to the service node, the throughput of an individual flooding flow drops as the attack load increases. Hence, when the density is low, the attack impact decreases as the attack load increases. When the density is high, the network inside the ring turns to be saturated by the

flooding traffic. As the attack load increases, the area near the service node is more congested and more good packets are lost.

4.3.3.3 Good Traffic Patterns: with the service node or random

In an ad hoc network, not all nodes connect with the service node. Two nodes may only communicate with each other. However, even they do not connect with the service node, their communication may still be congested by the flooding traffic. Hence, in this comparison, we study the difference of the attack impacts on two patterns of good traffic. One is the traffic that goes between the service node and normal nodes. In the simulations, we randomly set the direction of the traffic from or to the service node. The other type of good traffic is the traffic that is between two randomly selected nodes.

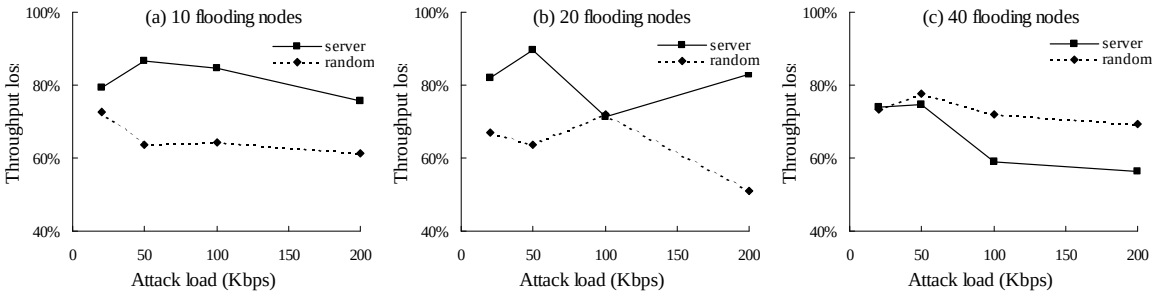


Fig. 4.5. Good traffic patterns: with the service node or random

Figures 4.5(a), (b) and (c) illustrate the attack impacts on the two traffic patterns. In each figure, the solid lines stand for the throughput loss of the good traffic that connects with the service nodes, and the dashed lines for the traffic between two randomly selected nodes.

The throughput loss of both patterns of good traffic has a decreasing trend as the attack load increases. The high end and low end of the throughput loss has a difference around 15%. However, it is noted that when normal nodes communicate with the server, the good traffic is more affected by the number of flooding nodes. When the number of flooding nodes is small as in Figure 4.5(a), the good traffic connecting with the service node can have more than 80% throughput loss. As the number of flooding nodes grows, the throughput loss drops to 70% or even less. As a contrast, the throughput loss of random good traffic keeps a similar dropping pattern from 70% to 60%, no matter how many flooding nodes are in the network.

This comparison indicates that the flooding traffic mainly affects the service node, especially when the number of the flooding nodes is small. Since all flooding flows go toward the service node, the flooding traffic concentrates in the vicinity of the service node. In other areas, especially those far away from the service node, the flooding traffic may not be so intense. Hence, the good traffic toward the service node is seriously congested. Because the random good traffic is affected when it interferes with the flooding traffic, it is less congested than the good traffic connecting with the service node. When the number of flooding node is large, the network is full of flooding traffic and thus any kind of good traffic will be congested. In this situation, the throughput loss of both types of good traffic in Figure 4.5(c) is more similar than in other Figures.

Note that, although the random good traffic is less affected, its throughput loss is still high, around 70% in average. Since some flows may still go through the vicinity of the service node or may be close to the flooding nodes, they are congested as the traffic toward the service node.

4.3.3.4 Good Traffic Loads: $20Kbps$ or $50Kbps$

So far, we only studied the attack factors and the attack impacts when the good traffic is generated at $20Kbps$ per flow. Since all nodes are equally competing in the network, the flooding traffic may have less impact when good traffic has a higher load. In this comparison, we study the impacts on good traffic with different loads. We let the good nodes have a higher load at $50Kbps$, and compare its throughput loss with the good traffic of lower load. In the generation of a higher load good traffic, we use larger data packets instead of faster packet generation. In particular, the good node or the service node generate one packet every $100ms$. If a good traffic flow has a load of $20Kbps$, the payload in each packet of this flow is 250 bytes. If the flow has a load of $50Kbps$, the payload in each packet is 625 bytes.

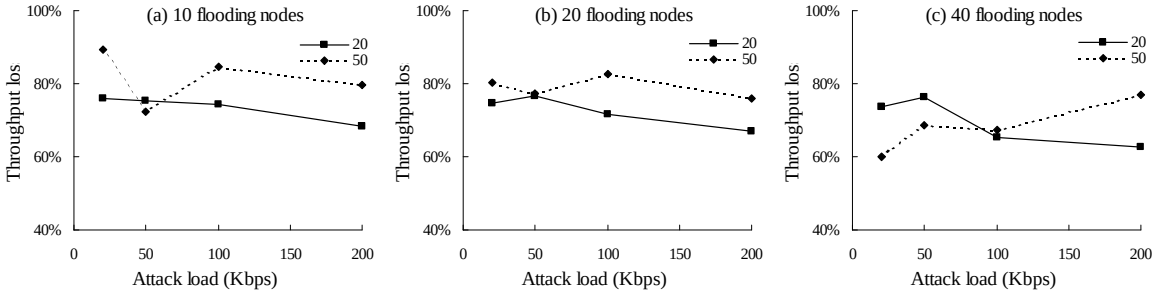


Fig. 4.6. Good traffic loads: $20Kbps$ or $50Kbps$

Figures 4.6(a), (b) and (c) illustrate the attack impacts on two loads of the good traffic. In each figure, the solid lines stand for the throughput loss of the good traffic at $20Kbps$ per flow, and the dashed lines for the good traffic at $50Kbps$ per flow.

The throughput loss of good traffic has different trends at different loads. When the load of good traffic is $20Kbps$, the throughput loss always drops from 75% to around 65% as the attack load increases. However, if the load of good traffic increases to $50Kbps$, the throughput loss drops from 90% to 80% only when the number of flooding nodes is 10 as in Figures 4.6(a), but keeps flat around 80% as in Figures 4.6(b), and even grows from 60% to 75% as in Figures 4.6(c).

Note that although the difference of the throughput losses of the good traffic with the different loads is obvious, it is not statistically significant (P-value is 0.104 in ANOVA test). Furthermore, the throughput loss of good traffic at $50Kbps$ is only 10% more than the good traffic at $20Kbps$. Hence, a higher load of good traffic can make more throughput, although the good traffic suffers from the higher throughput loss.

4.3.3.5 Flooding Flows: UDP or TCP

In all previous simulations, attackers flood UDP packets in the network. In reality, attackers may use two types of flooding traffic, UDP or TCP. When using TCP connections, each attacker intends to achieve his own maximum flooding throughput by reducing self congestion with TCP congestion control mechanism. As analyzed in previous sections, a single UDP flow may have less throughput than a single TCP flow because the UDP flow has no congestion control mechanism and suffers more from self congestion. However, in a packet injection attack, many flooding flows may have cross congestions. Since the channel capacity is fixed and both flooding nodes and good nodes are competing in the channel, the attack impact is mainly determined by how much flooding traffic can be successfully transmitted in the network. It is thus

important to compare them and know which kind of flooding traffic an attacker would like to use.

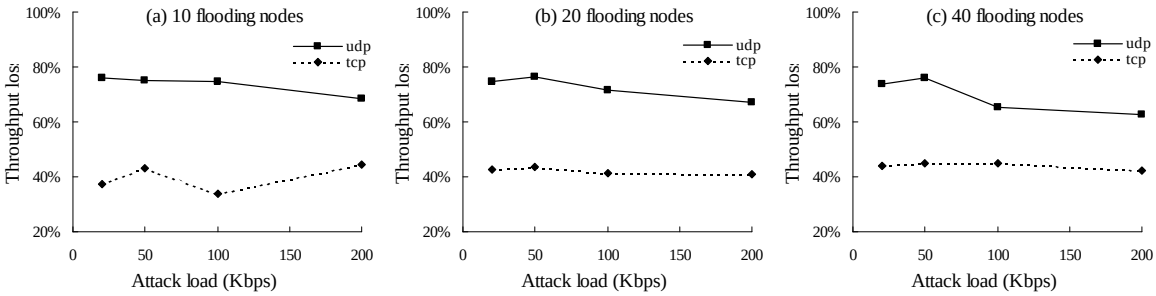


Fig. 4.7. Flooding flows: UDP or TCP

Figure 4.7(a), (b) and (c) depict the attack impacts of these two types of flooding traffic. In each figure, the solid lines stand for the throughput loss of the good traffic when the flooding nodes send UDP traffic at different attack loads, and the dashed lines for TCP flooding traffic.

The throughput loss of good traffic under UDP flooding traffic is 72%, while the throughput loss under TCP flooding traffic is only 42%. The throughput loss of the good traffic is significantly higher when the attackers flood UDP traffic (P-value is 0.000 in ANOVA test). Hence, with the congestion control mechanism in TCP, the attackers cannot get more flooding throughput in packet injection attacks. Instead, the total volume of the TCP flooding traffic is reduced more than the UDP traffic, and thus the attack impacts under TCP flooding traffic is decreased.

It is also noticed that the throughput loss under TCP flooding traffic is quite flat with different attack loads, which indicates that the throughput of good traffic is stable even though the attackers increase the attack loads. However, the throughput loss under UDP flooding traffic

decreases slightly as the attackers increase the attack loads. This contradicts our common sense, but conforms to the self congestion pattern in Figure 4.2. As the attackers increase the UDP flooding loads, more UDP packets are congested before they can be sent out to interfere other traffic.

4.3.3.6 Summary

Besides two types of flooding flows, we studied six factors in the simulations that may affect the attack impacts: the attack load, the number of flooding nodes, the position of flooding nodes, the load of good traffic, and the pattern of good traffic. In previous comparisons, we studied the interactions among these factors. In this part, we give an overall evaluation on the main effects of these factors as shown in Figure 4.8.

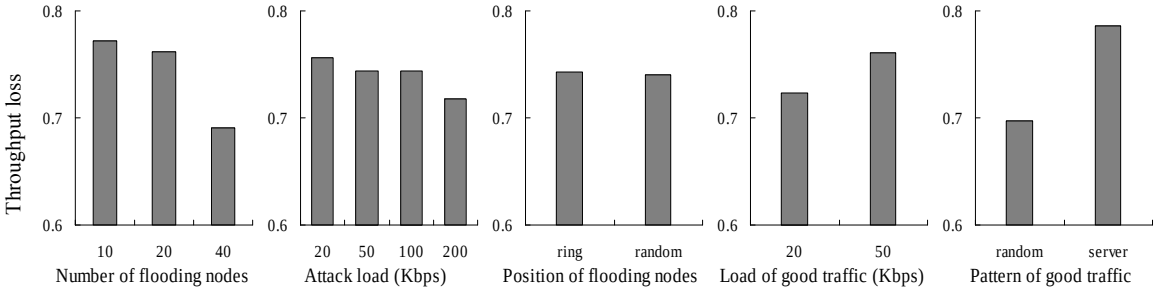


Fig. 4.8. Attack impacts under different factors in remote attacks

Among these factors, the number of flooding nodes and the pattern of good traffic have significant influences on the attack impacts[1]. The number of flooding nodes owns an effect contradictory to a normal sense that more flooding nodes leads to less throughput loss. In our simulations, the throughput loss drops from 77% (10 flooding nodes) to 69% (40 flooding nodes). This indicates that the cross congestions among flooding flows can significantly reduce the effective volume of the flooding packets in the network. Hence, the remote attack is special in our simulations in that if the attacker uses 10 flooding nodes, he has a better chance to congest the network.

The other main factor shows a normal trend. If all good nodes communicate with the service node, the good traffic suffers from higher throughput loss. Because all good traffic flows concentrate at the service node, they have serious cross congestion among themselves. This result illustrates that the good traffic itself can bring packet loss in addition to the damage caused by the flooding traffic.

Other factors show slight influence on the attack impacts[2]. Ring positioned flooding nodes may cause slightly more damage than random positioned flooding nodes. Higher load of good traffic cause higher throughput loss due to self congestions. But, higher load of flooding traffic slightly reduce the throughput loss. Consequently, in the remote attacks, the damage can be caused by a few flooding nodes with a low attack load.

Note that the difference of throughput loss under various factors is relatively small regarding the average throughput loss. In general, the high end of the throughput loss is around

[1] In ANOVA test, P-value of the number of flooding nodes is 0.005, and P-value of the pattern of good traffic is 0.000.

[2] In ANOVA test, P-value of the load of flooding traffic is 0.75, P-value of the position of flooding nodes is 0.80, and P-value of the load of good traffic is 0.05.

80%, while the low end of the throughput loss is around 70%. Hence, in a remote attack, even if the attackers can control many flooding resources, the actual attack impact may not be greatly improved. As a summary, in our simulations, 10 flooding nodes, each of them generates attack traffic at $20Kbps$, can cause the most damage in average.

4.4 Local Attacks

4.4.1 Attack Approaches

In a local attack, the attackers flood the traffic to their neighbor nodes instead of the service node. Since the flooding packets can impact all neighbor nodes, the traffic through the neighbor nodes can be affected too. One benefit of the local attacks is that the flooding nodes do not need to send the traffic over multiple hops. So the flooding nodes do not rely on other routing nodes. However, the attack is effective only if the good traffic goes through the flooding area. Greedy attackers may attack a lot of areas to take the maximum impact on the whole network instead of a single node. Furthermore, the flooding nodes have less self congestion, since the flooding traffic only goes through one hop. The flooding nodes also have less cross congestion, especially when two flooding nodes are far away from each other and cannot sense each other.

Instead of flooding, malicious nodes can misuse MAC and PHY protocols to cause congestion [2, 57]. However, all these congestion approaches require the nodes to be equipped with specialized wireless NICs or be able to modify the wireless NIC's firmware. This requirement could be unrealistic in a packet injection attack, unless the attackers can prepare and deploy many specialized nodes. Since the real attackers may not know the type of wireless NIC of

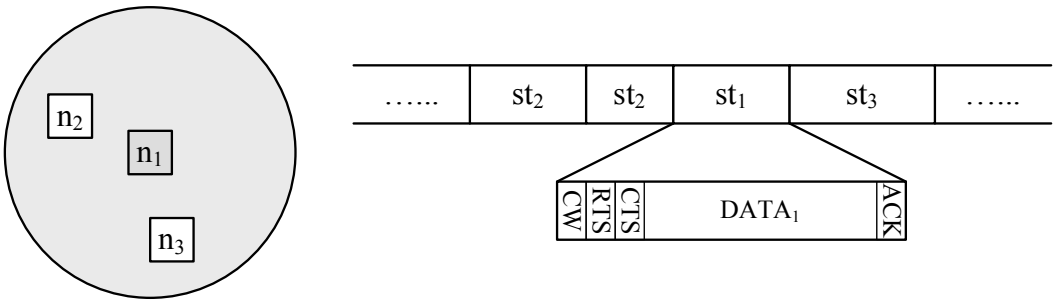
the compromised nodes and the compromised nodes may not have the authority to modify the hardware, it would be easier for the attackers to simply flood in the network.

By following the MAC protocol, a compromised node can send out packets. However, the node needs to compete the channel with other normal nodes. Hence, it is questionable whether such a local attack can be successful. As we analyzed in the following, a flooding node can still exhaust the channel. When the channel is exhausted, a normal node cannot get enough chance to send out its own packets and thus be congested. The reason that a flooding node can exhaust the channel while equally competing with other nodes is that he can compose packets with a large payload, even though he cannot change the hardware.

4.4.2 A Congested Area

To study the congestion around a flooding node in an ad hoc network, we present an analysis of a congested area based on IEEE 802.11 MAC protocol. The objective of this analysis is to demonstrate the principle on how a flooding node can congest other nodes that are equally competing in the channel. Many researches [51, 15, 6] focused on modelling the internal procedures of the MAC protocol, and studied such issues as the performance of collision avoidance, the fairness of backoff mechanism, the realized saturated network capacity. However, these are not the focus of our study. We assume that an attacker is in the center of the congested area and does not know how nodes sense the channel, avoid collision or backoff the transmission. He follows the MAC protocol as other nodes, and he just quickly generates many packets to be transmitted. We assume equality among all nodes that when some nodes have packets ready for transmission, they will have the same chance to transmit a packet in the channel. Note that there

could be short-term unfairness in the MAC protocol [2, 6]. However, in the long term, the MAC protocol is fair to all nodes that compete in this congested area.



(a) A congested area (b) A sequence of packets in transmission

Fig. 4.9. Traffic parameters

When an area is congested, all nodes in this area buffer packets in their routing queues and wait and compete for the chance next to the current transmission in the channel. Figure 4.9(a) depicts an area of three nodes and we assume node 1 is the attacker. Since the channel is congested and busy, packets are transmitted one by one in the channel. If we observe the area, we may record such a sequence of packets as depicted in Figure 4.9(b). We denote *the average transmission period* of one packet from node i as st_i, which includes the time for transmitting both a DATA packet and the MAC overheads.

In IEEE 802.11, st_i has the following components according:

$$st_i = \frac{ld_i}{R} + \tau_{RTS} + \tau_{CTS} + \tau_{ACK} + \tau_{IFS} + \tau_{CW_i(N)} \tag{4.1}$$

ld_i is the average length of a DATA packet sent by node i, and R is the transmission rate ($1Mbps$ in our study). In this study, ld_i is the sum of lengths of application data (pay load), IP header (20 bytes) and MAC header (52 bytes). τ_{RTS}, τ_{CTS}, and τ_{ACK} are the time to send RTS, CTS and ACK packets respectively. τ_{IFS} is the time for distributed coordination function interframe space (DIFS) and short interframe space (SIFS). $\tau_{CW_i(N)}$ is the average time for channel contention. By using the estimation in [16][3], $\tau_{CW_i(N)} < SLOT \times \frac{CW_{min}}{4N} < SLOT \times \frac{CW_{min}}{8} < 0.08ms$, if more than 2 nodes are competing in the channel. τ_{RTS}, τ_{CTS}, τ_{ACK} and τ_{IFS} are constant overhead, and their sum is $1.04ms$. Hence, st_i is mainly determined by ld_i, and a malicious node can compose a large packet to have a large st_i. For example, if the node compose a packet with pay load of 1500 bytes, $\frac{ld_i}{R} = \frac{(1500+20+52)\times 8}{1\times 10^6} = 12.576ms$ and st_i is thus $13.696ms$. Note that st_i is almost not affected by the number of nodes in the area, since $\tau_{CW_i(N)}$ is much smaller than other factors.

In the competition, each node can obtain a *portion of the channel*, denoted as p_i. If we observe all activities in this area for a long period t and find that node i averagely gets t_i for its transmission, then $p_i = \frac{t_i}{t}$. In this period, assume node i successfully transmits α_i packets, then $t_i = \alpha_i st_i$ and $t = \sum_{i=1}^{N} t_i = \sum_j \alpha_j st_j$. Accordingly, $p_i = \frac{\alpha_i st_i}{\sum_{j=1}^{N} \alpha_j st_j}$. In a short term, a node may have a larger α_i than other nodes, i.e. send more packets than others. However, in the long term, each node gets the equal chance to send packets. Hence, in average, α_i is the same

[3] $\tau_{CW_i(N)} \simeq SLOT \times \frac{1+Pc(N)}{2N} \times \frac{CW_{min}}{2}$, where N is the number of nodes in the area, CW_{min} is the minimum contention window size, $SLOT$ is $20\mu s$, and $Pc(N)$ is the proportion of collisions experienced for each packet successfully acknowledged at the MAC level ($0 \leq Pc(N) \leq 1$).

for all nodes in the congested area, and

$$p_i = \frac{st_i}{\sum_{j=1}^{N} st_j} \tag{4.2}$$

Assume node i inserts one packet every gt_i into its buffer for transmission[4]. If the node can get enough bandwidth, it can send out all its packets. Otherwise, if the bandwidth is exhausted, it is congested. To address the exhaustion, we define the demanded bandwidth and the allocated bandwidth of a node as following. Given st_i, gt_i and p_i for node i, its demanded bandwidth B_i^d and allocated bandwidth B_i^a are

$$B_i^d = B \times \frac{st_i}{gt_i} \tag{4.3}$$

$$B_i^a = B \times p_i = B \times \frac{st_i}{\sum_{j=1}^{N} st_j} \tag{4.4}$$

where B is the channel capacity, $1Mbps$ in this study. The congestion happens when $B_i^d > B_i^a$.

A flooding node thus can use a big st_i to congest its neighbor nodes. For example, consider a congestion situation in the area in Figure 4.9(a), where all the nodes generate one packet every $20ms$. Assume the size of the packets generated by the legitimate nodes is 485 bytes, then $st_2 = st_3 = 5ms$ according to Eq.(4.1). Hence, either one of these two legitimate nodes demands bandwidth of $250Kbps$ according to Eq.(4.3). To congest these nodes, the malicious node can deliberately compose and flood packets with pay load of 1663 bytes, then $st_1 = 15ms$. As a result, the allocated bandwidth for the legitimate nodes is $200Kbps$ according to Eq.(4.4).

[4]The packet may be generated by one of node i's applications or be a packet that node i needs to forward.

It is observed from this example that the attacker can get more bandwidth by selecting a larger st_i. If the parameter is increased from st_1 to $st_1 + \delta$, the bandwidth allocated to the legitimate node is decreased from $B_2^a = B\frac{st_2}{\sum_{i=1}^{3} st_i}$ to $B_2^a = B\frac{st_2}{\sum_{i=1}^{3} st_i + \delta}$.

This analysis illustrates that a flooding node can congest others by composing and flooding large packets. Note that, although the maximum data size in the MAC layer is sufficient for a successful bandwidth exhaustion, it is not possible that a flooding node can 100% exhaust the channel. In the MAC layer, the maximum packet size can reach roughly $32K$ bits (st is about $32ms$). One problem for attackers is that a compromised node may not be able to make full use of the maximum packet size, especially when he uses socket programming to compose such a large packet. In the TCP/IP protocols, a maximum transmission unit (MTU) is set for any UDP and TCP packet. In general, the MTU is 1500 bytes, which means a compromised node may have st_i up to around $13.7ms$. In the later simulations, we will show that this is still enough for a flooding node to disrupt the network.

This analysis is still limited to show the real attack impacts. When a good node is suppressed by a flooding node and unable to get sufficient bandwidth, it not only has to defer the transmission of its packets, but also has limited time to accept packets from other nodes. Other nodes may think the node malfunctions and the link to this node may be conceived failure. This will further trigger other nodes to break routes going through this node or drop packets toward this node. We will use simulations to study the complicated attack impacts.

4.4.3 Attack Constraints

In a local attack, a flooding node only has the direct impact on its vicinity. Hence, the questions for a local attack are how the flooding nodes may be deployed and how serious the

attack is. For analysis purpose, we first observe the channel at a location x for a period T. During this period, it takes $t_{tr}(x)$ for transmission in the channel. Of $t_{tr}(x)$, $t_{norm}(x)$ is allocated for normal traffic. Then, define normal traffic density at location x, $D_{norm}(x) = \frac{\frac{t_{norm}(x)}{T}}{\int_S \frac{t_{tr}(x)}{T} dx}$, where S means the whole network.

The damage of a local attack can be measured as $M = 1 - \int_S D_{norm}(x)dx$. Because $t_{norm}(x) \leq t_{tr}(x)$, $\int_S D_{norm}(x)dx \leq 1$. If there is no attack, $t_{norm}(x) = t_{tr}(x)$ and thus $\int_S D_{norm}(x)dx = 1$ and $M = 0$, i.e. damage is zero. If normal transmission is totally disabled, $t_{norm}(x) = 0$ and $t_{tr}(x)$ is for the attack traffic only. In this case, $\int_S D_{norm}(x)dx = 0$ and $M = 1$, i.e. the network is 100% damaged.

It is very complicated to measure $t_{tr}(x)$ and $t_{norm}(x)$ under attack, because (a) routes are highly dynamic under attack due to link failure, (b) the network traffic may be re-distributed due to route changes, and (c) the effect of the attack traffic on the normal traffic is determined by their interaction and thus it is uncertain due to the first two reasons. However, it is possible to study some properties with simplified models. Assume N compromised nodes can disable their vicinities 100% once they start attack, then the damage (before the normal traffic is re-distributed) is

$$M = 1 - \int_S D_{norm}(x)(1 - d(x))dx \tag{4.5}$$

where $d(x)$ is a damage ratio, and $0 \leq d(x) \leq 1$. At location x, $d(x) = 1$ if x is inside the attack area of any attack host; otherwise, $d(x) = 0$, i.e. no damage to this location.

Eq.(4.5) gives a two-fold meaning. First, given a traffic topology and an attack deployment, attackers can estimate the damage. For example, assume N attack nodes are randomly distributed in the network, we can derive the average damage as $E(M) = 1 - \int_S D_{norm}(x)(1 -$

$E(d(x)))dx$. Second, attackers may select an optimal $d(x)$ and a good attack strategy to maximize the total damage M. Assume that normal traffic is uniformly distributed in an ad hoc network, i.e. $D_{norm}(x) = \frac{1}{S}$. If attackers can congest area s, it is not difficult to prove that the damage is $M = \frac{s}{S}$, which indicates that the damage is proportional to the congested area in a network with uniformly distributed traffic.

Hence, the good attack strategy is to deploy as many attack hosts as possible and assign each attack host to a non-overlapped area. Apparently, the optimal $d(x)$ depends on $D_{norm}(x)$. In reality, the interaction between attack traffic and normal traffic cannot be a simple zero-one relationship as in Eq.(4.5). Complex traffic distribution results in complicated attack strategies. We use the following simulations to study these issues.

4.4.4 Simulations

In this section, we present details of our simulations to study the characteristics of the local attacks. We still focus on the attack impacts under the same factors as in Section 4.3.3. Furthermore, we will compare the local attacks with the remote attacks to identify their unique features.

We use the same communication model, network topology, traffic model and default traffic setting as in Section 4.3.3. However, in the simulations, every flooding node only sends packets to one of its neighbors, which is randomly selected from all its neighbor nodes. We also measure the same metrics as in Section 4.3.3 to evaluate the attack impacts.

Each flooding node generates one packet every $100ms$. By composing the packets with different payloads, they can generate different attack loads. Table 4.1 lists the relations between the attack loads, the payloads and the transmission periods defined in Eq.(4.1). For example,

if the attack load is $20Kbps$, the length of each DATA packet in the MAC layer is 322 bytes (including IP header and MAC header), and thus the average transmission period of this attack load is $3.696ms$ according to Eq.(4.1).

Table 4.1. Attack loads and corresponding parameters

Attack load (Kbps)	20	50	100	200
Average length of pay load (bytes)	250	625	1250	2500
Average transmission period (ms)	3.7	6.7	11.7	21.7

4.4.4.1 Positions of the Flooding Nodes: random or ring

In the local attacks, the attack impacts are mainly determined by how many areas are flooded by the attackers. If all flooding nodes gather in a small area, the flooding traffic may not be able to affect traffic in other areas. The attackers can still put all flooding nodes in the ring as in the remote attacks. Although the ring may actually circumvent the service node, the flooding traffic can only interfere the good traffic going through the ring and thus the attack impacts may be reduced. Hence, it may not be better than the random positioned flooding nodes.

Figures 4.10(a), (b) and (c) illustrate the attack impacts of these two positions of flooding nodes. In each figure, the solid lines stand for the throughput loss of the good traffic when the flooding nodes are randomly distributed in the network, and the dashed lines for the ring type positions.

Different positions of flooding nodes have different influences on the attack impacts. In general, when the flooding nodes are randomly positioned, the throughput loss of good traffic

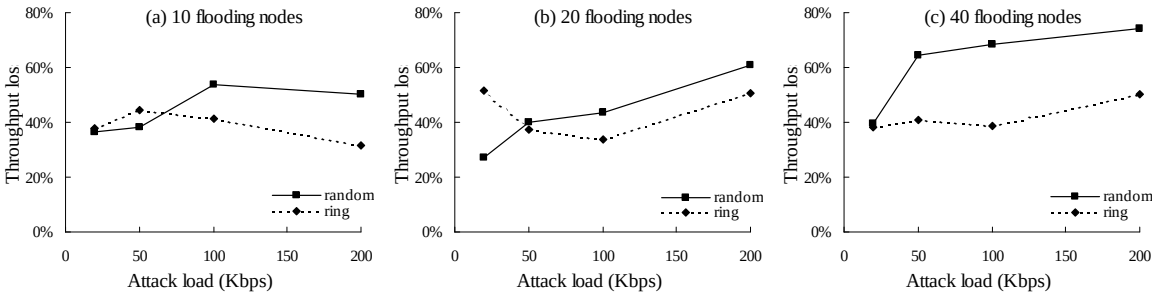


Fig. 4.10. Positions of the flooding nodes: random or ring

grows as the attack load increases, and the loss is also increased as the number of flooding nodes increases. In Figure 4.10(a), the throughput loss grows from 39% to 50%. In Figure 4.10(b), the throughput loss grows from 26% to 60%. In Figure 4.10(c), the throughput loss grows from 40% to 72%. This justifies the analysis that the attack impact grows as the number of the flooding nodes and the attack load increase due to the fact that more flooding nodes can interfere more areas in the random position.

When the flooding nodes are positioned in the ring, the throughput loss of good traffic shows a different pattern. The throughput loss is almost not changed regarding both the number of the flooding nodes and the attack loads. In Figure 4.10(a), the throughput loss grows slightly as the attack load increases; while in Figure 4.10(c), the throughput loss drops slightly as the attack load increases. In addition, the throughput loss is in a small range around 40% no matter how many flooding nodes are in the network. Since the flooding nodes gather in the ring area, they can only affect the traffic going through the ring. The chance that the good traffic goes through the ring is almost independent of the attack loads and the number of flooding nodes, the throughput loss is thus flat in average.

Consequently, in the local attacks, the attackers may try to put the flooding nodes uniformly in the network so that the flooding nodes do not gather in a small area. However, if the attackers want to damage a certain area, they may try to put the flooding nodes in this area.

4.4.4.2 Good Traffic Patterns: with the service node or random

Figures 4.11(a), (b) and (c) illustrate the attack impacts on the two types of good traffic. In each figure, the solid lines stand for the throughput loss of the good traffic that connects with the service nodes, and the dashed lines for the traffic between two randomly selected nodes.

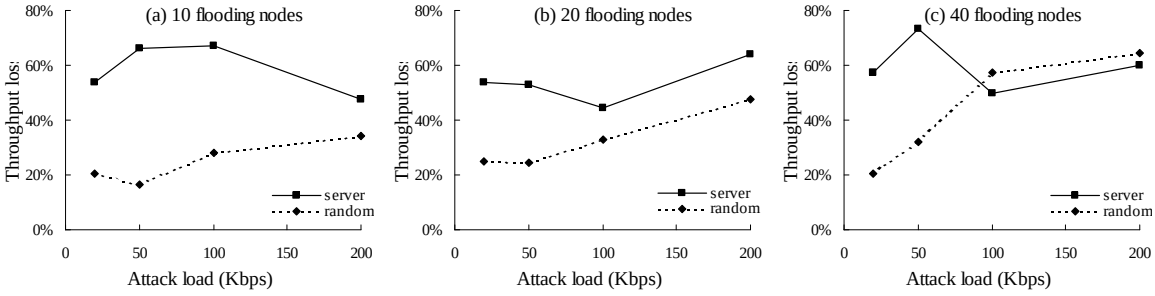


Fig. 4.11. Good traffic patterns: with the service node or random

It is illustrated that when good nodes communicate with the service node, the flooding traffic has only slight influence on the attack impacts. The average throughput loss of good traffic is in a range around 60% regarding different numbers of flooding nodes and different attack loads. Since in this situation, the good traffic aggregates in the vicinity of the service node, the good traffic flows suffer from the cross congestion among themselves. The flooding nodes only bring additional damage to the good traffic.

On the contrary, the random good traffic has less cross congestion, and is thus more affected by the flooding traffic. Figure 4.11(a) shows that the throughput loss of random good traffic grows from 20% to 34% as the attack load increases. In Figure 4.11(b) the throughput loss grows from 22% to 42% and in Figure 4.11(c) the throughput loss grows from 20% to 61%. However, the throughput loss of random good traffic is generally less than that of the good traffic connecting with the service node. In the simulations, if the attack load is low at $20Kbps$, the throughput loss of random good traffic is only around 20%. The chance that the random good traffic is affected by the flooding traffic is also influenced by the number of flooding nodes. Especially when the attack load is high at $200Kbps$, the high end of the throughput loss of random good traffic grows as the number of flooding nodes increases. In Figure 4.11(a), the high end of the throughput loss of random good traffic is only 34%. While in Figure 4.11(c), the high end of the throughput loss of random good traffic reaches 61%.

The throughput loss of random good traffic shows the attack impacts without good traffic's own congestion. In the local attacks, the throughput loss grows as the attack load increases, which is different from the remote attacks. In general, the throughput loss in local attack is less than the remote attacks. However, we can find that in both remote attack and local attack, the gap between two curves in each figure decreases as the number of flooding nodes increases. When the network is full of flooding nodes, no matter whether flooding traffic goes through one or multiple hops and whether good traffic is random or connects with the service node, most areas of the network are congested, and the throughput loss in both remote and local attacks is close to 60%.

4.4.4.3 Good Traffic Loads: $20Kbps$ or $50Kbps$

Figures 4.12(a), (b) and (c) illustrate the attack impacts on two loads of the good traffic. In each figure, the solid lines stand for the impacts on the good traffic at $20Kbps$ per flow, and the dashed lines for the good traffic at $50Kbps$ per flow.

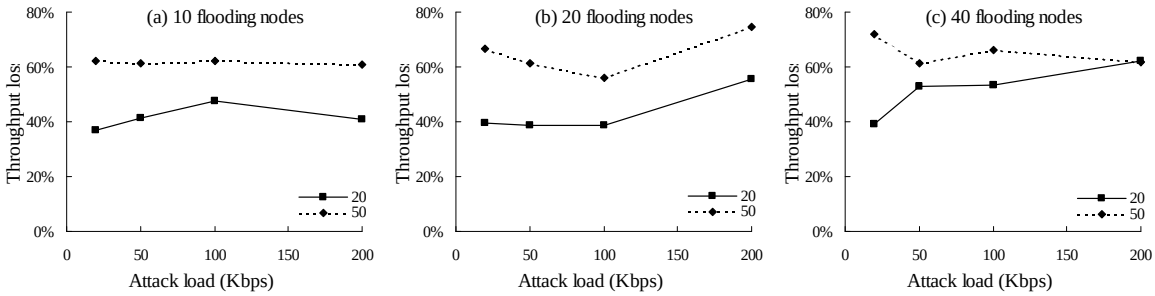


Fig. 4.12. Good traffic loads: $20Kbps$ or $50Kbps$

The results show that the good traffic with a higher load has larger throughput loss. First, consider the good traffic of $50Kbps$. If 10 flooding nodes are in the network, the throughput loss is flat around 60% as the attack load increases. This again shows the influence of the area the flooding nodes occupy. When the number of flooding nodes is small, the chance that good traffic goes through flooded area is not high. Consequently even if the attack load is high, the throughput loss does not increase. In addition, because the good traffic uses a larger packet to make a higher load, it is possible the good traffic can better compete with flooding nodes in the channel. As shown in Figure 4.12(c), when the number of flooding nodes is 40, the throughput loss drops from 72% to 60% as the attack load increases.

The throughput loss of good traffic at $20Kbps$ shows different trends. When the number of flooding nodes is small, the throughput loss is flat around 42%. The reason is similar to the analysis above. But Figure 4.12(b) shows that when the number of flooding nodes is 20, the throughput loss of good traffic grows from 40% to 56% as the attack load increases. In Figure 4.12(c), the throughput loss can grow to 60%. This again demonstrates more flooding nodes can affect more good traffic. Furthermore, the gap between the curves for good traffic of two loads is slightly reduced when more flooding nodes are put in the network.

4.4.4.4 Flooding Flows: UDP or TCP

Because the flooding traffic only goes through one hop in the local attack, the self congestion and the cross congestion may not be important to the flooding traffic in the local attacks. Figures 4.13(a), (b) and (c) depict the attack impacts of two types of flooding traffic, UDP and TCP. In each figure, the solid lines stand for the throughput loss of the good traffic when the flooding nodes flood UDP traffic at different attack loads, and the dashed lines for TCP flooding traffic.

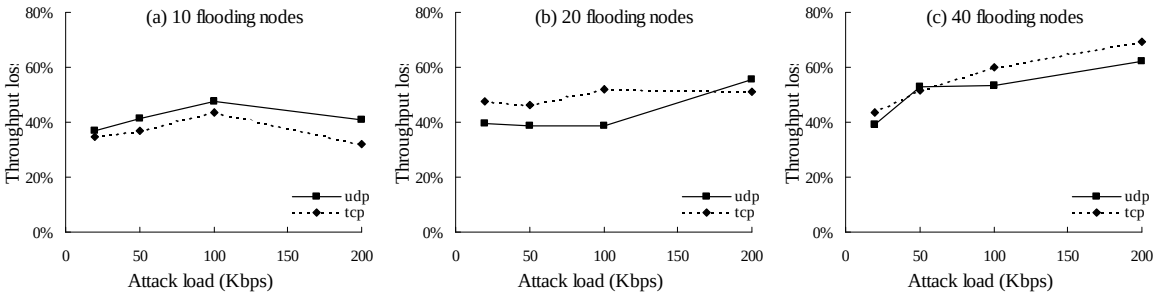


Fig. 4.13. Flooding flows: UDP or TCP

Compared with Figure 4.7, it is very different from the remote attacks that the TCP flooding traffic in the local attacks may bring more damage to the network with a slightly higher throughput loss of good traffic than the UDP flooding traffic in some situations. When the number of flooding nodes is small as in Figure 4.13(a), TCP congestion control mechanism limits the throughput of the flooding traffic when it collides with the good traffic. But when the number is large as in Figure 4.13(b) and (c), TCP congestion control mechanism helps to reduce the cross congestions among nearby flooding nodes, and thus increases the throughput of flooding traffic, which in turn suppresses the good traffic.

Because the flooding traffic only goes one hop, both types of the flooding traffic show the similar trends regarding the number of flooding nodes and the attack loads, which is another property different from what we observed in the remote attacks. Figure 4.13(a) shows that both throughput losses are around 41%. When the number of flooding nodes is large, the curves of throughput loss show a growing trend as the attack load increases. In Figure 4.13(b) the throughput losses grow from 42% to 51%, and in Figure 4.13(c) they grow from 41% to 60%.

Furthermore, the difference between the curves for the throughput loss caused by these two types of flooding traffic is not significant. Hence, the UDP flooding traffic still seems to be a good choice for the attackers in the local attacks.

4.4.4.5 Summary

In previous comparisons, we studied the interactions among different attack factors. In this part, we give an overall evaluation on the main effects of these factors, which shows the great differences of the local attacks from the remote attacks. Figure 4.14 shows the main effects of the six factors on the attack impacts.

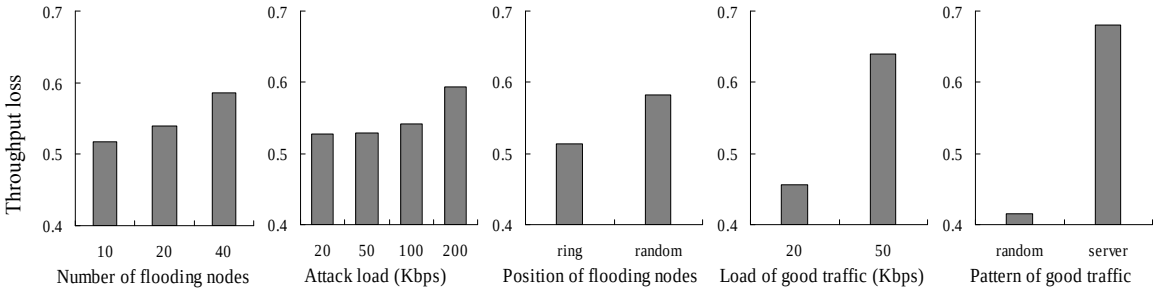


Fig. 4.14. Attack impacts under different factors in local attacks

In local attacks, the position of flooding nodes, the pattern of good traffic, and the load of good traffic are the main factors that can significantly influence the attack impacts[5]. Note that in both remote attack and local attack, the pattern of good traffic is a main factor. This indicates that an ad hoc network is vulnerable to all kinds of traffic. If the network is full of good traffic, the result will be similar to a packet injection attack. In the viewpoint of the attackers, a good packet injection attack strategy is to make use of the good traffic. The attackers only need to deploy the flooding nodes in the area where good traffic is not intense.

Different from the remote attack, the position of flooding nodes is a main factor in a local attack. In a remote attack, since flooding traffic goes through multiple hops, the position of the flooding nodes has less influence on where the traffic can go. In a local attack, one hop flooding traffic can only affect the nearby traffic. Hence, the attackers may try to deploy the flooding nodes uniformly in the network, if they can control the position of flooding nodes.

The load of good traffic in the local attack is also a main factor. This indicates that the ability of a node to compete the channel in the local attack is an important factor that determines

[5]In ANOVA test, P-value of the position of flooding nodes is 0.013, P-value of the pattern of good traffic is 0.000, and P-value of the load of good traffic is 0.000.

how much portion of the channel the node can obtain in a congestion situation. In the remote attack, the importance of this ability is reduced due to other problems in multi-hop transmission, such as exposed node problem and link failure [12].

Other factors show slight significant influence on the attack impacts[6]. But they also exhibit some properties different from the remote attacks. First, in local attacks, the attack impacts are increased with the increase of the number of flooding nodes. Since the flooding traffic in the local attacks suffers less from self and cross congestions, more flooding nodes obviously can bring more damage to the network. The second difference is that higher attack load in the local attacks can cause more damage to the network. In a local attack, the most damage is caused when 40 flooding nodes are deployed in the network and each node floods at the highest rate. Finally, in average, the throughput loss in the local attacks (0.55 ± 0.23) is less than that in the remote attacks (0.74 ± 0.15). Note that, when the network is crowded with flooding nodes, the gap of the throughput losses can be reduced that both types of attacks have the similar attack impacts.

4.5 Conclusion

This study shows that packet injection attacks are a serious threat to an ad hoc network. In this study, we mainly studied the attack impacts of two types of packet injection attacks, and compared important factors that influence the attack impacts. Based on the simulations, we find that the remote attack is more effective and efficient for packet injection attackers to damage the network. It is noted that more flooding nodes and higher attack load cannot increase or

[6] In ANOVA test, P-value of the number of flooding nodes is 0.11, and P-value of the load of flooding traffic is 0.27.

even reduce the attack impacts in a remote attack. On the contrary, the local attack needs more resource than the remote attack. The damage in a local attack grows if more flooding nodes send traffic at higher attack load in the network. In the simulations, the good traffic also has attack impacts on itself, and the packet injection attacks bring additional damage to the network.

Chapter 5

Defense Against Packet Injection in Ad Hoc Networks

5.1 Introduction

Denial-of-Service (DoS) attacks are a serious threat to ad hoc networks where the network resources are typically very limited. Since the wireless channel is a shared media, injecting a large number of junk data packets into a set of legitimate routes may not only cause serious network congestion on the destination side, but may also lead to severe wireless channel contention along each of the legitimate routes. Hence, in this study, we are mainly concerned with this type of junk packet injection attack. Compared with other types of DoS attacks in ad hoc networks, the injection attack generally is easier to enforce, since they do not need any special support on the underlying routing or MAC protocol, but may be more difficult to defend.

Ad hoc network security has been extensively studied recently. However, most of the previous work [18, 20, 62] focus on secure routing. After a route is established, there is no authentication in forwarding data packets. As a result, an attacker can inject a large number of junk data packets into the route with the goal of depleting the resources of the nodes that relay or receive the packets. To mitigate this attack, an en route node needs to filter out the injected junk data packets as early as possible instead of leaving it for the destination to detect. The longer a junk data packet stays in the network, the more resources it will consume.

One solution is to enforce source authentication in forwarding data packets. When the source sends a data packet after discovering a route, it puts authentication information into the

data packet. An en route node only forwards the data packet if it is authenticated. In this way, only the data packets from the real source can go through the route and reach the destination. Public key based source authentication is not considered in this study, because of its unbearable computational demand on mobile nodes. In addition, public key based signature could be replayed during its lifetime and thus be exploited by attackers to inject the replicated data packets in different areas of an ad hoc network. The authentication of a forwarded data packet is also different from the authentication of a routing packet [42], which relies on TESLA. The authentication protocol needs a forwarding node to buffer the packet and then verify it later. However, in order to limit the impacts of injected packets, a good forwarding node should be able to verify a received packet before forwarding it to the next hop. Hence, hop-by-hop source authentication [60, 65] has been considered as the necessary measure to ensure that an injected false date packet can be filtered out immediately. If some nodes do not verify the source in forwarding data packets, an attacker may inject packets into the network through these nodes.

There are many practical challenges in applying source authentication approaches in an ad hoc network among which the most critical is the unreliability of the network. For example, packet loss and route change make it impossible for en route nodes to verify the source even if the source is not an attacker. Because of this, even good packets will be discarded when these approaches are adopted for defense purposes. To address these problems, we propose a lightweight, on-demand and hop-by-hop source authentication protocol in forwarding data packets. Since our emphasis is on handling the unreliability instead of proposing another group authentication method, a straightforward approach is used as the baseline in our protocol. First, the source secretly shares a pairwise key with each en route node according to [9, 27, 66]. Then, the source computes an authentication token for each en route node with the key shared between

them, when it sends a data packet. Thus, the data packet can be verified hop by hop. This approach can provide immediate source authentication and inherently supports the on-demand nature of ad hoc networks. In this study, we will illustrate how to use this protocol to handle the unreliability.

The rest of the chapter is organized as follows. Section 5.2 presents background information regarding attack and defense approaches. In Section 5.3, we present various problems that an authentication protocol will face. Section 5.4 introduces the design overview, the detail of SAF, and its security analysis. We evaluate SAF in Section 5.6 and conclude this chapter in Section 5.7.

5.2 Background

5.2.1 Attack Scenarios

A node launches resource consumption attacks because it has been compromised or it intentionally does it; we do not distinguish the attack motivation here. The attacker may use its own ID, a fabricated ID, or another node ID as the source of the packets it is injecting. We assume, however, that attackers will impersonate other non-compromised nodes to hide themselves, because it is risky for an attacker to misbehave in its own name. In the following, we show the major attack scenarios in Figures 5.1-5.4 to demonstrate the attack principle and the collaboration among attackers. For discussion, we need three addresses to represent nodes: attacker's address (A_a), the impersonated address (A_b), another spoofed address (A_c), and the target address (A_d).

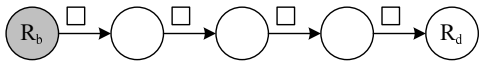


Fig. 5.1. Packet injection scenario 1

Scenario 1 In this scenario, an attacker establishes a route with the source address as A_a or A_c and follows the routing protocol to ensure a good route. If the routing protocol is secure, the attacker may have to use A_a, unless it has compromised node A_c and obtained its keys for routing authentication. After a route is set, the attacker sends a packet with the source as A_b and the destination as the target A_d. It can claim the packet is forwarded from A_b. When an intermediate node receives the injected packet, it checks its routing table, finds the forwarding entry for A_d, and then forwards the packet, although the route is actually from A_a to A_d. When the target detects A_b, it cannot figure out where the packet comes from since no regular node has a forwarding entry from A_b to A_d or the entry points back to A_b instead of A_a.

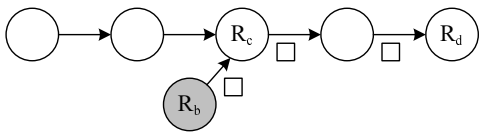


Fig. 5.2. Packet injection scenario 2

Scenario 2 In this scenario, an attacker does not establish a route to A_d. Instead, it listens to the nearby traffic until it finds that node A_c has a route to A_d. Maybe node A_c wants to

communicate with A_d, or A_c is occasionally in a route from another node to the target. Then, the attacker sends a packet to node A_c with the source as A_b and the destination as A_d. Similarly to scenario 1, intermediate nodes forward the packet and lose its origin.

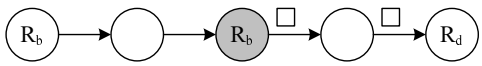


Fig. 5.3. Packet injection scenario 3

Scenario 3 In this scenario, an attacker in a route from A_b to A_d, and thus it does not establish a route to A_d. The attacker sends a packet with the source as A_b and the destination as A_d as if it was A_b. It can claim the packet is forwarded from A_b. Similar to the first two scenarios, intermediate nodes forward the packet. What is different is that the target can trace the packet back along the route if it detects its maliciousness. Unfortunately, the trace goes back to node A_b instead of the attacker A_a in the middle.

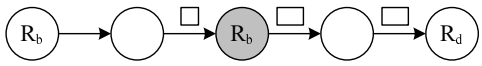


Fig. 5.4. Packet injection scenario 4

Scenario 4 In this scenario, an attacker does not establish a route to A_d as in scenario 3, nor does the attacker actively inject junk packets into the route from A_b to A_d. Instead, upon receiving a packet A_b, the attacker appends junk bits to enlarge the packet so that the packet can consume more bandwidth and transmission power as it is forwarded along the route. In this way, the attacker does not need to spoof A_b, and certainly the trace goes back to node A_b.

5.2.2 Defense Approaches

Because the attackers are likely to impersonate other nodes and flood packets with spoofed source addresses in order to hide themselves, source authentication is the key technique to prevent these attacks. Once security is introduced into forwarding, each en route node will hold certain credentials for defense purpose, including the attackers and the legitimate router. Hence, the attacker could be an *outsider* (unauthorized) node that does not possess a valid credential, or an *insider* (authorized) node that possesses a valid credential. In particular, regarding a specific route, its insiders are the routing nodes which are supposed to have the credential to forward data packets, and its outsiders are the nodes which are not. Note that insiders and outsiders are defined in a relative way; an insider of one route could be an outsider of another route.

Recently two techniques [60, 65] have been proposed for filtering injected false data packets in sensor networks. Ye *et al.* [60] proposed a statistical filtering scheme that allows en route nodes to filter out false data packets with some probability. Zhu *et al.* [65] proposed an interleaved hop-by-hop authentication scheme that guarantees that false data packets will be detected and dropped within a certain number of hops.

In general, the defense process consists of three steps. First, the source and the destination need to establish a route, which can be secured by [18]. Then, the source node needs to

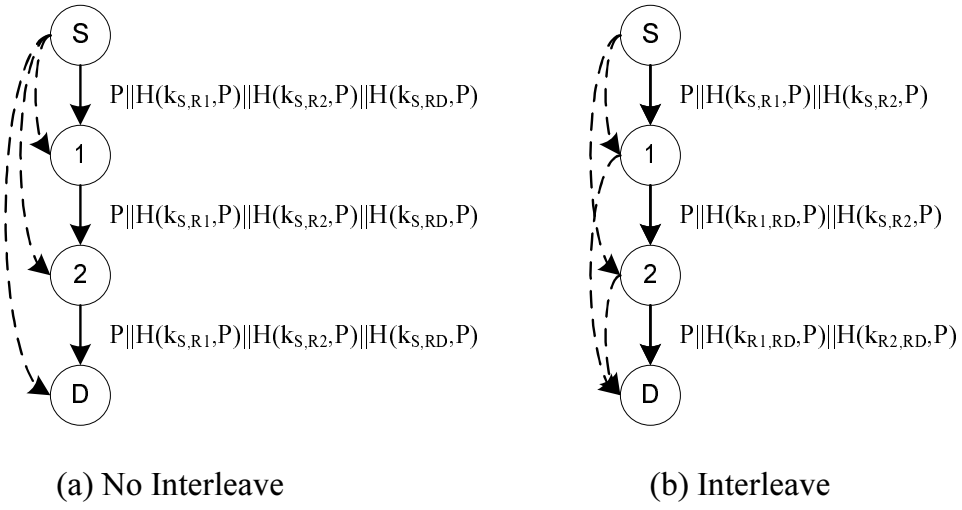


Fig. 5.5. Forwarding in a new route

set up pairwise keys with the en route nodes. Finally, the source computes the authentication header as depicted in Figure 5.5, in which a route has three hops from source S to destination D. The solid lines show the route, and the dashed lines represent how each node sets up keys. In Figure 5.5(a), the source sets up one key with each en route node, and no key is set up between any two en route nodes. In Figure 5.5(b), all nodes have an interleave association. For example, node 1 sets up keys with S and D. When S sends a packet, it simply computes an authentication header with message authentication codes for each en route nodes so that every node can verify the source.

These approaches are good in sensor networks, which are more reliable than ad hoc networks. However, these approaches are not sufficient for the attacks we examined in this study, In [60], a node verifies a token if the token is designated to it. Otherwise, the node will pass the authentication header with a belief that the following forwarding nodes will verify it. Hence, it

is possible that a junk packet can go through the network (although it will be discarded at the destination) if it does not carry keys that the en route nodes have. The scheme in [65] cannot be applied in our study either, because the interleave relationship cannot be sustained when the route changes. The authentication header is computed based on the interleave relationship, and only the nodes in this relationship can verify the header. When a route is changed to a new one, a node in the new route does not have the relationship with the previous forwarding node and thus will discard packets.

5.3 Problem Statements

When applying source authentication in an ad hoc network, the unreliable mobile environment puts many limits on defense approaches. The corresponding problems can even be exploited by injectors to launch attacks and hide their identities.

5.3.1 Packet Loss

A packet could be lost due to communication error, hardware error, buffer overflow, etc. In a TCP session, this will trigger the source to retransmit the lost packets. However, retransmission allows attackers to legally replay packets. When attackers replay packets, these packets will be verified successfully by other en route nodes, since the replayed packets are authentic and attackers can claim they are just retransmitting these packets. Furthermore, in some authentication approaches (for example, multicast authentication [60, 40]), authentication headers can be verified by all nodes in the network (for data integrity purposes). The attackers could thus replay these packets in other areas in the network instead of in the target area or routes.

5.3.2 Route Change

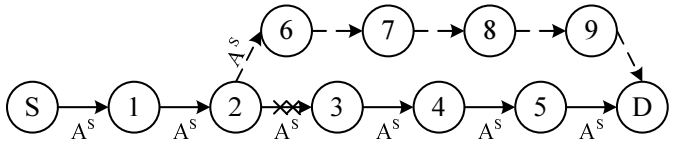


Fig. 5.6. The change of a route

In an ad hoc network, a new route may be set up for a variety of reasons. For example, the routing protocol itself enables an en route node to overhear routing messages and discover shorter routes, or the route can be broken due to link failures or the leaving of an en route node. If the new route diverges from the previous one at the source node, the source node simply bootstraps a new forwarding procedure as in Section 5.4.2. However, if the new route diverges from the previous one at an en route node, i.e. the beginning segment of the new route overlaps with the previous one, the resulting situation is very complicated and deserves more in-depth analysis.

Figure 5.6 depicts an example where the old route (solid lines) between S and D is broken at the link between nodes 2 and 3. Since node 2 knows another route (dashed lines) that can reach D, the new route diverges from the previous one at node 2. Note that nodes 3, 4 and 5 can still use the old route to forward packets, since the old route is still valid at their positions and their buffered packets have valid authentication headers.

Because S may not know the new route immediately when the old route is broken, nodes 6 to 9 will not have any pairwise key with S before S starts a new forwarding procedure in the new route. In addition, some data packets may be already buffered in nodes 1 and 2 for forwarding, and S cannot modify the authentication headers in these packets. Hence, these buffered packets may be filtered even after S computes new authentication headers in the new route.

5.4 Proposed Approaches

5.4.1 Assumptions

5.4.1.1 Network and Communication

In this study, we mainly prevent attacks in unicast communication. We assume that in the wireless communication, a failure link can trigger a node to re-discover a route. These assumptions hold in IEEE 802.11 protocol [22] and ad hoc routing protocols [23]. SAF is designed to work with the routing protocol DSR [23], since it needs the IDs (i.e. the node's address) of en route nodes along the forwarding path. Other protocols, such as AODV [39], cannot be applied with our protocol directly, unless these protocols are extended to carry the IDs of en route nodes in the routing protocol packets. In addition, we consider a complex environment in ad hoc networks. For example, a packet could be lost due to transmission error, and a route could be broken and changed due to a routing node powering down. SAF is designed to fit such an unpredictable and unfriendly environment.

5.4.1.2 Security

We notice that an attacker can launch DoS attacks at the physical layer or the data link layer. For example, it can jam the radio channels or interrupt the medium access control protocol. However, this study does not address these types of attacks nor does it address attacks against routing packets, which have been addressed by secure routing protocols [18, 20, 62].

We mainly consider the junk packet injection attacks in which an attacker injects junk packets into an ad hoc network with the goal of depleting the resources of the nodes that relay or receive the packets. We believe that attackers will impersonate other non-compromised nodes to hide themselves, because an attacker takes a great risk when misbehaving in its own name. To achieve the attack goal, an attacker may eavesdrop on all traffic, replay older packets, or modify overheard packets and re-inject them into network. Multiple attackers may collude to launch attacks.

Note that in this study, an en route node does not verify whether the content of a data packet is maliciously modified, because modification of content will not cause congestion or require more transmission power in ad hoc networks. An en route node only makes sure that the data packet is originated from the claimed source by verifying authentication information. The authentication of the packet content can be accomplished with any end-to-end authentication protocol.

5.4.1.3 Pairwise Keys Establishment

Hop-by-hop source authentication requires that the source node sets up a pairwise key with every en route node along the path. Because the source node can obtain IDs of routing nodes from DSR route reply packets, the source node and any one of the routing nodes can

mutually figure out a pairwise key based on their IDs. Note that two en route nodes do not need to have a pairwise key.

The simplest way to set up pairwise key is to pre-load pairwise keys into nodes, although the memory requirement is not tolerable when the network is large. The literature provides many novel key management schemes with better performance. For example, in random key schemes [10, 9, 27, 66], any two nodes can establish a pairwise key with a sufficiently high probability, and only $O(n)$ memory is needed. Recently, [8] proposed a non-probabilistic scheme with a memory requirement of $O(\sqrt{n})$. In addition, Zhang *et al.* [63] provided a rekeying scheme based on the collaboration among neighboring nodes to counteract compromised nodes in filtering false data.

In this study, the proposed hop-by-hop source authentication protocol is based on the existing works for key setup and management as long as they can ensure the security of the pairwise keys. This protocol focuses on solving the unreliability problems in the following forwarding procedures at discussed in Section 5.3.

5.4.2 Framework

We first give the simple version of the proposed protocol in a reliable ad hoc network, where the route from a source to a destination will not break and no packet will be lost during the transmission. Then, in Section 5.4.3, we present the complete version of the protocol to handle problems in a realistic unreliable ad hoc network.

Every node in the ad hoc network enforces the proposed protocol as shown in Figure 5.7, where the left module represents a regular or secure routing protocol, and the right module is our scheme for forwarding. The forwarding module, like the routing module, is an independent

module in the network layer and decides if a data packet[1] should be forwarded or not. More specifically, the forwarding module consists of three subcomponents: *forwarding entry creation*, *forwarding bootstrap*, and *forwarding verification and update*.

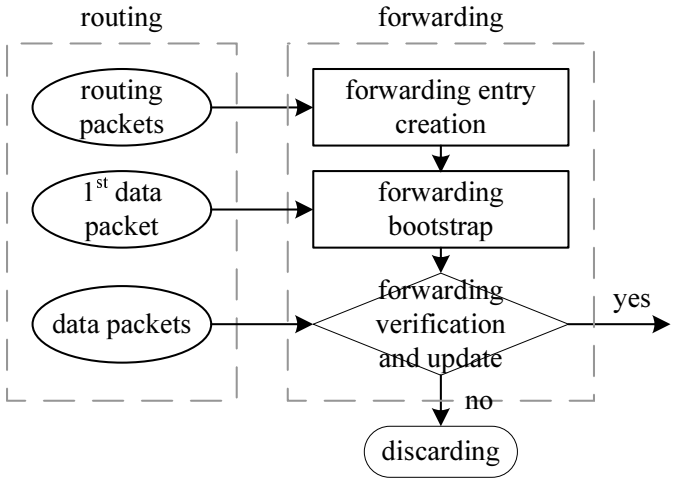


Fig. 5.7. Framework of SAF

For discussion, we assume that a source node S sends packets to a destination node D through a route of $n-1$ routing nodes, which are ordered as $R_1, ..., R_j, ..., R_{n-1}$, and R_n is D.

5.4.2.1 Entry Creation

When S wants to send data packets to D, it uses the routing protocol to find a route. First, the routing packets trigger a node to create a routing entry for the route. Our protocol

[1]Data packets refer to the packets in the network layer, but exclude routing packets (for routing) and keying packets (for pairwise key establishment). The excluded types of packets are generally secured by their own protocols [18, 20, 62, 9, 27, 66], which can prevent attackers from exploiting these packets.

requires that the source node knows the IDs of the en route nodes and an en route node knows the IDs of the downstream nodes. This information is readily available in the ROUTE REPLY packet of the routing protocol DSR [23]. Based on IDs, S and each en route node R_j can set up a pairwise key k_{SID,R_j} according to [9, 27, 66]. Accordingly, a routing entry records such basic information as the source address S, the destination address D, and a unique identification RID. In DSR, RID is the routing sequence number which is used by the source to discover a route. If a node keeps RID together with the route in its routing cache, the node can identify the route later when it receives a packet with S, D and RID. After the route is discovered, the source creates a forwarding entry, which is an extension of the routing entry. As discussed later, a forwarding entry will include additional information for forwarding verification, such as:

- SID: identification of source/starter,
- FID: identification of forwarding entry,
- PC_{1st}: the first received packet,
- PC_{last}: the last received packet.

5.4.2.2 Bootstrap

Upon the setup of a route, the source node sends its first data packet $PKT(1)$. Its forwarding bootstrap component attaches the initial authentication information $A(1)$ to the packet. An en route node receiving this packet records this initial information, verifies the source node and extracts some secrets that are only shared among the en route node and the source node.

$$A(1) = [SID||RID||FID||PC(1)||\delta_{R_1}(1)||...||\delta_{R_n}(1)]$$

Where SID is the source ID, RID is the identification of the routing entry, FID is the identification of the forwarding entry, and $PC(1)$ is the count of the first packet. Upon receiving a packet, a node can find the corresponding routing entry according to S, D and RID. FID is used to help en route nodes know whether SID selects a new forwarding entry to forward packets. As we will show later, the same routing entry may have multiple different forwarding entries. FID is used to distinguish different forwarding entries which are extension of the same routing entry. Hence, even though the source node may have multiple routes to the same destination due to routing changes, FID is necessary to help the source node to distinguish a forwarding entry from any other entry for the same destination. Nevertheless, for two different forwarding entries that are extension of two different routing entries, FID can be the same.

$\delta_{R_j}(1)$ is the authentication token for R_j. The size of an authentication token is determined by the tradeoff between security and performance. For discussion, we set a token as an 8-bit number in this study, although the hash output could be 256 bits or longer.

$$\delta_{R_j}(1) = H_{k_{SID,R_j}}(RID||FID||PC(1)||L_j)$$

Where k_{SID,R_j} is the pairwise key shared only between SID and R_j, and $H_k(*)$ is a keyed hash function. L_j is the sum of the data size, the number of authentication headers, and the number of remaining authentication tokens in the authentication header when the packet arrives at R_j. For example, in Figure 5.8, when R_2 receives a packet, the packet should include 1 authentication header, and the header has 2 tokens ($\delta_{R_2}(1)$ and $\delta_{R_3}(1)$). Hence, assume the packet has 100-byte data, then $L_2 = 100 + 1 + 2 = 103$. Similarly, when R_3 receives the packet, R_3 should have $L_3 = 100 + 1 + 1 = 102$.

S appends $A(1)$ to the first data packet (thus the whole packet is called *bootstrap packet*), and sends it to the en route nodes toward D. Upon receiving the bootstrap packet, R_j first obtains SID, RID, FID and $PC(1)$ from the authentication header. Since R_j is in the route, R_j should be able to identify a routing entry that has S, D and RID, and thus R_j knows L_j[2]. R_j can then verify $\delta_{R_j}(1)$. If the verification fails, the packet is discarded. If the verification is successful, R_j removes tokens (if any) for current and previous hops from $A(1)$ to save communication overhead (because $\delta_{R_j}(1)$ and all previous tokens are no longer useful for the following en route nodes to do verification).

R_j will create a new forwarding entry, because the packet is the first one R_j receives and R_j cannot find a forwarding entry matching this packet. R_j records SID and FID and keeps two copies of $PC(1)$ in the new forwarding entry. One copy of $PC(1)$ indicates the packet count of the first received packet, still denoted as PC_{1st}. The other copy indicates the packet count of the last received packet, denoted as PC_{last}. Then, R_j forwards the bootstrap packet to the next hop R_{j+1}; otherwise, it discards the bootstrap packet.

After this procedure, each node has SID, FID, PC_{1st} and PC_{last} to identify the corresponding forwarding entry, and is ready for the following verification. Note that when R_j receives $A(1)$, it will find that either no routing entry exists for S, D and RID, or no forwarding entry exists for SID and FID. Hence, the bootstrap packet is in fact the first packet that the en route node receives from the source. In this way, even the real bootstrap is lost in forwarding, each en route node can still have a valid forwarding entry bootstrapped by the first packet that the

[2]Because each en route node knows how many tokens should be left when it receives a packet, R_j does not directly obtaining L from the received packet. Instead, R_j calculates L by adding the data size, the number of headers, and the number of authentication tokens that should be at its own hop. This approach is applied to address verification in unreliability.

node receives later. Only after SID, FID, PC_{1st} and PC_{last} are all stored in the forwarding entry, this entry is bootstrapped for later forwarding and verification.

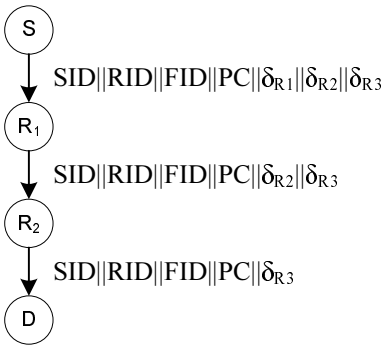


Fig. 5.8. An example of forwarding

5.4.2.3 Update

For each new data packet $PKT(i)$, S composes a new authentication header $A(i)$ as

$$A(i) = [SID||RID||FID||PC(i)||\delta_{R_1}(i)||...||\delta_{R_n}(i)] \tag{5.1}$$

Where $PC(i)$ is one unit increment of $PC(i-1)$, i.e. $PC(i) \leftarrow PC(i-1)+1$. Since the route does not change in a reliable environment, in $A(i)$, S still uses the forwarding entry FID and its own ID SID for the following packets. $\delta_{R_j}(i)$ is computed as follows, and the whole packet

is called *update packet* and sent into the route toward D.

$$\delta_{R_j}(i) = H_{k_{SID,R_j}}(RID||FID||PC(i)||L_j) \tag{5.2}$$

Upon receiving $PKT(i)$, R_j first obtains S, D, SID, RID, FID and $PC(i)$ from the packet, and finds the corresponding routing and forwarding entry, in which R_j retrieves PC_{last}. R_j verifies $A(i)$ and compares $PC(i)$ to PC_{last}. If the verification is successful and $PC(i)$ is greater than the last PC_{last}, R_j updates $PC_{last} = PC(i)$ and removes $\delta_{R_j}(i)$ and all tokens (if any) for previous hops from $A(i)$. Then, R_j forwards the data packet to the next hop R_{j+1}. Otherwise, i.e. the verification fails or $PC(i) \leq PC_{last}$, R_j discards the data packet. In case the bootstrap packet is lost, R_j may find that the verification is successful, but there is no forwarding entry for the verification of packet count. Then, R_j treats $PKT(i)$ as the bootstrap packet and goes to the bootstrap procedure described above.

In summary, an en route node always keeps SID, FID, PC_{1st} and PC_{last} in the forwarding entry. This information enables an en route node to verify whether a packet is from the claimed source and whether a packet is replayed in the same route. Furthermore, by computing a token for each en route node, the packet cannot be replayed in any other route in any other area in an ad hoc network, because only R_j can verify the token $\delta_{R_j}(1)$. Hence, these measures ensure that only the true source can use the route to forward data packets. Hence, the design of forwarding in a reliable route enables that

- Each en route node R_j can only verify one token $\delta_{R_j}(i)$.

- Each token $\delta_{R_j}(i)$ is computed with the pairwise key k_{SID,R_j} that can only be derived from SID and R_j.
- Each packet $PKT(i)$ is marked with the count $PC(i)$, which is secured in the tokens.
- The authenticity of L_j is provided by the tokens.

Note that, since the forwarding module works in the network layer, it does not matter which application session in the source is using the route or whether TCP is used in the transport layer. For example, if TCP is used in a source's application, the source will treat a data packet retransmitted by TCP as two independent data packets in the forwarding module. Finally, the authentication header is added as an extension of the IP header of the packet, but is not authenticated as the IP header in IPSEC.

5.4.3 Forwarding in an Unreliable Ad Hoc Network

The simple version of the forwarding module obviously overlooks the unreliability of the ad hoc network in which packet loss, route change and packet disorder could be fatal. Hence, we extend the forwarding module to handle these problems. In this section, we consider only the problems caused by the network itself. Some attacks can also result in similar problems or attackers may exploit these problems to attack the network. We will analyze these security issues in Section 5.5.

5.4.3.1 Packet Loss

A packet could be lost due to communication error, hardware error, buffer overflow, etc. However, this will not affect the forwarding module. If the bootstrap packet is lost at R_j, en route

nodes after R_j will treat the first received authentic data packet as the bootstrap packet for this forwarding entry. If an update packet is lost, the simple forwarding module will not be affected either. Assume R_j successfully receives $PKT(i)$ and updates $PC_{last} = PC(i)$, but the next several packets are lost until R_j successfully receives $PKT(i')$. Since $PC(i') > PC_{last}$ and R_j can still compute and verify $A(i')$, $PKT(i')$ will be accepted. Hence, the loss of several update packets will not affect the forwarding entry.

5.4.3.2 Route Change

As discussed before, a route change will make en route nodes unable to verify packets and thus drop packet to cause denial of service to legitimate traffic. The idea to solve the problem of route change in packet forwarding is to let node 2 start another forwarding procedure in the new route. Assume the old route is broken when node 2 tries to forward $PKT(\alpha)$ to node 3. Now, node 2 bootstraps a forwarding procedure in the new route as illustrated in Figure 5.9. It basically appends its own authentication header A^2 to the authentication header A^S in each data packet.

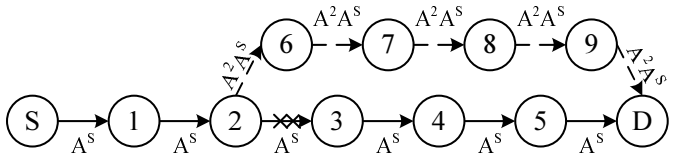


Fig. 5.9. Forwarding in a new route, which does not overlap with the old one.

When node 2 receives $PKT(\alpha)$, it should see an authentication header A^S as follows

$$A^S(\alpha) = [SID^S||RID^S||FID^S||PC(\alpha)||\delta^S_{R_2}(\alpha)||...||\delta^S_{R_5}(\alpha)||\delta^S_{R_D}(\alpha)]$$

Where $*^S$ means the information from the the source S, and

$$\delta^S_{R_j}(\alpha) = H_{k_{SID^S,R_j}}(RID^S||FID^S||PC(\alpha)||L^S_j)$$

Assume the packet has 100-byte data. Because the packet has only one authentication header and there should be 5 tokens when node 2 receives it, $L^S_2 = 100 + 1 + 5 = 106$. If node 3 can receive the packet, $L^S_3 = 100 + 1 + 4 = 105$.

Node 2 computes A^2 as if node 2 was the source of the new route.

$$A^2(\alpha) = [SID^2||FID^2||PC(\alpha)||\delta^2_{R_6}(\alpha)||...||\delta^2_{R_9}(\alpha)||\delta^2_{R_D}(\alpha)]$$

Where $*^2$ means the information from node 2, and

$$\delta^2_{R_j}(\alpha) = H_{k_{SID^2,R_j}}(RID^2||FID^2||PC(\alpha)||L^2_j)$$

Still assume the packet has 100-byte data. Because the packet has two authentication headers and there should be 5 tokens in A^2 when node 6 receives it, $L^2_6 = 100 + 2 + 5 = 107$. When node 7 receives the packet, $L^2_7 = 100 + 2 + 4 = 106$.

Node 2 appends A^2 to A^S. Hence, node 6 and all following nodes in the new route will see two authentication headers in packets. Because they can verify A^2, they will not discard

packets in the new route. Note that node 1 may not have any information about the new route and do not have any information of the new forwarding procedure in the new route. Node 1 may work as if nothing happens in the route. This new forwarding procedure works until S knows the new route and resets forwarding[3]. Finally, we name the nodes that bootstrap the forwarding procedure as *starter*, and SID in an authentication header is the starter ID instead of the source ID. In Figure 5.9, S is the starter in the old route, and node 2 is the starter in the new route.

5.4.3.3 Packet Disorder

As shown in the simple version, S increases PC for every update packet, and an enroute node only accepts an update packet with PC larger than the previous one. In this sense, PC represents the order of the packet delivery. However, if the order is mixed or reversed, the update packet with a smaller PC will be discarded in the simple SAF. We notice that the disorder can be caused by two reasons: either an attacker intentionally changes the order of the packets, or a route is changed. We will discuss the first reason in Section 5.5.

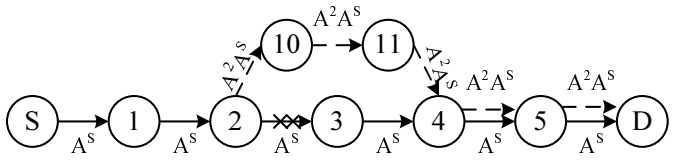


Fig. 5.10. Forwarding in a new route, which overlaps with the old one in some segment.

[3]In DSR, node 2 sends a "gratuitous" Route Reply to S that contains the IDs of the routing nodes in the new route. Hence, S can start a new forwarding procedure in the new route.

The second reason can be illustrated in Figure 5.10, where the solid lines represent the old route, the dashed lines represent the new route, and both routes overlap after node 4. Assume that node 3 is congested for a long time after the new route is discovered. Hence, the packets going through node 11 will reach node 4 before the old packets buffered in node 3. Because the packets buffered in node 3 have smaller PC, they will be discarded by node 4 if node 4 only records the latest PC in the packets from node 11 as in the simple SAF.

To solve this problem, we use both PC_{1st} and PC_{last} in the decision making. Based on the solution for route change, node 4 actually have obtained two forwarding entries for A^2 and A^S. In this two entries, we let node 4 records two packet counts: $PC^S_{1st} = PC(1)$ and $PC^2_{1st} = PC(\alpha)$ (assume the new route is set up by node 2 when forwarding $PKT(\alpha)$). Obviously, $PC(i')$ in any packet buffered in node 3 satisfies $PC^S_{1st} < PC(i') < PC^2_{1st}$; while $PC(i)$ in any packet going through node 11 satisfies $PC^2_{1st} < PC(i)$. We also let node 4 record PC_{last} for the two entries respectively, denoted as PC^S_{last} for the entry that has $SID = S$ and PC^2_{last} for the entry that has $SID = 2$. When node 4 receives an update packet $PKT(i)$, it compares $PC(i)$ with PC^S_{last} if $PC^S_{1st} < PC(i') < PC^2_{1st}$. Otherwise, it compares $PC(i)$ with PC^2_{last} if $PC^2_{1st} < PC(i)$.

5.4.4 Forwarding Algorithm

In summary, the forwarding algorithm has two components. The starter uses Aalgorithm 4 to compute authentication headers in packets, and en route nodes use Algorithm 5 to verify authentication headers in packets.

5.4.4.1 Starter

Algorithm 4 allows a starter to add a new authentication header in a packet when the packet cannot be delivered due to route change as described in Section 5.4.3.2. The algorithm consists of 5 phases.

In phase 1, the starter checks whether it is the source. Differing from other starters, the source node needs to set the packet count in a packet. If the packet is not the first packet that the source sends to the destination, the source should increase the packet count by one for each new packet. All other nodes (including other starters) simply record the packet count if the packet is authenticated.

In phase 2, the starter removes the last authentication header in a packet if the header was created by the starter itself. When the starter forwards a packet to the next hop, it is possible that the link to the next hop fails. In such a situation, the packet will be returned to the starter for retransmission in a new route. The returned packet has a header created by the starter. Hence, the starter needs to replace the old header with a new one.

Phase 3 is to discover a valid route for the packet. In DSR, due to a link failure to the next hop, a route could be revoked. It is also possible that the starter receives a route error message and revokes a route. Hence, if no route is available for the packet, the starter needs to discover a new route. In DSR, the routing packets will carry S, D and RID so that all en route nodes have the corresponding information of the new route.

In phase 4, the starter creates or updates the corresponding forwarding entry. If the packet is the first packet to be delivered in the new route, the starters need to create a new forwarding

Algorithm 4 SAF in a Starter

Assume the starter SID receives a packet PKT that should be sent from S to D. Assume the packet has m authentication headers $A^1 \cdots A^m$, where A^m is the authentication header for the latest route segment the packet will go through.

set SID in PKT to be the ID of the current node;
if $m = 0$ **then** ▷ ==Phase 1==
    if this is the first packet sent from S to D **then**
        set PC in PKT to be 1;
    else
        set PC in PKT to be a one-unit increment of PC in the previous packet;
    end if
end if
set $m' = m + 1$; ▷ ==Phase 2==
if In A^m, SID is the ID of the current node **then**
    remove A^m;
    set $m' = m$;
end if
if there is no valid route from S to D **then** ▷ ==Phase 3==
    find a new route to D
    create a routing entry that records the route and S, D, RID;
end if
set S, D and RID in PKT;
if there is no valid forwarding entry **then** ▷ ==Phase 4==
    create a forwarding entry F with a unique FID;
end if
set FID in PKT;
if there is no first packet count in the forwarding entry **then**
    set the first packet count in F to be PC in PKT;
end if
set PC_{last} in F to be PC in PKT; ▷ ==Phase 5==
compute and append a new authentication header $A^{m'}$ to PKT;
forward PKT to the next hop;
if forwarding fails **then**
    call Algorithm 4;
end if

entry. FID is used to uniquely identify the entry. If the forwarding entry has already been created, the starter simply records the current packet count.

Finally, in phase 5, the starter computes a authentication header and tokens as described in Eqs.(5.1) and (5.2). Then, the starter sends the packet to the next hop. In case the link to the next hop fails, the packet will be returned to the starter's forwarding module and the starter will execute Algorithm 4 again.

5.4.4.2 En Route Nodes

Algorithm 5 describes how an en route node verifies a received packet. It also consists of 5 phases.

In phase 1, the en route node verifies the token in the last authentication header. As discussed in Section 5.4.3.3, the node could receive packets from the same source via different routes. However it is obvious that the last authentication header reveals the route through which the packet goes. Hence, the node should verify whether the received packet is legitimate in the route it goes through. If it is not, the node should discard it.

Then, in phase 2, the en route node should check whether the received packet is the first one sent by the starter. If the packet is the first from the starter, the en route node should have no forwarding entry for the packet yet. Thus, the node should create a forwarding entry to record corresponding information as discussed in Section 5.4.2.2. Otherwise, the node should check the forwarding entry to see whether or not its PC is larger than the previous one. If the PC is smaller, the packet is replayed or forged and should be discarded. After passing the first two phases, the packet should be authenticated for the current route. The en route node removes the tokens for the current and the previous hops in the last authentication header.

Algorithm 5 SAF in an En Route Node

Assume an en route node receives a packet PKT that should be sent from S to D. Assume the packet has m authentication headers $A^1 \cdots A^m$, where A^m is the authentication header for the latest route segment the packet will go through.

obtain S, D, SID^m, RID^m, FID^m and PC from A^m;
find the routing entry R^m in the node according to S, D and RID^m;
verify the token for the current node in A^m; ▷ ==Phase 1==
if verification fails **then**
    discard the packet and quit;
end if
find the forwarding entry F^m in the node according to SID^m and FID^m; ▷ ==Phase 2==
if F^m does not exist **then**
    add a forwarding entry F^m;
    record SID^m and FID^m in F^m;
    record the first packet count $PC^m_{1st} = PC$ in F^m;
else if $PC \leq PC_{last}$ in F^m **then**
    discard the packet and quit;
end if
remove tokens for current and previous hops in A^m;
for i=1; $i \leq m-1$; i++ **do** ▷ ==Phase 3==
    obtain S, D, SID^i, RID^i, FID^i and PC from A^i;
    find the routing entry R^i in the node according to S, D and RID^i;
    if R^i exists **then**
        verify the token for the current node in A^i;
        if verification is not successful **then**
            discard this packet and quit;
        end if
        remove tokens for current and previous hops in A^i;
    end if
end for
for find all routing entries in the node according to S and D; **do** ▷ ==Phase 4==
    find a forwarding entry F;
    obtain PC_{1st} in F;
    if $PC \geq PC_{1st} > PC^m_{1st}$ **then**
        discard this packet and quit;
    end if
end for
set PC^m_{last} in F^m to be PC in PKT; ▷ ==Phase 5==
forward PKT to the next hop;
if forwarding fails **then**
    call Algorithm 4;
end if

In phase 3, the en route node checks other authentication headers. As shown in Figure 5.10, a packet may carry multiple authentication headers, each of which represents a possible route. For each authentication header, the node checks whether or not a routing entry exists for verification. If one does, the node verifies the token in the authentication header and discards the forged packet. However, the node does not check whether PC is larger than PC_{last} as it does in phase 2. The check of PC is done in phase 4.

As discussed in Section 5.4.3.3, PC must belong to the interval of the forwarding entry for the last authentication header. Hence, the node checks whether or not PC belongs to intervals of other forwarding entries. If PC does belong to another interval, the packet is replayed and should be discarded. In phase 4, the node first finds all of the forwarding entries that are used for packets forwarded from S to D. Then, the node checks whether PC falls in any other intervals. If $PC \geq PC_{1st} > PC^{m}_{1st}$, PC must be in another interval and thus the packet is replayed and should be discarded.

Finally, in phase 5, the packet has passed all verifications, and the en route node sends the packet to the next hop. In case the link to the next hop fails, the packet will be returned to the node's forwarding module and the node becomes a starter so it can execute Algorithm 4.

5.5 Security Analysis

It is possible that an attacker intends to "legally" inject junk packets into the network by using its own identity. Although action can be taken to stop the injection later, we cannot prevent such a "legal" injection. Nevertheless, the objective of this study is to force any attacker to expose its ID if it wants to inject and to quickly filter the junk packets when they are spoofed. Hence, in the following security analysis, we do not consider the attacker or anyone

in its coalition as a "legal" source. The attacker or its coalition could be a starter or an en route node.

5.5.1 Packet injection

PROPERTY 4. *If an attacker is an en route node, it is infeasible for the attacker to break tokens and obtain pairwise keys for other en route nodes.*

To inject junk packets, the attacker needs to provide valid tokens for the junk packets, which require the attacker to know pairwise keys for other en route nodes. Assume that both the length of a pairwise key and the size of hash output are h-bit. Given a packet and its hash, $O(2^{h-1})$ computations are required to break the hash and obtain the key. Notice that a token has l bits picked from the hash output (for instance, the last 8 bits of the hash output). However, the difficulty to break a hash function is determined by the size of key. Hence, it will still require $O(2^{l-1})$ computations for an attacker to break tokens given a packet and its token, as long as the hash function is collision free.

PROPERTY 5. *If the attacker is an en route node, the probability that a forged packet can survive is negligible.*

In order for an injected packet to not be filtered in the route, the attacker (as an en route node) needs to provide valid authentication tokens. Since it is infeasible for the attacker to break into tokens and obtain pairwise keys for other en route nodes, the attacker can only fabricate tokens. Hence, whether an attacker can forge a correct token is determined by the size of token. Assume a token has l bits. The attacker has a 1 in $O(2^l)$ chance to forge a correct token. For example, in this study, we use 8-bit tokens. Hence, the probability that a forged packet can be

accepted by the next hop is $\frac{1}{256}$. Accordingly, the probability that a forged packet can go through m hops is only $(\frac{1}{256})^m$. SAF can effectively filter out junk data packets injected by malicious en route nodes with high probability.

PROPERTY 6. *If an attacker injects in a route by guessing tokens, it is highly possible to detect the misbehavior.*

Because the probability for an attacker to successfully guess a token for the next hop is very small, the next hop of the attacker may experience continuous verification failure when the attacker tries to inject junk packets. Assume a simple intrusion detection algorithm that raises alert when v verifications fail continuously. Then, such a detection algorithm can detect the injection with a probability of $1 - \frac{v}{2^l}$, given that a token has l bits.

PROPERTY 7. *If the attacker claims to be a starter, it must use its own ID to inject junk packets.*

Because the attacker does not know the pairwise key that is only shared between the starter and the corresponding en route node and it is infeasible for the attacker to break the token, it is impossible for a node to impersonate another node to send the data packet. Assume node 2 in Figure 5.9 is the attacker claiming that the route is broken and it needs a new route to forward packets. By doing so, node 2 can inject junk packets into the new route. Node 2 must provide authentic A^2; otherwise, a data packet with a forged A^2 will be detected and discarded by node 2's next hops. But node 2 may forge A^S in junk packets for injection. In Figure 5.9, these forged packets will not be filtered by nodes 6 to 9, since they do not verify U^S. However, D can detect them, because D can verify both A^2 and A^S. If the new segment is the one in Figure 5.10, the forged packet will be detected by node 4. Hence, the attacker can "legally"

inject junk packets in the new route that does not overlap with the old route, but SAF exposes its ID to the destination and to the nodes in the overlapped route segment.

PROPERTY 8. *If the attacker claims to be a starter, it must use its own ID to inject extra bits into packets.*

Since the data size, the authentication header size and the number of previous authentication headers are protected in tokens, an attacker cannot insert junk bits into packets as an en route node. However, an attacker, if claiming to be a starter, can put junk bits into packets by forging some authentication headers for non-existing route segments. For example, in Figure 5.10, node 2 (as an attacker) can insert junk bits by putting junk authentication headers between A^2 and A^S. The downstream nodes can still verify A^2 and A^S, even though none of them can verify the junk headers. However, such an injection is in fact "legal" in the route, since the attacker uses its own identity to authenticate the packets. This situation also exists in legitimate traffic. As in the previous property, the destination node can detect such an injection, because the destination node knows all valid routes. Hence, the attacker exposes himself as well.

5.5.2 Misuse of SAF

An attacker may misuse SAF to cause other attacks. As an en route node, the attacker can drop, replay, disorder or modify the authentication headers in the packets it needs to forward. In the following, we analyze how misuse may impact the regular packet forwarding. We find that misuse of SAF generally results in the drop of misused data packets, but does not affect other legitimate data packets.

PROPERTY 9. *If an attacker intentionally modify the authentication header, the result is the same as that the attacker drops the packet.*

The attacker can modify any field in the authentication headers. This modification will easily fail verification and the modified packet will be discarded. Hence, the impact of modification is the same as the drop of the modified packet. Furthermore, as discussed in Section 5.4.3, if the bootstrap packet or any update packet is dropped, SAF is not affected.

PROPERTY 10. *If an attacker replays a packet, the packet will be discarded.*

An attacker may replay previous update packets in order to inject junk packets in any area of the network. If the packet is replayed in the same route, SAF easily filters it, since a good en route node only accepts the update packet whose PC is greater than the PC in the last data packet. If the attacker replays a previous update packet but increases PC in the replayed packet, the packet will be discarded since its authentication token AUT can be verified based on the increased PC. If the packet is replayed to another node not in the route, the packet will be filtered, because it cannot be verified by any other node outside the route.

PROPERTY 11. *If the attacker disorders the packets to be forwarded, the result is the same as that the attacker simply discards these disordered packets.*

Assume the attacker buffers a few update packets, but forwards the latest update packet (whose PC is the largest among all buffered update packets) first and then forwards previous update packets. This is how the attacker intentionally disorders the update packets. In SAF, a good en route node will accept the first forwarded update packet and then discard all the other buffered update packets. However, sooner or later, the buffered update packets will be depleted.

New update packets have larger PC and thus will be accepted by good en route nodes. Hence, if the attacker disorders a few update packets, only these packets will be discarded. The impact is the same as when the attacker simply drops these packets.

5.6 Performance Evaluation

We implemented the forwarding protocol in NS2 [35] to evaluate its performance. We examine how much overhead this protocol brings to each routing node in an ad hoc network. In the following, we first present the detail settings for the simulations, and then illustrate and discuss the impact of this protocol on data forwarding.

The simulation will use the following communication models. In the physical layer, the two-ray ground reflection model is used to model the signal propagation in the simulations. We chose the widely used IEEE 802.11 as the MAC and PHY protocols for the communications among mobile nodes. The CSMA and DCF functions are used to avoid transmission collision among nearby nodes. Each node has a sensing range of 550 meters and a transmission range of 250 meters. The maximum bandwidth of the channel is $1Mbps$. For communications over multiple hops, DSR is used as the routing protocol.

5.6.1 Comparison Targets

The base line in comparison the regular packet forwarding procedure, in which an en route node simply forwards a data packet according to the destination address and its routing table. SAF is supposed to be transparent to the regular packet forwarding and the routing protocol. Other source authentication protocols can also be the target for comparison. However, because

they cannot fit in unreliable ad hoc networks due to their inherent nature, these authentication protocols need to be improved based on SAF to survive in an ad hoc network.

5.6.2 Simulation Factors

The simulation factors have two categories. One is network topology, which includes the network area (m^2), node numbers, node distributions (uniform or centralized), node moving patterns (random or directional), etc. In NS2, we can use its scenario generation tool to generate network topologies with uniformly distributed nodes. According to the literature, an example setting for the network is as follows. The network is in a $1500m \times 1500m$ area, where 100 nodes are randomly put in the network. Nodes move randomly at the maximum speed of $2m/s$, $5m/s$ or $10m/s$.

The secondary category of simulation factors is traffic model, which includes the number of connections, types of connections (TCP or UDP), loads of connections ($Kbps$), payload of data packets ($byte$), traffic generation model of each connection (constant rate or Poisson rate), etc. For example, we can set 10 connections in the network. Each connection picks a random time during the first 5 seconds to start its traffic, and all traffic lasts 60 seconds. The load of each connection is $5Kbps$, $10Kbps$, $20Kbps$, $30Kbps$ or $40Kbps$, and the payload of a data packet is 512 bytes.

5.6.3 Performance Metrics

We measure four performance metrics of SAF in all scenarios. The first is *Data throughput per flow*, which directly illustrates the impact of SAF on the network, i.e. whether SAF interfere with regular packet forwarding. Data throughput per flow is measured as the data rate

(Kbps) of the data forwarding with SAF. If SAF does interfere with regular packet forwarding, we may see that the throughput of SAF is less than the regular throughput. The other three metrics are used to examine how and why SAF might interfere with regular data forwarding and what cost SAF brings to the network. *Communication overhead per hop* is measured as the number of bytes that are carried to each data packet. *Authentication per starter* is measured as the number of authentication tokens that a starter computes to authenticate a data packet. *Verification per hop* is measured as the number of authentication tokens that are designated to an en route for source verification. These metrics can show the direct cost SAF brings to the network, which may be the reason for the impact of SAF on the network (i.e. the first two metrics).

5.6.4 Evaluation Results

5.6.4.1 Overhead of SAF

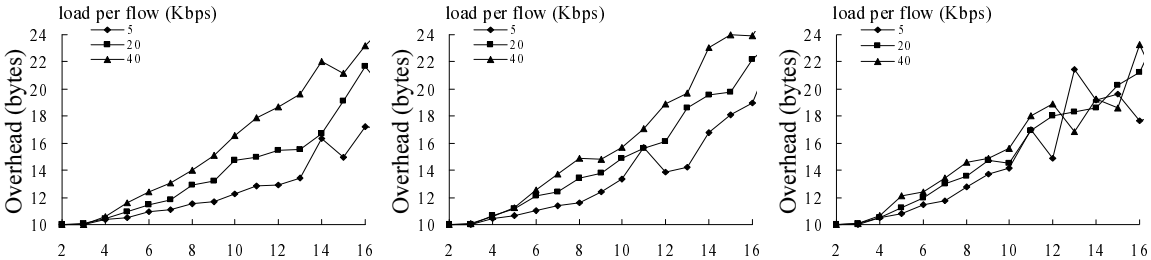


Fig. 5.11. Communication overhead per hop

SAF appends an authentication header of several bytes to every data packet, which includes the starter ID, the packet count, and authentication tokens. The size of the authentication token changes along the route, as an en route node removes its corresponding authentication tokens from a packet when it forwards the packet, or a starter adds new authentication tokens to a data packet for the new segment in the path[4]. Figure 5.11 shows the average overhead vs. the total hops of a path. Each sub figure shows the overhead at different maximum speeds, and each line in a sub figure represents the overhead at different loads.

As illustrated, the authentication header is larger when the path is longer. When the destination is far away from the source or the network is unreliable, a packet has to go through several new segments in the path. The overhead has a constant part about 10 bytes, and increases linearly to the total hops with a slope that is influenced by the load and the speed. Our simulation shows that a path with one more hop only adds 0.5 bytes to the average overhead when the load is light (5Kbps) and the speed is low (2m/s). On the other hand, when the load or the speed is high, the network becomes unreliable, and the overhead increases more quickly. In the unreliable environment (40Kbps and 10m/s), a path with one more hop could increase the overhead by more than 1 byte on average. Furthermore, when the speed is low, there is an obvious difference of slopes under various loads. While the speed is high, this difference is diminished. Compared with the sizes of payload, IP header and MAC header, the overhead of SAF is lightweight, around 10 to 24 bytes in our simulation.

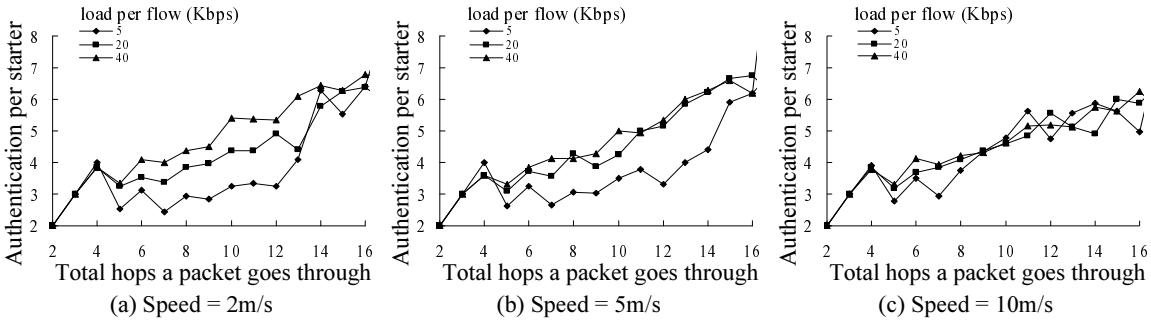


Fig. 5.12. Number of authentication tokens a starter needs to compute

5.6.4.2 Computation of SAF

The starter needs to compute authentication headers for data packets and each en route node needs to verify packet sources. The computational demand for starters, measured as the number of authentication tokens that a starter needs to compute, is depicted in Figure 5.12. Differing from the overhead, the computation for authentication does not increase as much as overhead when the path is longer. As we trace each data packet, we find that many data packets go through a path with several new segments before reaching the destination and each new segment needs a starter to compute a new authentication header. Hence, even when the whole path is longer, each starter in the path only computes for its own segment. However, the accumulative computation of all starters along the path might increase more as the path gets longer, which can be inferred from the average overhead of the path. Similar to overhead, network unreliability (higher load and speed) increases the computation for starters (although slightly). In the

[4]Here, a path means all hops a data packet goes through until reaching the destination. Hence, a path may contain several segments, each of which is set up when a new route is used to replace the broken one.

worst case (40Kbps and 10m/s), a starter needs to compute around 0.3 authentication tokens on average for each hop in the path.

The computation cost for each en route node, which is measured as the number of authentication tokens the node needs to verify, is depicted in Figure 5.13. In fact, the per hop computation is less related to the total hops. Hence, the figure directly shows the influences of load and speed on verification. Load is a more important factor than speed. When the load is low (5Kbps to 10Kbps), a little more than 1 verification is needed in each hop for each data packet. In the cases of light loads, more verification is needed when the speed is higher. When the load is between 10Kbps and 20Kbps, the verification quickly increases from 1.05 to 1.3. Then the increase is slowed down as the load is more than 20Kbps. Note that the maximum verification is less than 1.5 even in the very unreliable situation. This result, combined with the overhead, indicates that many new segments in a path do not overlap with the old segments. Hence, even if a data packet carries a large authentication header with many authentication tokens, each en route node may only find one or two tokens that are designated to it. In another words, many tokens for broken routes in an unreliable environment cannot be verified in the new segments, which is the reason that source authentication approaches in the literature are not suitable in ad hoc networks.

5.6.4.3 Throughput of SAF

Finally, we use Figure 5.14 to address the major concern on whether or not SAF will affect the throughput. For comparison, we also conduct simulations, where only regular DSR is used. In the figure, throughput of SAF is represented by solid lines, and DSR by dashed lines. SAF does not interfere with DSR when the network is reliable, i.e. light loads and low speeds.

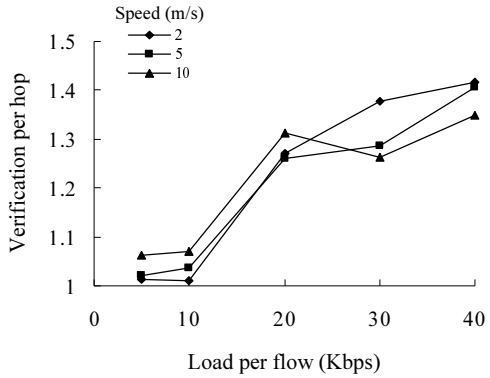


Fig. 5.13. Number of authentication tokens a hop needs to verify

When the load is more than $20Kbps$, the network becomes unreliable. Although the overhead of SAF becomes a factor in reducing throughput, the impact is statistically insignificant. For example, when the speed is $10m/s$, the throughput of SAF is very close to that of DSR. Hence, the design of SAF interfere negligibly with the regular data packet forwarding.

Figure 5.14 also implies that the packet injection attacks could seriously disrupt the legitimate traffic. According to the dashed lines (where attackers can inject because only DSR is applied), when the traffic load is $40Kbps$, the throughput is only $15Kbps$, which means more than 62% of packets are dropped. Hence, when attackers inject this kind of high load traffic, legitimate services will suffer due to the high percentage of packet drops. Figure 5.14 also shows that SAF is practical in an unreliable ad hoc network. The solid lines demonstrate that SAF can work even when more than 62% of packets are dropped. Note that the network may be disrupted by the legitimate loads, since SAF does not set any rate limit on legitimate traffic. Hence, the solid lines, which stand for the throughput of legitimate traffic, indicate that SAF not only protects the network, but also does not interfere with normal network traffic.

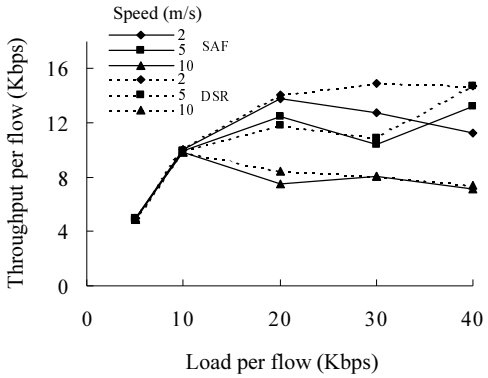


Fig. 5.14. Throughput comparison

5.7 Conclusion

To defend against packet injection DoS attacks in ad hoc networks, we present SAF, a hop-by-hop source authentication protocol in forwarding data packets. This protocol is designed to fit in the unreliable environment of ad hoc networks. The protocol can either immediately filter out injected junk data packets with very high probability or expose the true identity of the injector. For each data packet, the protocol adds a header of a few bytes for source authentication. Every en route node needs to verify less than 1.5 authentication tokens for each packet even when the network is very unreliable. Hence, the protocol is lightweight, interfering negligibly with regular packet forwarding.

Chapter 6

Conclusion

6.1 Summary of Thesis

In this study, we mainly investigated two critical security topics in wireless networks: key management in wireless broadcast services, and analysis and defense against packet injection attacks in wireless ad hoc networks.

In the first topic, a new key management approach, KTR, was proposed for service confidentiality and secure access control in wireless broadcast services. Since many approaches in secure multicast literature overlook a main feature of broadcast services that programs overlap with each other in terms of users, they require a user to handle a set of keys for each subscribed program and thus are highly demanding for the users who subscribe to multiple programs. KTR exploits the overlapping among programs and allows multiple programs to share a single tree so that the users subscribing to these programs can hold less keys. We used the key forest model to formalize and analyze the key structure in our broadcast system. It was shown that the shared key structure saves the keys to be handled in both the server and mobile devices.

Since multiple programs are allowed to share the same set of keys, we proposed an approach to handle shared keys which further reduces rekey cost by identifying the minimum set of keys that must be changed to ensure broadcast security. We found that, in many circumstances, when a user subscribes to new programs or unsubscribes to some programs, a large portion of keys that he will hold in his new subscription can be reused. Our simulation showed that KTR

can save about 45% of the communication overhead in the broadcast channel and about 50% of the decryption cost for each user, as compared with the traditional LKH approach.

In this study, we also investigated the packet injection attacks. Because there is no authentication in forwarding data packets, an attacker can inject junk packets into routes. We first studied the attack impacts of two types of packet injection attacks, and compared important factors that influence the attack impacts. Based on the simulations, we found that the remote attack is more effective and efficient for packet injection attackers to damage the network. It was noted that more flooding nodes and higher attack loads cannot increase or even reduce the attack impact in a remote attack. On the contrary, the local attack needs more resources than the remote attack. The damage in a local attack grows if more flooding nodes send traffic at a higher attack load in the network. In the simulations, good traffic also has attack impact on itself, and the packet injection attacks bring additional damage to the network.

To defend against packet injection DoS attacks in ad hoc networks, we presented SAF, a hop-by-hop source authentication protocol in forwarding data packets. This protocol is designed to fit in the unreliable environment of ad hoc networks. The protocol can either immediately filter out injected junk data packets with very high probability or expose the true identity of the injector. For each data packet, the protocol adds a header of a few bytes for source authentication. Every en route node needs to verify less than 1.5 authentication tokens for each packet even when the network is very unreliable. Hence, this protocol is lightweight with little interference on regular packet forwarding.

6.2 Future Works

Key Distribution in a Multi-hop Network for Broadcast Services The KTR approach only addresses how to change keys when a user changes the subscription. The server needs to distribute new keys to all nodes in the network. It is not a big problem in a one-hop network (e.g. an infrastructure network), because the server can directly broadcast the rekey message to all mobile devices. However, in a multi-hop network (e.g. an ad hoc network), the integrity of a rekey message is not ensured, because the rekey message will be forwarded over several hops to reach all nodes in the network. Even though the message is encrypted, it may be forwarded by an attacker or a compromised member that has the keys to decrypt the message. The attacker can modify the rekey message, and thus cause DoS to other members that receive the modified rekey message.

Besides digital signature, many multicast and broadcast authentication approaches have been proposed in the literature. They amortize the cost of a digital signature over multiple packets [37, 54], or ask users to buffer a received packet and wait for a later verification until they receive the authentication credentials [40]. They are not suitable in key distribution, because key distribution is different from general applications. In broadcast services, the rekey messages are determined by user activities. Users cannot wait until the server has a block of rekey messages ready for authentication. Users also need to verify new keys upon receiving the rekey messages. Therefore, the authentication of rekey messages should be on demand and immediate.

An idea to provide the rekey authentication is based on the feature of current broadcast protocols [52]. Because a node can receive multiple broadcast packets in a broadcast protocol, it can select the correct message from the majority of the received broadcast message if we assume

that attackers are not the majority. A server can divide the rekey message into several parts and broadcast them one by one with an interval. In this way, an attacker cannot modify the rekey message until he receives all parts. At the same time, the the good nodes may have already received the good parts to decrypt the new keys. More study is deserved for security analysis on this issue.

Isolation of Attackers in Ad Hoc Networks The SAF protocol targets filtering junk packets and exposing the injector's ID. Nevertheless, the protocol only partially achieves defense objectives, since the injector should be isolated and removed from the network. Because only the source can authenticate the packets, a failed verification indicates that the claimed source is injecting in the network. Hence, failed verifications are the evidence of the injection attack. Basically, the defender can infer that either the source or the forwarding node that is the sender in the link where the failed verification happened is an injector. However, such an inference based on failed verification may be misused by an attacker to deceive the defender.

When a forwarding node verifies a packet, it does not leave any information in the packet to indicate the verification is committed. Hence, it is easy for an attacker to hoax in the forwarding protocol. An attacker can intentionally modify one of the authentication tokens carried in a packet so that the packet cannot pass verification in the node which verifies the modified token. In this misuse way, the attacker does not inject packets, but deceives the defender to treat the source as the injector. It is also possible that a malicious node intentionally claims verification failures when receiving packets. Then, the defender will treat the source as the injector based on the claimed verification failures.

The basic question for the defender is how to decide whether a claimed verification failure is caused by an injector or faked by a liar. The way to solve the puzzle is to re-design the authentication tokens. In the current hop-by-hop authentication approaches, tokens are separately computed for each forwarding nodes. An idea is to combine the tokens into one unit so that the integrated token can be individually verified by each forwarding node and at the same time it is very hard for a forwarding node to modify it. Hence, it needs more efforts to find the efficient and secure combination of tokens.

References

[1] I. Aad, J.P. Hubaux, and E. Knightly. Denial of service resilience in ad hoc networks. In *ACM MobiCom*, 2004.

[2] John Bellardo and Stefan Savage. 802.11 denial-of-service attacks: real vulnerabilities and practical solutions. In *USENIX Security Symposium*, pages 15–28, Washington D.C., 2003.

[3] C. Blundo and A. Cresti. Space requirements for broadcast encryption. In *Advances in Cryptology, Eurocrypt*, pages 471–486, 1994.

[4] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: the insecurity of 802.11. In *ACM MobiCom*, pages 180–189, 2001.

[5] Bob Briscoe. Marks: zero side effect multicast key management using arbitrarily revealed key sequences. In *NGC*, pages 301–320, 1999.

[6] Federico Cali, Marco Conti, and Enrico Gregori. Ieee 802.11 protocol: design and performance evaluation of an adaptive backoff mechanism. *IEEE Journal on Selected Areas in Communications*, 18(9):1774–1786, 2000.

[7] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: a taxonomy and some efficient constructions. In *IEEE Infocom*, volume 2, pages 708–716, 1999.

[8] Haowen Chan and Adrian Perrig. Pike: peer intermediaries for key establishment in sensor networks. In *IEEE Infocom*, 2005.

[9] Wenliang Du, Jing Deng, Yunghsiang S. Han, and Pramod K. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *ACM CCS*, pages 42–51, 2003.

[10] Laurent Eschenauer and Virgil D. Gligor. A key-management scheme for distributed sensor networks. In *ACM CCS*, pages 41–47, 2002.

[11] Amos Fiat and Moni Naor. Broadcast encryption. In *Advances in Cryptology, CRYPTO*, pages 480–491, 1994.

[12] Zhenghua Fu, Petros Zerfos, Kaixin Xu, Haiyun Luo, Songwu Lu, Lixia Zhang, and Mario Gerla. The impact of multihop wireless channel on tcp throughput and loss. In *IEEE Infocom*, volume 3, pages 1744–1753, San Francisco, 2003.

[13] P. Golle and N. Modadugu. Authenticating streamed data in the presence of random packet loss. In *NDSS*, pages 13–22, 2001.

[14] Qijun Gu, Peng Liu, and Chao-Hsien Chu. Tactical bandwidth exhaustion in ad hoc networks. In *the 5th Annual IEEE Information Assurance Workshop*, pages 257–264, West Point, NY, 2004.

[15] Piyush Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transaction on Information Theory*, 46(2):388–404, 2000.

[16] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda. Performance anomaly of 802.11b. In *IEEE Infocom*, pages 836–843, 2003.

[17] Yih-Chun Hu, D.B. Johnson, and A. Perrig. Sead: secure efficient distance vector routing for mobile wireless ad hoc networks. In *the 4th IEEE Workshop on Mobile Computing Systems and Applications*, pages 3–13, 2002.

[18] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Ariadne: a secure on-demand routing protocol for ad hoc networks. In *ACM MobiCom*, pages 12–23, 2002.

[19] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Rushing attacks and defense in wireless ad hoc network routing protocols. In *ACM workshop on Wireless security*, pages 30–40, 2003.

[20] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Sead: secure efficient distance vector routing for mobile wireless ad hoc networks. *Ad Hoc Networks*, 1(1):175–192, 2003.

[21] Yi-an Huang and Wenke Lee. A cooperative intrusion detection system for ad hoc networks. In *the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 135–147, 2003.

[22] IEEE. Wireless lan medium access control (mac) and physical (phy) layer specification, June 1999.

[23] D. Johnson, D. Maltz, Y. C. Hu, and J. Jetcheva. The dynamic source routing protocol for mobile ad hoc networks (dsr), ietf internet draft, draft-ietf-manet-dsr-09.txt, Feb. 2002.

[24] Mike Just, Evangelos Kranakis, Danny Krizanc, and Paul van Oorschot. On key distribution via true broadcasting. In *ACM CCS*, pages 81–88, 1994.

[25] Chris Karlof, Naveen Sastry, Yaping Li, Adrian Perrig, and Doug Tygar. Distillation codes and applications to dos resistant multicast authentication. In *NDSS*, 2004.

[26] Yongdae Kim, Adrian Perrig, and Gene Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *ACM CCS*, pages 235–244, 2000.

[27] Donggang Liu and Peng Ning. Establishing pairwise keys in distributed sensor networks. In *ACM CCS*, pages 52–61, 2003.

[28] Donggang Liu, Peng Ning, and Kun Sun. Efficient self-healing group key distribution with revocation capability. In *ACM CCS*, pages 231–240, 2003.

[29] M. Luby and J. Staddon. Combinatorial bounds for broadcast encryption. In *Advances in Cryptology, Eurocrypt*, pages 512–526, 1998.

[30] Sergio Marti, T. J. Giuli, Kevin Lai, and Mary Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *ACM MobiCom*, pages 255–265, Boston, Massachusetts, United States, 2000. ACM Press New York, NY, USA.

[31] Suvo Mittra. Iolus: a framework for scalable secure multicasting. In *ACM SIGCOMM*, volume 277-288, 1997.

[32] M. Moyer, J. Rao, and P. Rohatgi. Maintaining balanced key trees for secure multicast. *draft-irtf-smug-key-tree-balance-00.txt*, 1999.

[33] Dalit Naor, Moni Naor, and Jeffrey B. Lotspiech. Revocation and tracing schemes for stateless receivers. In *Advances in Cryptology, CRYPTO*, pages 41–62, 2001.

[34] Peng Ning and Kun Sun. How to misuse aodv: a case study of insider attacks against mobile ad-hoc routing protocols. In *the 4th Annual IEEE Information Assurance Workshop*, pages 60–67, West Point, 2003.

[35] NS2. The network simulator, http://www.isi.edu/nsnam/ns/, 2004.

[36] P. Papadimitratos and Z.J. Haas. Secure routing for mobile ad hoc networks. In *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference*, San Antonio, TX, 2002.

[37] Jung Min Park, Edwin K. P. Chong, and Howard Jay Siegel. Efficient multicast packet authentication using signature amortization. In *IEEE Symposium on Security and Privacy*, page 227, 2002.

[38] Jung Min Park, Edwin K. P. Chong, and Howard Jay Siegel. Efficient multicast stream authentication using erasure codes. *ACM Transactions on Information and System Security*, 6(2):258–285, 2003.

[39] C.E. Perkins, E.M Royer, and Samir R. Das. Ad hoc on-demand distance vector (aodv) routing, ietf internet draft, draft-ietf-manet-aodv-11.txt, June 2002.

[40] A. Perrig, R. Canetti, J.D. Tygar, and Dawn Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, pages 56–73, Berkeley, CA, 2000.

[41] A. Perrig, D. Song, and D. Tygar. Elk, a new protocol for efficient large-group key distribution. In *IEEE Symposium on Security and Privacy*, pages 247–262, 2001.

[42] Adrian Perrig, Ran Canetti, Dawn Song, and Doug Tygar. Efficient and secure source authentication for multicast. In *NDSS*, 2001.

[43] Adrian Perrig, John Stankovic, and David Wagner. Security in wireless sensor networks. *Communications of the ACM*, 47(6):53–57, June 2004.

[44] Kimaya Sanzgiri, Bridget Dahill, Brian Neil Levine, Clay Shields, and Elizabeth M. Belding-Royer. A secure routing protocol for ad hoc networks. In *IEEE ICNP*, pages 78–89, 2002.

[45] Sanjeev Setia, Samir Koussih, Sushil Jajodia, and Eric Harder. Kronos: a scalable group re-keying approach for secure multicast. In *IEEE Symposium on Security and Privacy*, pages 215–228, 2000.

[46] J. Snoeyink, S. Suri, and G. Varghese. A lower bound for multicast key distribution. In *IEEE Infocom*, volume 1, pages 422–431, 2001.

[47] D. Song, D. Zuckerman, and J.D. Tygar. Expander graphs for digital stream authentication and robust overlay networks. In *IEEE Symposium on Security and Privacy*, pages 241–253, 2002.

[48] J. Staddon, S. Miner, M. Franklin, D. Balfanz, M. Malkin, and D. Dean. Self-healing key distribution with revocation. In *IEEE Symposium on Security and Privacy*, pages 241–257, 2002.

[49] Yan Sun and K.J. Ray Liu. Scalable hierarchical access control in secure group communications. In *IEEE Infocom*, 2004.

[50] D. Wallner, E. Harder, and R. Agee. Key management for multicast: issues and architectures. *IETF RFC 2627*, 1999.

[51] Yu Wang and J.J. Garcia-Luna-Aceves. Performance of collision avoidance protocols in single-channel ad hoc networks. In *IEEE ICNP*, pages 68–77, 2002.

[52] Brad Williams and Tracy Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *ACM Mobihoc*, pages 194–205, 2002.

[53] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. Secure group communications using key graphs. In *ACM SIGCOMM*, pages 68–79, 1998.

[54] Chung Kei Wong and Simon S. Lam. Digital signatures for flows and multicasts. *IEEE/ACM Transactions on Networking*, 7(4):502–513, 1999.

[55] Chung Kei Wong and Simon S. Lam. Keystone: a group key management service. In *International Conference on Telecommunications*, 2000.

[56] Avishai Wool. Key management for encrypted broadcast. *ACM Transactions on Information and System Security*, 3(2):107–134, 2000.

[57] Chris Wullems, Kevin Tham, Jason Smith, and Mark Looi. Technical summary of denial of service attack against ieee 802.11 dsss based wireless lans. Technical report, Information Security Research Centre, Queensland University of Technology, Brisbane, Australia, 2004.

[58] J. Xu, D.L. Lee, Q. Hu, and W.-C. Lee. Data broadcast. In Ivan Stojmenovic, editor, *Handbook of Wireless Networks and Mobile Computing*, pages 243–265. John Wiley and Sons, New York, 2002.

[59] Yang Richard Yang, X. Steve Li, X. Brian Zhang, and Simon S. Lam. Reliable group rekeying: a performance analysis. In *ACM SIGCOMM*, pages 27–38, 2001.

[60] Fan Ye, Haiyun Luo, Songwu Lu, and Lixia Zhang. Statistical en-route detection and filtering of injected false data in sensor networks. In *IEEE Infocom*, 2004.

[61] Seung Yi, Prasad Naldurg, and Robin Kravets. Security-aware ad hoc routing for wireless networks. In *ACM MobiHoc*, pages 299–302, 2001.

[62] Manel Guerrero Zapata and N. Asokan. Securing ad hoc routing protocols. In *ACM workshop on Wireless Security*, pages 1–10, Atlanta, GA, USA, 2002. ACM Press New York, NY, USA.

[63] Wensheng Zhang and Guohong Cao. Group rekeying for filtering false data in sensor networks. In *IEEE Infocom*, 2005.

[64] Yongguang Zhang and Wenke Lee. Intrusion detection in wireless ad-hoc networks. In *ACM MobiCom*, pages 275–283, 2000.

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

[66] Sencun Zhu, Shouhuai Xu, S. Setia, and S. Jajodia. Establishing pairwise keys for secure communication in ad hoc networks: a probabilistic approach. In *IEEE ICNP*, pages 326–335, 2003.

Vita

Qijun Gu enrolled in the School of Information Sciences and Technology at the Pennsylvania State University in 2001 to pursue Ph.D. degree. He received Master and Bachelor degrees in Department of Electronics at Peking University, Beijing, China, in 2001 and 1998 respectively. He is a student member of IEEE. His research interests cover security, network and telecommunication. He has been working on several projects, including denial of service in wireless networks, key management in broadcast services, worm propagation and containment, genetic algorithm in network optimization, etc. during his Ph.D. study.