uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Xu Li

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Master of Computer Science

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Secure and Anonymous Routing in Wireless Ad-Hoc Networks

TITRE DE LA THÈSE / TITLE OF THESIS

Azzedine Boukerche

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Chung-Horng Lung

Amiya Nayak

Gary W. Slater

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

Secure and Anonymous Routing in Wireless Ad-Hoc Networks

By

Xu Li

A Thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of

The requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institution for Computer Science

School of Information Technology and Engineering

University of Ottawa

Ottawa, Ontario, Canada

Your file Votre référence
ISBN: 0-494-11325-1
Our file Notre référence
ISBN: 0-494-11325-1

The following publications by the author are relevant to this thesis:

Journal

- A. Boukerche, K. El-Khatib, X. Li and L. Korba. "A Secure Distributed Anonymous Routing Protocol for Ad Hoc Wireless Networks". *Journal of Computer Communications*, Elsevier. (To appear).

Conference

- A. Boukerche, K. El-Khatib, X. Li and L. Korba. "A Novel Solution for Achieving Anonymity in Wireless Ad hoc Networks". In *Proceedings of the 1st ACM International Workshop on Performance Evaluation of Wireless Ad hoc, Sensor, and Ubiquitous Networks 2004*, pages 30-38, Venice, Italy, Oct. 2004.

- A. Boukerche, K. El-Khatib, X. Li and L. Korba. "SDAR: A Secure Distributed Anonymous Routing Protocol for Wireless and Mobile Ad hoc Networks". In *Proceedings of the 29th IEEE Conference on Local Computer Networks 2004*, pages 618-624, Tampa, U.S., Nov. 2004. (Nominated for the Best Paper Awards in the 4th international IEEE workshop on wireless local networks, held in conjunction with the 29th IEEE Conference on Local Computer Networks.)

- A. Boukerche, K. El-Khatib, X. Li and L. Korba. "Anonymity Enabling Scheme for Wireless Ad Hoc Networks", *IEEE GlobeCom 2004 Wireless Ad Hoc and Sensor Networks*. Houston, U.S., 2004.

Abstract

In wireless ad-hoc networks, malicious nodes can constitute a threat to the security and/or anonymity of the exchanged data between communicating nodes. While data encryption can protect traffic content, plain routing headers may reveal valuable information about end nodes and their relationship. The main purposes of this thesis are to study the feasibility of achieving anonymity in wireless ad-hoc networks, and to propose a secure distributed anonymous routing protocol (SDAR), which is similar to the onion routing concept used in wired networks. SDAR employs a special mechanism to establish trust among wireless nodes and to avoid untrustworthy ones during route discovery processes. The major objective of SDAR is to allow only trustworthy nodes to participate in route construction without jeopardizing the anonymity of communicating nodes. In this thesis, we elaborate on the SDAR protocol and report on its performance evaluation using an extensive set of simulation experiments; last but not least, we present the preliminary work we have done on an agent-based trust and reputation management scheme (ATRM) for wireless sensor networks.

Acknowledgements

I would like to thank my supervisor Dr. Azzedine Boukerche for his confidence in my abilities, his precious comments and financial support on this work. He has been always more than generous in both his time and his knowledge. Without his supervision, I would not be able to finish this work.

The main body of this thesis, SDAR routing protocol, is done in collaboration with National Research Council Canada (NRC). I would like to thank the computer scientists from NRC, Khalil El-Khatib, Larry Korba and Ronggong Song. Their previously-published work[1] served as the base of the development of SDAR. Without the cooperation of K. El-Khatib, L. Korba and R. Song, this work would not have been possible.

I would like to show special gratitude to K. El-Khatib for his help in reviewing this thesis.

I would also like to thank my parents and my wife Kaiyuan for their unconditional support and constant encouragement. I will never forget the inspiring words they gave to me when I felt frustrated. Without them, I could not have finished this work smoothly.

[1] K. El-Khatib, L. Korba, R. Song and G. Yee. "Secure Dynamic Distributed Routing Algorithm for Ad Hoc Wireless Networks". In *Proceedings of the 1st International Workshop on Wireless Security and Privacy*, pages 359-366, Kaohsiung, Taiwan, Oct. 2003.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the rapid advance in hardware technologies, computers and other communication devices are made smaller and smaller to facilitate carrying. In addition, with the help from swift-developing wireless communication technologies such as 802.11[1], HiperLAN[2] and Bluetooth[3], communication devices are getting off the constraint from wires and become capable to keep communication connections while moving. Under these circumstances, the small portable and mobile devices with wireless communication capability are able to perform spontaneous communication and constitute a dynamic environment, namely a wireless ad-hoc network, without the interference from any centralized administration or fixed infrastructure. In this chapter, we first give an overview on wireless ad-hoc networks and briefly introduce some relevant open research problems. Thereafter, we present the motivation of this thesis and outline its organization.

[1] 802.11 refers to a family of specifications developed by the IEEE for wireless LAN technology.
[2] HiperLAN is a family of specification developed by the ETSI for wireless LAN technology.
[3] Bluetooth technology is an open specification that enables short-range wireless connections between communication devices.

1.1 Wireless Ad-hoc Networks

The continuous advance in wireless and mobile communication technologies coupled with the recent proliferation of portable computer devices has led development efforts for future ubiquitous wireless networking towards wireless ad-hoc networks. A wireless ad-hoc network is a dynamic environment, where a group of battery-powered nodes communicate with each other through radio frequency without the intervention of any centralized controller or pre-defined infrastructure. It can be satisfactorily used in situations such as emergency rescue and battlefield scenarios, where deploying cables is impractical, and/or where the communication infrastructure is destroyed.

In a wireless ad-hoc network, nodes act autonomously without the use of any predefined infrastructure. Because of their limited transmission ranges, each wireless node functions both

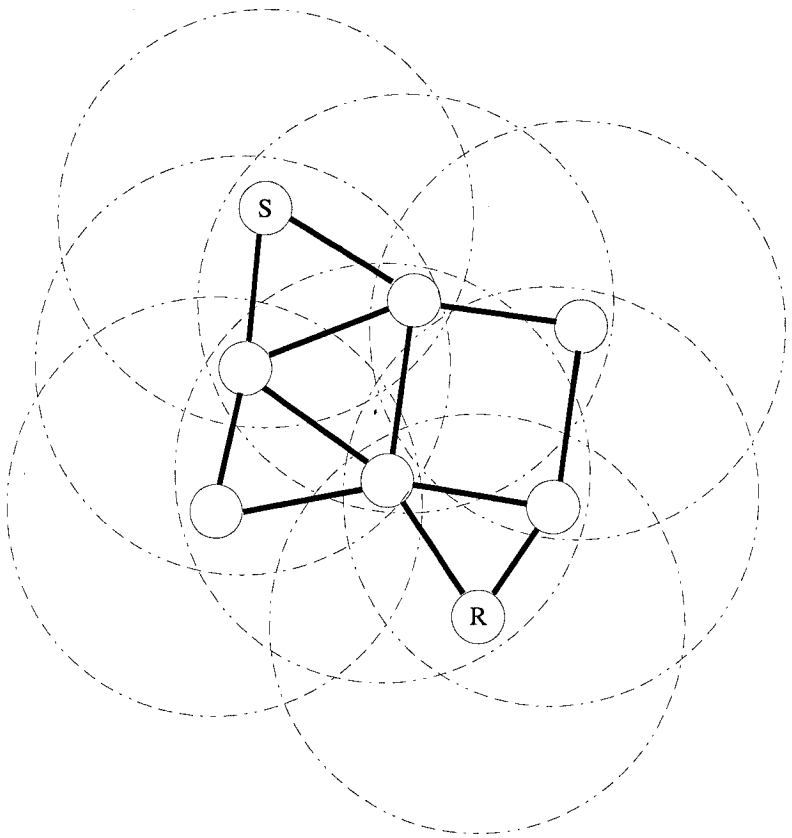


Figure 1.1: A wireless ad-hoc network with eight hosts

as a host and as a router. Figure 1.1 shows a wireless ad-hoc network with eight nodes. The dotted circle centered on each node shows the radio communication range of the node. Any connection between node S and node R in the figure has to go through other intermediate nodes.

In a wireless ad-hoc network, nodes could be either mobile or static. The wireless ad-hoc networks composed of mobile nodes are usually called *Mobile Ad-hoc Networks* (MANETs), while *Wireless Sensor Networks* (WSNs) are a typical example of static wireless ad-hoc networks. The characteristics of wireless ad-hoc networks are summarized as follows:

- *Self-organized network*

  There is no centralized administration or fixed infrastructure in a wireless ad-hoc network. Wireless nodes behave autonomously and share network control with all the other nodes.

- *Narrow bandwidth*

  The low throughput of wireless communication channels leads to the narrow bandwidth of wireless ad-hoc networks.

- *Limited lifetime*

  Battery-powered wireless nodes are able to operate only for a limited period of time, and consequently, the lifetime of wireless ad-hoc networks composed of such nodes is restricted.

- *Dynamicity*

  In a wireless ad-hoc network, it is a common phenomenon that nodes suddenly fail due to hardware defect, system crash, power exhaust, harshness of physical surroundings, or any other possible reason. Nodes can also roam freely without any restrictions on locality. Unpredictable node failure together with node mobility causes frequently-changing network topology.

- *Broadcast transmission:*

  The medium access control (MAC) problem, which is arbitrated by base stations in cellular networks, turns into a complicated task in wireless ad-hoc networks because of the absence of centralized administration. In wireless ad hoc networks, medium access procedures are carried out in a distributed and collaborative fashion. That is, every single node actively competes for medium access. As a result, the packet transmissions from different nodes are very likely to overlap with each other, and therefore, nodes have the ability to overhear the communication of their neighbors.

- *Multi-hop communication*

  Two nodes can directly communicate as long as they are within the radio communication range of each other because of the broadcast nature of wireless communication. However, the range of a node's radio is generally short because of the power constraint. So, any two nodes, which are not in each other's transmission range and are intending to communicate, have to establish communication through other intermediate nodes between them in a message relay manner.

- *Vulnerability*

  The openness of wireless ad-hoc networks makes adversaries capable to freely enter the network and perform *active attacks*, while the wire-free strength and the broadcast nature of wireless communication facilitate *passive attacks*. In addition, the communication between two nodes in wireless ad-hoc networks relies mainly on relaying messages through other intermediate nodes, some of which might be malicious or compromised. In these cases, network security can be at risk.

- *Unreliable communication*

  By reason of the bandwidth constraint, the dynamicity and the vulnerability of wireless ad-hoc networks, network congestion and route breaking may occur very often. Packet delivery can not be guaranteed.

1.2 Problem Statement

Due to the fast and easy deployment of wireless ad-hoc networks as well as their potentials in both military and civilian applications, these networks have been extensively studied in the recent decade. With the unique network properties discussed in the previous section, wireless ad-hoc networks bring many challenges to researchers. Some of the main research issues in wireless ad-hoc networks, i.e., *routing*, *security*, *energy-conservation*, *scalability*, and *fault-tolerance*, are briefly presented below:

- *Routing*

  Efficient routing is a primary challenge in wireless ad-hoc networks. Existing wireless ad-hoc routing protocols can be basically classified into two categories, table-driven (proactive) and on-demand (reactive). The former category derives from conventional distance-vector or link-state routing algorithms. In this type of routing protocols, each node builds a global view of the network by maintaining one or more routing tables where routing information is stored. Periodical route update advertisements are used to report any topological change and keep routing tables up-to-date. The advantage of table-driven routing is that the route to any destination is always available when needed. However, the periodical route update advertisements cause the drainage of battery power as well as large amounts of traffic overhead. The other category includes on-demand routing protocols, which are known for their on-demand route discovery. In an on-demand routing protocol, a route discovery process is triggered by the route request flood as needed. Compared to table-driven routing protocols, on-demand routing protocols have advantages in both power and bandwidth saving but incur route discovery delay.

- *Security*

  Security is a complex issue in wireless ad-hoc networks. The complexity stems from a number of factors, including the wireless medium, node mobility, and the lack of

infrastructure. On the one hand, malicious nodes (or attackers) can easily eavesdrop on wireless communication channels and interfere with network communication. On the other hand, communicating nodes have to rely on unknown intermediate nodes for message relay. This openness of wireless ad-hoc networks makes wireless nodes more susceptible to both *active attacks* and *passive attacks*. Unfortunately, the traditional security mechanisms designed for wired networks cannot be applied directly to wireless ad-hoc networks due to the particular properties of wireless ad-hoc networks.

- *Energy-conservation*

  Wireless nodes are usually powered by small and low-energy batteries. The energy requirement of fundamental network operations and the limited lifetime of batteries make energy conservation one of the main issues in wireless ad-hoc networks. There have been an amount of research [27, 55, 59] at various protocol layers to reduce the power consumption of wireless nodes. The typical approaches to save energy include: turning off radio signals when the node is idle, avoiding unnecessary retransmission, performing energy-aware routing, reducing network-wide flooding, etc.

- *Scalability*

  According to [32], in a wireless network, the node throughput is determined by the combination of channel capacity and the number of nodes in the network. The node throughput decreases with the increase of the number of nodes when channel capacity is fixed. Hence, scalability becomes an important issue in the wireless ad-hoc networks such as wireless sensor networks (WSNs) containing a large number of nodes. Clustering is a good option to solve this problem, and it has been continually studied by many researchers [5, 6, 28, 46].

- *Fault-tolerance*

  Fault-tolerance have been studied for several decades in network paradigm to deal with various failures, for example, node failure, communication failure, and system failure.

Redundancy is a popular approach to achieve the fault-tolerance feature in all kinds of applications. However, in wireless ad-hoc networks, fault-tolerance mechanisms are required to have low cost while still providing high resilience against node and communication failures. So, traditional large-redundancy approaches are too resource-consuming to be optimal solutions in wireless ad-hoc networks. Effective, efficient and lightweight fault-tolerance mechanisms are highly expected for wireless ad-hoc networks.

1.3 Motivation

Wireless ad-hoc networks have attracted considerable research attention in recent years. This is mainly due to their wide range of potential application areas. All of the wireless ad-hoc networking problems presented earlier need in-depth investigation and research. In this thesis, we concentrate only on the security issue of wireless ad-hoc networks. More specifically, the main focus is on secure and anonymous routing in wireless ad-hoc networks.

In a hostile environment such as wireless ad-hoc networks, the information exchanged between two communicating nodes may include highly sensitive data that must be protected when transmitted over insecure communication channels. While end-to-end security mechanisms can provide a certain level of security to transmitted data, valuable information such as the location and relationship of communicating nodes may be easily disclosed from traffic/data analysis. Network-based anonymity techniques may offer the prospect of hiding this kind of information.

For the Internet[1], there are several network-based anonymity techniques that can support anonymous communication between end nodes. These techniques include DC-nets [15], MIX-nets [14], Onion Routing [66] and Crowds [67]. Both MIX networks and Onion Routing share the same concept of establishing anonymous paths for data transfer. To construct an anonymous path, they require that source node store and maintain the information about network

[1] Internet is a world-wide communication network that links computer networks and organizational computer facilities around the world.

topology. But keeping up-to-date topological information is complicated and expensive in the absence of fixed infrastructure and in the presence of dynamic topology, as is the case with wireless ad-hoc networks. In this thesis, a novel secure distributed routing protocol using onion routing concept for anonymous communication in wireless ad-hoc networks is proposed, and its performance evaluation based on our experimental results is also presented.

1.4 Thesis Outline

The remainder of this thesis is organized as follows:

- Chapter 2 briefly introduces typical attacks against wireless ad-hoc networks, presents network security primitives, discusses anonymity and anonymity-enabling schemes, and finally talks about some incentive schemes for node collaboration.
- Chapter 3 presents a review of the previous work on wireless ad-hoc routing in literature.
- Chapter 4 is the core of the thesis. It throughly presents our proposed secure distributed anonymous routing protocol (SDAR) for wireless ad-hoc networks, analyzes its characteristics, and proves its correctness.
- Chapter 5 presents the performance evaluation of the SDAR. It introduces simulation setup, defines performance metrics, demonstrates experimental results, and evaluates the performance of the SDAR protocol accordingly.
- Chapter 6 introduces our initial work on an agent-based trust and reputation management scheme (ATRM) for wireless sensor networks.
- Chapter 7 concludes the thesis and points out the future work.

Chapter 2

Wireless Ad-hoc Network Security

Network security issues have been deeply investigated on an ongoing basis. In traditional networks, the concept of security has been properly defined, for instance, [38] defines five basic security services that can be provided optionally within the framework of the OSI (Open Systems Interconnection) Reference Model[1]. In order to provide security services, cryptography schemes and authentication methods are widely employed in wired networks. Since all the attacks on wired networks can be applied to wireless ad-hoc networks, the two kind of networks share the same security goal, and naturally, the security-supporting techniques for wired networks can be employed in wireless ad-hoc networks. In this chapter, we introduce some typical types of attacks against wireless ad-hoc networks and discuss network security fundamentals. In particular, an important security property – anonymity, and anonymity-enabling schemes are emphasized separately. At the end, we talk about a motivational approach to secure networks.

[1]OSI Reference Model defines how two entities may communicate with each other. It is a standard networking framework composed of seven layers: the physical layer, the link layer, the network layer, the transport layer, the session layer, the presentation layer and the application layer. Routing protocols usually run at the network layer.

2.1 Attacks on Wireless Ad-hoc Networks

Because of the broadcast nature of wireless communication and the wire-free property of wireless ad-hoc networks, attackers can easily eavesdrop on wireless communication channels and/or interfere with network communication. In such a hostile environment, nodes and routing protocols are more susceptible to various attacks by hackers than in wired networks. Additionally, due to the movement of nodes and the absence of fixed infrastructure, wireless communication has to rely on the message relay of intermediate nodes that might however be selfish or malicious.

All of the known attacks in wired networks can be used against wireless ad-hoc networks, making wireless ad-hoc routing protocols unreliable. As in conventional networks, these attacks can be categorized, according to their nature, as *passive attacks* and *active attacks* [83]. In practice, the two types of attacks are usually combined together by attackers for various purposes (e.g., routing disruption and resource consumption). The following subsections first discuss the two types of attacks and then introduce some practical attack examples.

2.1.1 Passive attacks

A passive attack happens when an unauthorized attacker unintrusively taps onto a communication channel between two nodes without disturbing the communication. Its primary goal is not to interrupt the operation of the communication channel but to discover some valuable information from the data or control messages sent over the communication channel. Most commonly, passive attacks occurs in form of eavesdropping, where the attacker (eavesdropper) observes the traffic between senders and receivers. While passive attacks might not look harmful, they form a non-ignorable threat to the security and privacy of the system. If traffic is not cryptographically protected, passive attackers can easily capture confidential information such as passwords, configuration data, and system logs. Although data may be encrypted, the traffic analysis on clear control information, for example, routing information, can expose

nodes' identities, disclose the relationship between nodes, or reveal the topology of the network. For instance, if an eavesdropper finds out that a large volume of traffic is sent through a particular node, he can draw the conclusion that the node is playing an important role in the network, and he might later mount an active attack against that node, leading to a major disruption of the network.

2.1.2 Active attacks

An active attack typically involves an attacker's direct intervention with the data and/or control messages sent over the communication channel. Active attackers may adversely participate in routing protocols by deliberately replaying, inserting, modifying, or deleting routing packets. The active attacks on routing protocols can lead to malicious updates on nodes' route tables and result in packets being sent to false destinations, or they might induce the creation of routing loops and network congestion. Active attacks usually happen in the following forms or the combination thereof: *masquerade*, *replay*, *message insertion*, *message modification*, *message deletion*, and *Denial-of-Service* (DoS).

- *Masquerade*

  In a *masquerade attack*, the attacker impersonates certain legal users to illegally gain some services that are not supposed to be accessed by the attacker. For example, during a route discovery process, the attacker might deceive the sender by replying the route request using the destination's identity to obtain confidential information such as passwords from the sender.

- *Replay*

  In a *replay attack*, the attacker intercepts some valid messages, records them, and resends them later to the original receiver. For example, an attacker may adversely rebroadcast an old valid route advertisement and then mislead its neighbors in updating their routing tables with stale routing information.

- *Message Insertion*

  In a *message insertion* attack, the attacker forges messages (usually with fake source addresses) and injects them into the network. In most cases, this attack is used as a part of another attack such as a *blackhole* attack.

- *Message Modification*

  In a *message modification* attack, the attacker removes a message from network traffic, alters it adversely, and then re-sends it. For instance, the attacker maliciously modifies the routing metric field of every route request message he receives before re-transmitting them so that his neighbors are most likely to route packets through him, and later, he can performs further attacks such as a *man-in-the-middle* attack.

- *Message Deletion*

  In a *message deletion* attack, the attacker simply removes messages from network traffic. For example, the attacker drops all the messages going through him from a certain source node, causing a communication failure.

- *Denial-of-Service (DoS)*

  In a *Denial-of-Service* attack, the attacker attempts to stop the normal use of network resources by overconsuming them. For example, an attacker may frequently flood a network with messages to consume network bandwidth and thereby prevents legitimate network traffic.

2.1.3 Attacks in Practice

In practice, attacks are often the combinations of the basic types of attacks introduced above. Based on their purposes, practical attacks can be classified as *resource consumption attacks* or *routing disruption attacks*. The objective of resource consumption attacks is to stop the normal use of network resources by overconsuming them, as is the case with DoS attacks. Routing disruption attacks aim at disrupting the normal operation of routing protocols

by adversely dropping, modifying routing packets, and fraudulently disseminating incorrect routing information. In what follows, we introduce several typical routing disruption attacks.

- *Blackhole Attacks*

  In a *blackhole* attack, the attacker provides false routing metrics to all possible destinations such that all its neighbors route packets through it, and then, it discards all the packets it receives. The blackhold attack creates a black hole in the network, which attracts and drops packets. A special case of blackhole attack is called *grayhole attack.* In a grayhole attack, the attacker selectively discards packets, for example, dropping all the data packets except the routing packets.

- *Man-in-the-middle Attacks*

  In a *man-in-the-middle* attack, the attacker is required to be present on the message path linking the sender to the receiver. The attacker monitors all the messages exchanged between the two communicating nodes to obtain any unprotected information, and it may also perform other attacks by impersonating one communicating node while talking with the other.

- *Wormhole Attacks*

  The execution of a *wormhole* attack requires the collusion of malicious nodes. In a wormhole attack, the attacker records a message at one location in the network, tunnels it to another location, and relays it with the help from the colluding adverse node/s. Since a wormhole attack misleads the sender and the receiver to incorrect knowledge about their distance, discovering routes more than one or two hops long always fail.

- *Rushing Attacks*

  A *rushing* attack is harmful only when used against the on-demand routing protocols with duplicate suppression mechanisms at each node. In this attack, the attacker broadcasts a forged route request message which makes the legitimate one like a duplicate. If all the neighbors of the destination receive the forged route request from the attacker

first, they will discard the genuine one later. Rushing attacks can cause route discovery failure.

2.2 Network Security Primitives

In a wireless ad-hoc network, successful attacks on the routing protocol put the entire network at risk. Robust wireless ad-hoc routing protocols must provide the five basic security services defined in [38]. These services are *data confidentiality*, *data integrity*, *authentication*, *non-repudiation* and *access control*. In practice, these services may be provided optionally or in combination with particular security mechanisms according to specific requirements. We describe these five basic security services as follows:

- *Data Confidentiality*

  This is the property in which the information embedded in network traffic is prevented from unauthorized disclosure. Since one of the main reasons that an attacker can successfully attack network nodes and protocols is due to the leak of sensitive information such as passwords and configuration data, data confidentiality is a very important property of network security.

- *Data Integrity*

  This is the property in which the originalness of the information transmitted over the network is ensured. It is often combined with *data origin authentication* since data integrity alone can not help receivers decide whether the received data are forged or have been tampered with.

- *Authentication*

  This is the property in which the identity of the connected entity (node) can be confirmed during connection phase (peer entity authentication), and the source of the message transmitted during the data transfer phase can be verified (data origin authentication).

- *Non-repudiation*

  This is the property in which communication participants' denial of the existence of their involvement in communication is prevented. Non-repudiation together with proper evidence can prevent senders from attempts to disavow having sent a message, and prohibit receivers' intentions of falsely denying having received a message.

- *Access Control*

  This is the property in which accessible resources, including information, programs, storage space, CPU cycle and communicating devices to name a few, are protected against unauthorized use over communication channels.

2.2.1 Cryptography Schemes

In order for networks to provide the above fundamental security services, the use of cryptography is a must [17]. In any cryptography scheme, there are two types of operations involved, *encryption* and *decryption*. Encryption is the process of transforming plaintext, readable and comprehensible text, into an encoded and unreadable gibberish – ciphertext. The inverse transformation, the process of inverting ciphertext to plaintext, is referred to as decryption. Encryption is used to disguise the real substance of data from unauthorized readers, and thus, it protects data confidentiality. Cryptographic algorithms themselves are public, but the secret parameters (encryption/decryption keys) that they use are known only to the intended ones (senders and/or receivers); based on whether encryption keys are the same as decryption keys, cryptography schemes can be classified as either symmetric or asymmetric [17].

Symmetric Cryptography

Symmetric cryptography schemes use the same secret key for both encryption and decryption [17]. Thus, each entity maintains only one key. For any pair of communicating entities,

they should share a secret key beforehand in order to communicate; if a group of more than two entities wants to communicate, the entities have to share the same secret key.

There exists basically two types of symmetric cryptographic algorithms, including block ciphers (e.g., Data Encryption Standard (DES) [24] and International Data Encryption Algorithm (IDEA)) and stream ciphers (e.g., Software-optimized Encryption Algorithm (SEAL) [71] and RC4 [49]). Block cipher algorithms encrypt/decrypt data in multiple rounds; in each round the algorithms operate on a fixed block of bits, while stream cipher algorithms perform encryption and decryption bit by bit.

The main advantage of symmetric cryptography is its fast encryption and decryption operation. However, this cryptography scheme suffers from the complexity of key management. In a hostile network environment like wireless ad-hoc networks, secret keys can not be distributed simply over insecure communication channels. Instead, they are most likely to be disseminated offline or with the help from an asymmetric cryptography scheme.

Asymmetric Cryptography

Asymmetric cryptography, also known as public key cryptography [17], is the most recent cryptographic tool that is becoming increasingly popular. The primary point that makes asymmetric cryptography different from its symmetric counterpart is that the key used to decrypt ciphertext is different from the key used to encrypt plaintext. Each entity, a network node, has to maintain two keys in an asymmetric cryptography scheme. One, called the *public key*, is available to anyone who needs it, while the other, called the *private key*, is kept secret by the entity itself. If an entity A wants to communicate with another entity B in a secret manner, it just encrypts the messages for B with B's public key. Because B's private key is owned and managed only by B itself, B is the only one who is able to successfully decrypt the encoded messages from A and read the contents. The most widely employed asymmetric cryptography is the Rivest Shamir Adleman (RSA) algorithm which is based on the difficulty of performing the factorization of a large number – the product of two primes [70].

The main strength of asymmetric cryptography is its ease of key management. As introduced above, only public keys need to be distributed. The communication channel for public key distribution however is not necessarily secret. Actually, an authentic channel is adequately effective; whereas, the drawback of asymmetric cryptography is that it needs a relatively larger amount of computation power and time compared to symmetric cryptography. In practice, a hybrid approach combining both symmetric and asymmetric cryptography is often employed. That is, an asymmetric cryptography scheme is used to distribute the secret key of a symmetric cryptography scheme which is used for the actual data encryption/decryption.

2.2.2 Authentication Methods

When two nodes communicate in an insecure network like wireless ad-hoc networks, having only data confidentiality is not sufficient. It is necessary for the receiver to be able to verify that the received message is identical with the one that is originated from the sender and that the message sender is the same as it claims to be. In other words, authentication is indispensable for securing network communication. Cryptography can actually be used to support authentication [17]. Depending on the trust relationship between communicating nodes, authentication can be performed through different cryptography schemes. If there is mutual trust between senders and receivers (i.e., sharing a secret), symmetric authentication based on symmetric cryptography can be used. In the case of missing trust relationship, asymmetric authentication based on asymmetric cryptography may be applied. The representative symmetric authentication tool is a *one-way hash function*, while a typical asymmetric authentication tool is a *digital signature* [17].

One-way Hash Functions

A one-way hash function is a transformation algorithm, such as MD5 [69] and SHA-1 [19], which takes a variable-sized message as input and returns a fixed-sized digest, referred to as "hash value". For a one-way hash function, it is hard to invert a hash value back to the

original input message. In addition, hash values can be considered as "digital fingerprints" of the corresponding input messages because it is computationally infeasible to find two different input messages that lead to the same hash value. Due to the one-way and collision-free properties, one-way hash functions can be used for data authentication and integrity checks. Since hash functions are public, they should be used only in the case that entities trust each other. For example, the sender encrypts the data with a secret key, hashes the encoded data, attaches the hash value [Message Authentication Code (MAC)] to the original data, and then sends the original data together with the hash value to the receiver [21]. After the receiver receives the message, it encrypts the data with the same secret key, hashes the result, and compares the hash value with the MAC contained in the message. If they are equal, the receiver can be reassured that the data was actually sent from the sender and has not been altered in transit.

Digital Signatures

A digital signature is a "stamp" placed by the sender on the data to be transmitted. It is unique to the sender and is difficult for others to forge. Also, any modification to the signed data is detectable. In this sense, a digital signature is the equivalent of a signature made by hand on a paper document. Asymmetric cryptography can be used for generating digital signatures [17]. In practice, the data to be transmitted is first hashed, and the hash value is then encrypted with the sender's private key. The resulting encoded data is the sender's digital signature for the data. After the receiver gets the signed data, it hashes the data, decrypts the signature with the sender's public key, and checks whether the hash value is equal to the decrypted signature. If they are not equal, the receiver can conclude that either the signature or the data was illegitimately modified. Because the private key is owned only by the sender itself, nobody can forge the sender's digital signature, and the sender can not falsely deny having sent the data. In this case, we can see that the digital signature actually provides authentication, data integrity and non-repudiation simultaneously. Available signature schemes include RSA [70], Digital Signature Algorithm (DSA) [23], and the Fiat-Shamir scheme [22].

2.2.3 Asymmetric Cryptography in Practice

While writing asymmetric cryptography is simple and straightforward, merely implementing an asymmetric cryptographic algorithm is insufficient. In reality, the implementation of asymmetric cryptography needs to define a set of supporting components to perform key (certificate) creation, distribution and revocation. The following subsections briefly discuss two different kinds of asymmetric cryptography implementations: Public Key Infrastructure (PKI) [82] and Pretty Good Privacy (PGP) [90].

However, before proceeding further, we would like to first introduce a commonly used concept by PKI and PGP, called a *digital certificate*. In the two systems, public keys are distributed and used in form of a digital certificate. A digital certificate, in short certificate, is just a special data structure which defines a binding between a particular user's ID and its public key. Other than a user ID and a public key, it may also contain other information such as issuance date, expiry date, issuer's signature and so on. The current industry standard for digital certificates is the CCITT X.509 international standard [35].

Public Key Infrastructure (PKI)

PKI [82] is a framework supporting public cryptography. It generally consists of the following components: security policy, certificate authority (CA), registration authority (RA) and directory service. Security policy specifies how the PKI operates, e.g., procedures for key generation, issuance, storage, and revocation. A CA is a trusted third party responsible for creating, issuing and revoking certificates. A RA is a trusted third party, e.g., Notary Public, which verifies that a certificate applicant is truly the one it claims to be. A directory service mainly helps users find other users' certificates.

In PKI, a CA may issue certificates not only to end users but also to other CAs. The certificate of a CA is used by that CA to sign the certificates that it issues. Since digital signature can not be forged, the trust in a CA can be derived from the trust in the higher-level CA that issues a certificate to that CA. All the CAs together constitute a hierarchical tree-like

structure of trust. The root CA/CAs, in the highest level, is/are the base of this hierarchy of trust and is/are known to every other CA as well as to each end user.

When a user creates a (PublicKey, PrivateKey) pair, it submits the pair to a CA for certification. After certification, the user gets the resulting certificate from the CA and then registers it with a directory service. If user A wants to send a message secretly to another user B, it should first obtain B's certificate via the directory service. In the case that both A and B are certified by the same CA, the directory server just sends B's certificate to A which then uses the common CA's public key to validate it. If A and B are certified by different CAs, the directory server will create a *certification path* from A to B, which has the form $CA_1 \overset{cert_1}{\rightarrow} CA_2 \overset{cert_2}{\rightarrow} \cdots \overset{cert_{n-1}}{\rightarrow} CA_n \overset{cert_n}{\rightarrow} B$ where $1 \leq i < n$, $cert_i$ is the CA_{i+1}'s certificate signed by CA_i, and $cert_n$ is B's certificate. If CA_1 is known to A, A will be able to find the certificate of B along this path. Clearly, one of best candidates of CA_1 will be a root CA.

Certificate Revocation Lists (CRLs) are used to publish the information that signifies which certificates become invalid. They exist outside the directory service and may be available to everybody even when the directory service is absent.

Pretty Good Privacy (PGP)

PGP [90] was designed originally by Phil Zimmermann in 1991. It is a public key encryption program used primarily for protecting email on Internet. It allows mutually-unknown users to be able to communicate securely over an insecure channel.

In contrast with the PKI [82], PGP does not distinguish between certificate authorities (CAs) and regular nodes. Each node must create a (PublicKey, PrivateKey) pair associated with its unique ID for itself and disseminates its own public key. If user A has a copy of the public key of another user B and believes that the copy has not been tampered with, it can tell a third user C that it believes in the originalness of that copy by signing and passing the copy to C. The singed copy of the public key of B is a certificate, and A actually acts as the introducer of B to C. In PGP, each user is required to tell the system who it trusts as introducer and also

to certify the introducers' public keys with its own private key. It may additionally specify the degree of trust that it has in each introducer. As a result, a node can validate a unknown node's public key through aggregating the trust degree of the corresponding introducers, and a web of trust is consequently established.

In PGP, when A wishes to send an email to B, it generates a session key and encrypts the email with that session key. In order for B to be able to read the email, A encrypts the session key with B's public key and sends it together with the email to B. After B receives the email from A, it retrieves the session key with its private key and then decrypts the email.

When a user's private key is compromised, the user widely disseminates a revocation certificate to claim this fact, and as a result, its private key is marked as compromised. The generation of a revocation certificate requires private keys.

2.3 Anonymity

Anonymity is another important property of network security [62]. Compared to the other security properties, it is much more difficult to achieve. There are four types of anonymity protection:

- *Sender anonymity*

  Sender anonymity hides the identities of message senders. Any node can not tell from whom the message is originated.

- *Receiver anonymity*

  Receiver anonymity protects the identities of message receivers. Any node that see a message can not tell to whom the message is designated.

- *Route anonymity*

  Route anonymity hides routing information. No one can identify the route connecting two communicating nodes by tracing traffic flows.

- *Unlinkability*

  Unlinkability protects the communication relationship between two nodes. Although a sender and a receiver can be identified as participating in some communication, their connection is still protected. That is, no one can tell which node a node is talking to.

For the Internet, there are already several well-known anonymity schemes including DC-nets [15], MIX-nets [14], Onion Routing [66] and Crowds [67]. These schemes, especially MIX-nets and Onion Routing, often serve as the basis of other anonymous systems. For example, anonymous systems [29, 78] use application level routing to provide anonymity through a fixed core set of MIXes. Each host keeps a global view of the network topology, and makes anonymous connections through a sequence of MIXes instead of making direct socket connections to other hosts. The authors in [44] used an alternate Onion Routing approach to provide anonymous communication for mobile agents in the Java Adaptive Dynamic Environment (JADE) [56]. Each JADE multi-agent has several onion agents that provide an anonymous data forwarding service, and at least one onion monitor agent that keeps track of the location of all other onion agents in the system. Onion monitor agents exchange onion agent reachability information in order to maintain a valid topology of the complete onion agent network.

Indeed, there are many variations of MIX-based networks. But, in order to give a clear and comprehensive view of available basic anonymity-enabling mechanisms, we are going to introduce only the basic DC-nets, MIX-nets, Onion Routing and Crowds in the following subsections.

2.3.1 DC-nets

DC-nets [15] are based on the DC (Dining Cryptographer) problem. The DC problem can be expressed as follows: find a solution which enables one of the three cryptographers to transmit a bit '1' in such a way that all the three get it but no one (except the transmitter) can tell from whom it was sent. To solve this problem, we just arrange the three cryptographers on a circle, and let each of them flip a two-side coin which can be seen only by the cryptographer

himself and his right-hand-side neighbor. We require that each cryptographer calculate the XOR of his coin and the coin of his left-hand-side neighbor, and we also require that the one wishing to transmit the bit broadcast the converse of his XOR while the others broadcast just their XOR. If an odd number of 1's were distributed, then it can be known that a bit '1' was transmitted. To allow more than three participants transmitting longer messages than one bit, the above protocol can be performed in rounds, and each pair of participants shares a chain of secret bits, one bit for each round.

In DC-nets, both sender anonymity and receiver anonymity are maintained, but nodes are vulnerable to message modification attack and DoS attack. In every round of transmission, all the participants are involved, and for every bit transmitted, there are two extra bits that are passed around the circle, therefore, the overhead is quite large.

2.3.2 MIX-nets

The basic building block of every MIX-net [14] is a MIX, which is a special node existing between senders and receivers. It delays, reorders, pads to constant size, or scrambles messages in order to confuse traffic analyzers. Each MIX node is pre-assigned a (PublicKey, PrivateKey) pair, and its public key is available to every other node. All the MIXes constitute certain topology which is known to all the nodes in the network.

When a sender wants to communicate with a receiver, it first choses a route through a sequence of MIXes to the receiver. Then, the sender adds a padding to the original message and encrypts them together with the receiver's public key. Afterwards, the sender encrypts the encoded message with the public key of each MIX recursively in the reverse order of the MIXes' appearance in the selected route to the receiver. To resist traffic analysis at each MIX in the route, paddings are also appended to the message each time when the message is encrypted. Finally, the multi-layer encrypted message is sent to the first MIX which in turn decrypts the message with its private key, removes the padding, and forwards the one-layer-less message to the next MIX along the route (if the next MIX is not pre-known, the message

should define it). Every intermediate MIX processes the message in the same way. After the last MIX finishes processing the message, it sends the message encrypted only with the receiver's public key to the receiver.

MIX-nets support sender anonymity and resist traffic analysis (note that there is however no sender anonymity from the first MIX). MIX-nets are vulnerable to the collusion by the first MIX and the last MIX in a route. In addition, Chaum MIXes requires public key cryptography, so MIX-nets are computationally expensive and slow for message transmission.

2.3.3 Onion Routing

Onion Routing [66] is an extension to MIX-nets [14]. Its primary objective is not to hide senders and receivers from each other but to protect their communication from others. In Onion Routing, MIXes are called onion routers, and they together form a onion router network. Similar to MIX-nets, Onion Routing requires pre-defined network topology.

To setup an anonymous path to the receiver, the sender makes a connection to an onion router which then builds a chain of onion routers using its pre-knowledge of the network topology and available onion routers. To protect data and routing information, the first onion router constructs a multi-layer encrypted data structure called an onion and sends it through the network. Each layer of the onion defines the next hop in the route. An onion router that receives an onion peels off the topmost layer of the onion, identifies the next hop, and sends the remaining onion to the next onion router. In addition to carrying next hop information, each onion layer contains key seed material from which keys are generated for later decrypting and encrypting data sent forward or backward along the anonymous connection. Once the anonymous connection is established, the data can be transfered in a similar multi-layer encryption way in both directions. When communication ends, the anonymous connection is torn down. This involves the removal of encoded next hop information in each onion router along the path. In order to reduce overhead in practice, Onion Routing uses asymmetric cryptography for establishing communication channels and symmetric cryptography for transmitting data.

Because communicating nodes' identity and communication content are both hidden, and because each onion router in an anonymous connection can only identify its immediate onion router neighbors, Onion Routing resists traffic analysis, eavesdropping, and other attacks from both outside and inside of the onion router network. However, compromised onion routers may still be able to reveal routing information under collusion.

2.3.4 Crowds

Crowds [67] is developed for private web browsing. In a Crowds system, nodes are grouped into a number of crowds, each of which issues requests to web servers on behalf of its members, and then, sender anonymity is protected by mixing one's action within others'.

For any node n_0 in a crowd, when it wishes to issue a request to a web server, it does not submit the request directly to the server but to a randomly selected member of the crowd, say n_1. After n_1 receives the request from n_0, it decides whether to send the request directly to the web server or to another randomly chosen member. For each intermediate node, it processes the request in the same way and remembers its prior and next hop for later transmitting traffic backward and forward. Using this approach, a request message will travel along a path consisting of a sequence of crowd members, $n_0, n_1, \cdots, n_i$, and will be finally submitted by node n_i directly to the web server. Once this path is established, it is used for n_0 to communicate with the server, and the subsequent traffic between n_0 and the server will be sent out of and received into the crowd always by node n_i. In the case, sender anonymity is protected since the submitter n_i is just a random member of the crowd, and even the collusion of some crowd members cannot tell who is the originator since n_0 may also be a node just forwarding the traffic.

Clearly, there is no receiver anonymity in Crowds. From web servers' view, it is not possible to identify the originator of a request, so sender anonymity is accomplished. However, by eavesdropping on nodes' in-coming and out-going traffic within a crowd, an attacker can find who is the sender, and therefore, there is actually no sender anonymity from local eaves-

dropper. In addition, Crowds also has other drawbacks, such as vulnerability to internal DoS attacks.

2.4 Incentive Mechanisms

A wireless ad-hoc network is a public autonomous system. In such an environment, we can not expect that all the network nodes behave as the cooperative requirements of network operation. That is, node misbehaviors are ineluctable. The reasons why nodes misbehave can be multifold. For instance, some nodes may be devised to act incorrectly; some may be compromised to do so. Some nodes probably perform improperly just by accident; and others could violate rules for economic purposes. Besides, proper nodes may occasionally appear to be misbehaving because of the misleading complex network situations such as network congestion and link breaking. If we ignore occasional mistakes and misunderstandings, node misbehaviors can actually be categorized as *attack behaviors* and *selfish behaviors*. The attack behaviors are what we discussed earlier in Section 2.1, and they are performed by attackers for disruptive purposes. Whereas, selfish behaviors are conducted by nodes for profits rather than disruption. As we know, the operation of wireless ad-hoc networks relies heavily on the collaboration of nodes because of the lack of fixed infrastructure and centralized administration. Therefore, nodes' selfish behaviors can break network operation unintentionally. Unfortunately, the wireless nodes with resource constraints are liable to show selfishness for resource saving purposes, e.g., saving battery power, CPU cycle, storage space, bandwidth or processing time. In this case, it is necessary for wireless ad-hoc networks to have an effective mechanism to encourage node collaboration, to identify and to isolate selfish (or disoperative) nodes. In the following subsections, we will introduce two main kinds of such incentive mechanisms, *pricing* and *reputation* [77].

2.4.1 Pricing

The idea behind the pricing scheme is quite simple and is inspired by a real-life model. Considering human society, people work for (or provide services to) others and are paid for their contribution. The money people earn from work can be used to buy products or services from others at any time, at any location. The money people make in different areas is comparable and exchangeable.

In computer networks, nodes pay digital money for the service they get from other nodes. After a node earns sufficient amount of digital money, it can buy services from other nodes with the money. Clearly, in a hostile environment where nodes do not trust each other, it is necessary that a service provider will offer a certain service to a service applicant only when the service applicant sufficiently pays in advance; and it is also necessary that the service provider is going to be paid only when the service applicant successfully gets the service it wants. This is quite a dilemma. Besides, it is obvious that the amount of money a node has can not be infinite, and a node should honestly pay the amount of money it is supposed to pay. In these cases, accounting becomes a central problem for the pricing scheme. A solution to this is using a trusted third party such as a bank to manage nodes' accounts. To do so, the bank should be able to monitor the trading between nodes. Using this apporach, and with the mutual trust between the central authorities of different environments as well as the proper exchange currency policy, a node in a certain environment may move to another environment to consume money (to request services unavailable in its home environment).

In the pricing scheme, nodes willing to help others can accumulate wealth and in turn gain better service from others, while selfish or malicious node will not earn adequate money because of their frequent denial of service and/or failed service delivery such that they are not able to get expensive services. If some nodes keep performing selfishly or maliciously, they will be isolated from the network after their money has run out. We can see that the pricing scheme can solve the motivation problem. However, the pricing scheme may not be the best solution for wireless ad-hoc networks, since it requires a centralized authority for accounting.

2.4.2 Reputation

In a distributed environment, an entity's *reputation* can be considered the global perception about the entity's future behavior based on the *trust* that other entities hold in the entity. Therefore, we should first clarify the concept of trust to properly express the substance of reputation.

Trust reflects the degree of confidence an entity holds in other entities about their future behavior. It is based on the entity's past experience with and observation of other entities' actions. In computer science, trust can be quantified as an accumulative value. If a node behaves as it is supposed to, its trust will be added to a positive value, or a negative value otherwise. Therefore, a node's trust may change in either direction over time. Since a node is evaluated independently by other nodes, the participating nodes might not necessarily trust it equally. In addition, nodes can have different trust degrees in a node under different contexts. Based on the above description, we can say that trust is temporal, dynamic, non-monotonic, subjective, non-transitive and contextual. In addition, we can derive from subjectivity and non-transitivity properties another characteristic of trust, that is unidirection.

A node's reputation is somewhat like the average of the trust that all the other nodes hold in that node. Because reputation reflects a "global" perception, it is objective rather than subjective. In this case, a well-behaving node may have a higher reputation level when compared to other nodes. Whereas, the misbehaviors of selfish/malicious nodes will definitely affect their reputations. Since trust serves as the basis of reputation, reputation has all the properties that trust has except subjectivity. In a reputation system, the nodes with high reputations will be rewarded by granting their service requests, while the service requests of the nodes with low reputations are very likely to be rejected due to their unsatisfactory or even bad history. In order to survive (to obtain necessary services), a node should try its best to offer good services to other nodes.

However, there are a number of difficulties in reputation systems. Firstly, since a node's trust information is distributed among other nodes, the problem of trust collection need to be

efficiently solved. Secondly, an accurate reputation model is required to truly reflect nodes' trustworthiness based on the collected trust information. Finally, it is crucial to prevent malicious/selfish nodes from refreshing their reputation by changing their identification from time to time. So far, a number of distributed trust/reputation systems, including [42, 51, 63, 85, 48] to mention just a few, have been proposed, and simulation results show that rating nodes' trust and reputation is indeed an effective approach to motivate node collaboration, to support decision making, and to enhance security. In Section 3.2.6, a reputation-based secure routing protocol, CONFIDANT [12], is introduced, and in Chapter 6, trust and reputation management is discussed in more detail.

Chapter 3

Wireless Ad-hoc Routing

Efficient routing is a primary challenge in wireless ad-hoc networks and has been long attracting research attention. The early efforts on wireless ad-hoc routing focused mainly on developing effective and efficient routing algorithms, and yielded some well-known routing protocols, e.g., [40, 54, 60, 61]. In recent years, with people's increasing awareness of the importance of security, considerable research are conducted on secure wireless ad-hoc routing, and some (but not so many) researchers even start to work on much strongly secure routing schemes – anonymous routing, in wireless ad-hoc networks. In this chapter, we will introduce some existing wireless ad-hoc routing protocols.

3.1 Basic Wireless Ad-hoc Routing Protocols

The primary concern in all of these wireless ad-hoc routing protocols are effectiveness and efficiency. When these protocols were developed, the developers did not keep security in mind. Hence, they are vulnerable to all kinds of attacks. Based on the employed route discovery methods, these protocols can be generally categorized as *table-drivens (proactives)* or *on-demands (reactives)*. Table-driven routing protocols derive from legacy distance-vector or link-state routing algorithms. They require that every node maintain a global view of the network topology by managing one or more routing tables. To have a consistent and up-to-date network view, nodes periodically respond to the changes in network topology by advertising route updates throughout the network. The main advantage of this kind of routing protocols is the fast route acquisition, while their apparent drawback is the large amount of overhead caused by the periodical route advertisements. In on-demand routing algorithms, nodes do not necessarily know about network topology, and route discovery is triggered only when necessary. When a node wishes to build a route to a destination, it initiates a route discovery process by broadcasting a route request message throughout the network. This process terminates once a route is established or after all possible candidate routes have been examined. Compared to the table-driven approaches, this kind of routing protocols have less overhead since they do not use periodical route update advertisements, but they incur relatively long route acquisition latency. In the following subsections, we introduce some of the existing table-driven and on-demand wireless ad-hoc routing protocols. A detailed comparison between these routing protocols based on performance evaluation can be found in [9].

3.1.1 Wireless Routing Protocol

Wireless Routing Protocol (WRP) [54] is a table-driven routing protocol. Its main novelty is the way of achieving loop-free property. Each node using the protocol checks the consistency of predecessor information reported by all its neighbors whenever it processes an event

involving one of its neighbors.

In WRP, every node maintains a distance table, a routing table, a link-cost table and a message retransmission list (MRL). Each entry of the MRL consists of the sequence number of an update message, a retransmission counter, an ACK-required flag, and a list of updates sent in the update message. The MRL indicates which updates of an update message have to be retransmitted and which neighbors should make a confirmation to such retransmission. Nodes keep track of their neighbors by receiving acknowledgments and listening to a HELLO message periodically broadcasted by each node. When a node receives a HELLO message from a newcoming node, it first adds the node to its routing table and then sends the node a copy of its routing table.

Nodes notify each other of link changes through route-table update messages exchanged only between neighboring nodes. An update message contains sender's ID, a sequence number originated by the sender, an update list of updates or ACKs, and a response list of the nodes supposed to acknowledge the update message. If an entry of the update list indicates an update, it is a tuple of the destination's ID, the distance to the destination, and the predecessor of the destination. If an entry of the update list stands for an ACK, it is a tuple of the source ID and the sequence number of the update message being acknowledged. A node updates its routing table according to the received update messages from its neighbors or the detected changes in the links to its neighbors, and it sends new update messages to its neighbors afterwards.

For each update entry in the update list of the update message, a receiving node modifies its distance table entries that are affected by the update entry and then checks for new possible routes through other nodes. Any new routes will be sent back to the sending node to update its tables. For each ACK entry in the update list of the update message, the receiving node scans its MRL to find the the sequence number matching the sequence number specified in the ACK entry to reset the corresponding ACK-required flag for the sending node.

3.1.2 Destination-Sequenced Distance-Vector Routing

Destination-Sequenced Distance-Vector Routing (DSDV) [60] is a table-driven routing protocol, which is based on the classical Bellman-Ford routing algorithm [25]. The improvement made to the Bellman-Ford routing algorithm is the application of sequence number. By using sequence number, stale routes are distinguished from fresh ones, and routing loops are prevented.

In DSDV, each node maintains a routing table listing all the possible destinations within the network, the shortest known distance, in number of hops, to each destination, and the first hop in the shortest route to each destination. Every routing table entry is marked with a sequence number originated by the corresponding destination. All the nodes periodically perform a route broadcast. Each route broadcast contains the destination ID (which actually indicates the originator node), the number of hops to reach the destination, the sequence number of the information received regarding the destination, and a new sequence number unique to the broadcast. The route with the most recent sequence number is always used. If more than one route update has the same sequence number, the one with smallest number of hops is used. Nodes also trace the number of time units between the arrival of the first and the best route to each particular destination. Based on these data, nodes decide whether to delay advertising unstabilized routes to reduce traffic and to damp fluctuation of route tables.

To keep routing tables consistent in a network with dynamically changing topology, each node is required to periodically broadcast its own route table to all its current neighbors. To help reduce the potentially large amount of network traffic caused by such updates, two type of packets, *full dump* and *incremental*, are employed. The former carries all the available routing information and usually requires multiple network protocol data unit (NPDU), while the latter contains only information changed since the last full dump and fits into one NPDU. When nodes' movement is infrequent, incremental packets rather than full dump packets are used to propagate differential routing information. If nodes' movement becomes frequent, the size of an incremental packet will be as big as that of a NPDU, and then a full dump can be scheduled

so that the next incremental packet will be smaller again.

3.1.3 Ad-hoc On-demand Distance Vector Routing

Ad-hoc On-demand Distance Vector Routing (AODV) [61] is classified as a pure on-demand routing protocol. It ensures loop-free property by using destination sequence numbers. In AODV, every node maintains a routing table, each entry of which corresponds to a particular route to certain destination. A route table entry is a tuple of destination ID, next hop, number of hops to reach the destination, destination sequence number, active neighbors, and route expiration time.

When a source node wants to discover a route to a destination node, it locally broadcasts a route request (RREQ) packet, carrying its own sequence number and the most recent sequence number it has for the destination. When an intermediate node receives the RREQ for the first time, it records the neighbor from which it receives the RREQ in its route table such that the reverse route is established. Note that the destination sequence number of the corresponding route entry will be set to the source sequence number in the RREQ packet to ensure reverse route refreshness. Afterwards, the intermediate node forwards the RREQ to its neighbors. If an intermediate node receives a RREQ it already received, it simply discards that RREQ.

Once the RREQ packet reaches either the destination itself or an intermediate node with an active route to the destination, which is associated with a sequence number equal to or greater than the destination sequence number in the RREQ packet, the destination/intermediate node sends a route reply (RREP) message along the reverse route back to the source. The most recent destination sequence number is included in the RREP packet to ensure the refreshness of the forward route to be established. Every intermediate node in the reverse route first records the neighbor from which it receives the RREP in its route table such that the forward route is established, and afterwards, it increases the hop count in the RREP packet and forwards the modified RREP packet to the next hop in the reverse route. When the RREP reaches the source, a route to the destination is successfully constructed.

A node revokes all of the never-used routes in its routing table by removing their corresponding entries. If a node does not broadcast any packets to its neighbors within a preconfigured period of time, it will actively broadcast a HELLO message to all its neighbors to announce its presence. Using broadcast messages, including HELLO messages, a node is able to have a view on its local connectivity. When a node detects a failure in the link to the next hop towards certain destination, it sends a special RREP along the reverse route to the corresponding source. All the intermediate nodes will be informed of the link failure while processing the special RREP. After the source is notified of the failure, it may re-initiate a route discovery for the destination if necessary.

3.1.4 Dynamic Source Routing

Dynamic Source Routing (DSR) [40] is a on-demand routing protocol. It consists of two main phases, route discovery phase and route maintenance phase. In DSR, each node maintains a route cache which stores the routes to some destinations. A node can cache routes either through actively originating or participating in route discovery or by passively overhearing routing information from others.

When a source node wishes to send a packet to certain destination, it first checks if it already has a route to the destination in its route cache. If so, the source just sends the packet using that route; otherwise, it initiates a route discovery phase by locally broadcasting a route request packet, which contains the IDs of both the source and the destination as well as a unique identification number. Each node receiving the route request packet checks its own route cache to see if it knows about a route to the destination. If not, the node appends its ID to the route request packet and forwards the modified packet to its neighbors. Once the route request packet reaches either the destination itself or an intermediate node whose route cache contains a valid route to the destination, the destination/intermediate node sends a route reply packet back to the source.

Since the route request packet contains the IDs of all the intermediate nodes along the

route it ever traveled, either the destination or the intermediate node is able to generate a route reply packet carrying the information of the complete route from the source to the destination. In order to send the source a route reply packet, the destination/intermediate node, obviously, must has a route to it. If the links between nodes are bidirectional, the route reply packet can be sent back to the source simply along the reverse route; otherwise, a new route discovery process for the source node will be initiated by the the destination/intermediate, and the route reply packet will be piggybacked on the corresponding route request packet. After the source receives the route request packet, it can retrieve the encapsulated route reply packet.

To avoid loop, a node forwards the same route request packet only once. In case that an intermediate node initiates route reply, the route will be the aggregation of the route cached by the intermediate node to the destination and the route carried by the route request packet, and the intermediate node removes any loop appearing in the aggregated route before it sends the route reply packet.

Route maintenance is accomplished through the use of route error packets and acknowledgments. If a node did not receive any confirmation, e.g., an active or passive acknowledgment, after re-sending a packet up to a maximum number of times, the node sends a route error packet to the source to notify it of the failing link. When a node receives a route error message, it removes the hop in error from its route cache and cut off all of the routes containing that hop at that point.

3.2 Secure Wireless Ad-hoc Routing Protocols

In wireless ad-hoc networks, nodes are supposed to cooperate with each other to form an operational network without any fixed infrastructure such as base stations and access points. Communication is expectedly based on the collaboration of nodes in a message relay fashion. However, wireless ad-hoc networks are not such non-hostile environments where nodes are willing to help each other. Instead, adverse nodes can freely enter the networks, listen network

traffic, compromise network nodes, and make a wreck of network operation; selfish nodes may not offer help to other nodes and then possibly cause various failures. Attacks on unprotected routing protocols can disrupt network performance and reliability, and even worse, the routing protocols with no security consideration might even be used by attackers to assault the network itself. Hence, wireless ad-hoc networks are in a great need of secure routing protocols.

A number of protocols have been developed to add security features to routing in wireless ad-hoc networks. Based on DSDV [60], Hu, Johnson and Perrig [36] proposed the SEAD, which is known for its addition of one-way hash chain for authentication on route update to the original DSDV. To enhance the security performance of AODV [61], Venkatraman and Agrawal [84] proposed an approach based on public key cryptography, which introduces two systems, EAPS (External Attack Prevention System) and IADCS (Internal Attack Detection and Correction System) respectively for preventing external and internal attacks; Sanzgiri and his colleagues [74] proposed ARAN where each intermediate node running the protocol verifies the integrity of the received message before forwarding it to its neighbor nodes, and source and destination nodes use the certificates included in the route discovery and reply messages to authenticate each other; Zapata and Asokan [88] proposed SAODV where digital signature technique is used to secure the non-mutable fields of routing packets while one-way hash chains are employed to authenticate the mutable field. To secure DSR [40], Papadimitriou and Haas [57] proposed SRP which assumes the existence of a security association between the source and the destination to validate the integrity of a discovered route; and Hu, Perrig and Johnson [37] proposed Ariadne which can protect routing packets through per-hop hashing mechanism and authentication method; and Buchegger and Le Boudec [12] proposed CONFIDANT which is based on a reputation system. Yi, Naldurg and Kravets [87] developed a generalized SAR (Security-Aware Ad-hoc Routing) protocol for discovering routes that meet certain security criteria. The protocol can be built on the top of any on-demand routing protocol and requires that all nodes that meet a certain criterion share a common secret key. Because of the space limitation, we are however going to detailedly discuss only SEAD,

ARAN, SAODV, SRP, Ariadne and CONFIDANT in the following subsections.

3.2.1 Secure Efficient Ad-hoc Distance-vector routing

Secure Efficient Ad-hoc Distance-vector routing (SEAD) [36] is based on the DSDV protocol [60]. SEAD is known for its addition of one-way hash chain for authentication on route update messages to the original DSDV. For a random value x, a one-way hash chain is defined as a sequence of hash value, $h_0, h_1, h_2, h_3, \cdots, h_n$, where $h_0 = x$ and $h_i = H(h_{i-1})$ and $0 < i \leq n$, for some n. In SEAD, each node is required to generate its hash chain at initialization time. The effectiveness of SEAD is grounded heavily on the assumption of the existence of a certain mechanism for a node to distribute an authentic element of its hash chain. The hash-chain-based authentication enables SEAD to be secure against forged route updates.

When a node sends a route update, the node assigns one hash value to each entry in that update. If a route update entry is destined for the node itself, it sets the entry's hash value to h_{n-i*m} where i is the corresponding sequence number and m is the upper bound of network diameter plus one; otherwise, it sets the entry's hash value to the hash of the hash value received in the route update entry where it learn that route to the destination. Because of the one-way nature of hash chain, a node receiving any route update can authenticate each entry in the update as long as it has any earlier authentic hash element from the same hash chain. For example, given an authentic hash value h_{i-3}, a node can authenticate h_i by computing $H(H(H(h_{i-3})$ and verifying that the resulting hash value equals h_i. Through authentication, each metric in a routing update entry is secured against being maliciously modified. The way it is done is as follows: the sequence number in an entry in a route update message is used to determine a contiguous group of m elements from the corresponding destination node's hash chain, and then, a particular element in the determined group is used to authenticate the entry. Specifically, for a sequence number i in some route update entry, let $k = n/m - i$, where n is divisible by m, then the group of m elements will be $(h_{km}, h_{km+1}, \cdots, h_{km+m-1})$. If the metric value of this entry is j, $0 \leq j < m$, then the element h_{km+j} is the one to be used to authenticate

the route update entry for that sequence number.

By using lightweight one-way hash function for authentication, SEAD reduces the risk of the DoS attacks where attackers broadcast a large number of forged route update packets to make nodes spend excess CPU cycle and processing time on verification. Through route update authentication, in SEAD, tampered route update packets can be detected and eliminated so that every node maintains correct routing information even with the presence of active attacks and compromised nodes. Nevertheless, this authentication mechanism has a problem. That is, a malicious node forwarding a route update might not increment the routing metric, namely hop count, but just uses the same hash value such that it will have more chance to make its neighbors to route packets through it.

3.2.2 Authenticated Routing for Ad-hoc Networks

Authenticated Routing for Ad-hoc Networks (ARAN) [74] is designed on top of AODV [61]. It requires a trusted authority issuing certificate to every node in the network. Nodes' certificates are used to authenticate the nodes themselves to other nodes during route construction process.

To initiate a route discovery, a source node broadcasts a route request packet signed with its private key. The route request message contains the destination ID, a certificate of the source, a nonce, and a timestamp. The nonce and timestamp together ensure the freshness of the packet. If a node receives a route request packet directly from the source, it just signs the packet with its own private key, appends its certificate to the packet, and then re-broadcasts the packet. If a node receives a route request packet from a non-source node, it first validates the certificate contained in this packet. If the certificate is valid, the node uses the public key in the certificate to verify the signature of the packet. If the signature is valid too, the node records the reverse route, removes both the signature and certificate of the prior hop, singes the packet, attaches its own certificate to the packet, and then forwards the packet to its neighbors. When the request reaches the destination, the destination signs a route reply packet and sends it back

to the source along the reverse route. The route reply packet is forwarded by each intermediate node in the same way as a route request packet except that each node unicasts the route reply to the node from which it received the corresponding route request. Route error packets are also signed by their initiators; a nonce and timestamp are used in each route error packet as well to ensure the freshness of the packet.

Because ARAN's security is based on public key cryptography, it is robust against almost all known attacks. However, since public key cryptography requires a large amount of computation power and processing time, ARAN is vulnerable to the DoS attacks where attackers floods the network with forged routing packets for which signature verification is needed.

3.2.3 Secure AODV

Secure AODV (SAODV) [88] is designed to secure the AODV protocol [61]. It assumes that every node has a (PublicKey, PrivateKey) pair and that the binding between a particular node and its public key can be verified securely. In SAODV, signature extensions are added to AODV routing packets to secure their non-mutable fields, while one-way hash chains are employed to authenticate the mutable field, i.e., hop count, in routing packets in the same way as in SEAD [36].

Before a source node sends a route request (RREQ) packet, it adds to the RREQ a route request single signature extension (RREQ-SSE). The RREQ-SSE is generated in the following way: the source node generates a one-way hash chain whose length is equal to the upper bound of network diameter plus one, signs the RREQ together with the top hash of the hash chain, and then includes the resulting signature and the top hash in the RREQ-SSE. The RREQ-SSE also contains an element of the hash chain. This element corresponds to the actual hop count indicated in the RREQ header. When an intermediate node receives the RREQ packet for the first time, it verifies the signature and the hash value in the RREQ-SSE of the packet. If the RREQ packet is valid, the node stores the reverse route, increments the hop count in the RREQ header, hashes the hash value contained in the RREQ-SSE, replaces the original hash

value with the resulting hash value, and finally forwards the modified RREQ packet to its neighbors. After the destination gets and processes the RREQ packet, it sends a route reply (RREP) packet back to the source along the reverse route. A RREP packet includes a route reply single signature extension (RREP-SSE) similar to RREQ-SSE. During the route reply process, the forward route is established.

Considering the case that a node A has a route to another node B. Node A must have previously forwarded a RREQ or RREP originated from B. Provided A had stored that RREQ/RREP and the contained signature of B, it can use that signature to reply future RREQs for B. However, it may happen that the pre-stored RREQ/RREP can no longer be fresh. In order to ensure freshness, Node A can check the sequence number or the lifetime field of the packet from which it learnt the route to B. To allow intermediate nodes to reply RREQ packets, SAODV uses a route request double signature extension (RREQ-DSE) and a route reply double signature extension (RREP-DSE). In order to protect route error messages (RERRs), SAODV requires that each node signs a RERR whether the RERR is originated from or forwarded by the node.

The objective of SAODV is to confine attacks only to dropping routing packets or lying about the information by the attackers themselves. Nevertheless, the one-way hash function based authentication suffers from the problem that a malicious node forwarding a RREQ packets might not increment the routing metric, namely hop count, but just use the same hash value it has received. When this situation happens, the malicious node would have more chance to make its neighbors to route packets through it. In addition, SAODV is not secure against DoS attacks.

3.2.4 Secure Routing Protocol

Secure Routing Protocol (SRP) [57] is based on DSR [40] and designed as an extension header attached to DSR route request and route reply packets. It works under the assumption that a security association (SA) can be established between a source and a destination.

In SRP, a source node initiates a route discovery process by broadcasting a route request packet containing a query sequence number, a random query identifier and a message authentication code (MAC) other than legacy DSR header. The MAC is computed using the source ID, the destination ID and the query identifier along with the shared secret between the source and the destination. After receiving a route request packet, an intermediate node extracts the query identifier, the source ID and the destination ID, stores them in its query table, and then retransmits the route request to its own neighbors. The combination of above three values determines a particular route request originated by the source toward the destination. If a node receives a route request matching one of the query table entries, the request is discarded. When a route request reaches the destination, the destination checks the freshness of the request by comparing the contained query sequence number with the maximum sequence number received by the SA from the source. Out-of-date route requests are discarded. If the route request is fresh, the destination checks the integrity of the route request packet by verifying the contained MAC; and if the packet is valid, it generates a route reply packet, which carries the accumulated route, the query identifier and the query sequence number of the corresponding route request as well as a MAC covering the rest of the route reply packet, and sends the packet back to the source. After receiving the route reply packet, the source will verify the integrity and freshness of the packet.

In order to allow intermediate nodes to reply route request, SRP employs the so-called intermediate node reply token (INRT). The generation of INRT is based on two alternative designs. In the first one, the source belongs to a group and shares a secret key with all the other group members, and the INRT will just be the MAC computed using the source ID and the destination ID along with the shared group key. In this case, the intermediate nodes that belong to the same group as the source and have an active route to the destination can validate the route request using the INRT and generate route reply using the key shared with the source. To enable this scheme, the source must either use the group key to generate the MAC for the route request or attach an extension, containing the group key based MAC, to the route request

packet. The alternative approach is to use digital signature. In this approach, the source uses its private key to generate the MAC for a route request such that every node is able to verify the request and then reply.

Although the freshness of a route request can be ensured through the use of sequence numbers, stale route requests can be detected only at destination nodes. In addition, SRP does not prevent unauthorized modification to route request/reply packets, and tampered or forged route request (reply) packets can be identified only by destination (source) nodes. Under these circumstances, intermediate nodes may transmit incorrect or false routing packets such that the valuable network bandwidth is consumed. Nevertheless, SRP nodes are able to process route requests selectively and therefore to be resilient against DoS attacks by weighing their neighbors based on the neighbors' out-going route request frequency.

3.2.5 Ariadne

Ariadne [37] is based on the DSR protocol [40]. It can authenticate routing packets using either shared secrets between each pair of nodes, or shared secrets between communicating nodes together with broadcast authentication, or digital signatures. In [37], Ariadne was introduced with TESLA, an efficient broadcast authentication scheme requiring loose time synchronization, under the assumption that the source and the destination share two keys, i.e., K_{SD} and K_{DS}, respectively for route request and route reply.

To initiate a route request process, the source node locally broadcasts a route request packet, which contains eight fields: ROUTE-REQUEST, source ID, destination ID, query identifier, TESLA time interval, hash chain, node list, and MAC list. Initially, the hash chain is set to the MAC computed using the first five fields along with K_{SD}, and both the node list and MAC list are empty. When a node X receives a route request packet for the first time, it validates the included TESLA time interval. If the time interval is valid, node X appends its own ID, $ID(X)$, to the node list, and replaces the hash chain field with $H(ID(X), hashchain)$, and then appends a MAC of the entire packet to the MAC list. The MAC is computed with

node X's TESLA key key_{X_i} where i is the index for the time interval specified in the route request packet. Afterwards, node X forwards the modified route request packet to its neighbors. When the destination receives the route request packet, it regenerates a hash chain based on the contained information and then checks whether the resulting hash chain equals the one contained in the route request packet. If they are equal, the destination returns to the source a route reply packet through the reverse order of the nodes specified in the node list field of the route request packet.

A route reply packet consists of eight fields: ROUTE-REPLY, destination ID, source ID, TESLA time interval, node list, MAC list, destination MAC, and key list. The destination ID, source ID, TESLA time interval, node list, and MAC list fields are just set to the corresponding values obtained from the route request packet while the key list is initialized to be empty. The destination MAC is based on all the other fields but the key list and is computed with K_{DS}. After receiving a route reply packet, an intermediate node waits until it is able to disclose its key from the time interval specified in the route reply packet, and then, it appends its key to the key list field of the route reply packet. Afterwards, the intermediate node forwards the modified route reply packet to its prior hop according to node list in the packet. After the source receives the route reply packet, it validates each key in the key list, the destination MAC, and each MAC in the MAC list. Any failure in the course of validation causes that the route reply packet is discarded.

Similarly, Ariadne modified the format of route error packet and requires that the source authenticate a route error packet to prevent fraudulent route error report. According to [37], Ariadne prevents attackers or compromised nodes from tampering with routing information and prohibits a large number of types of DoS attacks, but it is vulnerable to traffic analysis since routing information is not protected.

3.2.6 CONFIDANT

CONFIDANT protocol [12] is based on the DSR protocol [40]. It is a reputation-based secure routing protocol. In CONFIDANT, each node has four components: the monitor, the reputation system, the path manager, and the trust manager. These four components enable a node to detect deliberate malicious behaviors, e.g., no forwarding, unusual traffic attraction, malicious rerouting, lack of error messages, unusually frequent route updates, and silent route change, done by other nodes through observation and reports.

In CONFIDANT, for an arbitrary node A, its monitor, $M(A)$, keeps surveiling its neighborhood all the time. When a suspicious event, denoted by e, of certain neighbor, say X, is detected, $M(A)$ informs node A's reputation system, $R(A)$. To avoid the interference from X's occasional mistake due to, for example, network congestion, $R(A)$ decreases X's reputation rating (stored in a rating list) only when e happens more than a maximum number of times. Let us assume that e is performed by X on purpose and that X's reputation rating is bad. $R(A)$ then passes the information to the path manager of A, $P(A)$, which in turn deletes all the routes that go through X. Then, A's trust manager $T(A)$ sends an ALARM message to warn other nodes of the malicious node X. The intended receiver of the ALARM message could be either a source, or a destination, or a friend of A. Let us denote the destination of the ALARM message by B. After $M(B)$ receives the ALARM message, it passes the message to $T(B)$. $T(B)$ in turn checks how trustworthy A is and how many similar reports about X have been received, and then, it processes the message accordingly. After B is certain of the ALARM message from A about X, it passes the information to $R(B)$ which performs an evaluation on X again. To prevent false ALARM messages, authentication mechanisms such as PGP [90] can be used during above process.

In addition to rating lists, nodes also maintain black lists. Nodes appearing in black lists are avoided during routing, and packets are forwarded only to the nodes that are not contained in black lists. Using this approach, adverse nodes are identified and isolated from the network, and therefore, route robustness is increased, and network throughput is improved.

However, we should indicate that the problem of reputation improvement is not addressed in CONFIDANT and the reputation system only takes negative input into consideration.

3.3 Anonymous Wireless Ad-hoc Routing Protocols

As what we mentioned earlier, the threat posed by traffic analysis to wireless ad-hoc networks is not negligible. However, the secure routing protocols discussed in the previous section ensure only the authenticity and/or integrity but not the privacy of routing information. In those protocols, the intermediate nodes processing route control messages can easily learn the identities of communicating nodes, which must be protected in the case of anonymous communication. Using anonymous routes for communication is the most commonly employed approach to accomplish anonymous communication that is resilient against traffic analysis. However, anonymous routing is such a tough research problem, especially in wireless ad-hoc networks, that there are not so many people addressing it. Only recently, a small number of anonymous routing protocols, including ANODR [43], ASR [89], dynamic MIX route algorithm [39], and our SDAR routing protocol, were proposed for wireless ad-hoc networks to address the problem. Since ASR is just an extension to ANODR, and since the detailed presentation about our SDAR is provided in Chapter 4, in the following subsections, we are going to introduce only ANODR and the dynamic MIX route algorithm.

3.3.1 ANonymous On Demand Routing

ANonymous On Demand Routing (ANODR) [43] is developed using a new concept of "broadcast with trapdoor information". It borrows the idea of Onion Routing [66] for route discovery. In an anonymous route established by ANODR, both the sender and the receiver can not identify any intermediate node, and all the intermediate nodes know nothing about the route (neither the source/destination nor the prior/next hop).

In ANODR, a route discovery process is initiated by a source node by broadcasting a

route request (RREQ) packet. To do so, the source node randomly generates a symmetric key K_{src} and computes $K_{src}(ID_{src})$. Then, it randomly generates a commitment key K_c and computes $K_c(ID_{dest})$. The source node then encrypts the combination of ID_{dest} and K_c with the destination's TESLA key K_T, and it generates a globally unique sequence number *seqnum* and a one-time public key pair (pk_{one}, sk_{one}). The source node then assembles a RREQ packet as follows: ($RREQ$, *seqnum*, pk_{one}, $K_T(ID_{dest}, K_c)$, $K_c(ID_{dest})$, *Onion*), where the field $RREQ$ indicates message type and *Onion* is set to $K_{src}(ID_{src})$. Finally, the source node broadcast this RREQ packet to its neighbors. During the above process, the source node bookkeeps all the relevant data such as K_{src}, K_c, and (pk_{one}, sk_{one}).

When an arbitrary node X receives the RREQ packet, it first checks if it itself is the destination using the following steps: decrypt the $K_T(ID_{dest}, K_c)$ field of the RREQ packet with its own TESLA key K'_T to get ID'_{dest} and K'_c; then verify if its own ID is equal to ID'_{dest}; if they are equal, use K'_c to decrypt the $K_c(ID_{dest})$ field of the RREQ to double check if it is truly the destination. If X is not the intended destination, it randomly generates a symmetric key K_X and an asymmetric key pair (pk'_{one}, sk'_{one}). Then, it extracts the *Onion* from the RREQ packet, and encrypts the combination of the *Onion* and a random nonce N_X with K_X, and replaces the original *Onion* in the packet with the encryption result. Afterwards, it replaces the pk_{one} in the packet with pk'_{one}. Finally, it forwards the modified RREQ packet to its neighbors. During this process, X bookkeeps all the necessary data such as pk_{one}, (pk'_{one}, sk'_{one}), and K_X.

When the RREQ packet reaches the destination, the destination sends a RREP packet back to the source. To do so, the destination generates a random nonce K_{seed} and encrypts K_{seed} with the pk_{one} extracted from the RREQ packet. It then uses a trapdoor one-way function with K_c, *Onion*, and K_{seed} as input. The output of the one-way function, is denoted by $K_{seed}(K_c, Onion)$. Next, the destination assembles the RREP packet which has the following format: ($RREP$, $(K_{seed})_{pk_{one}}$, $K_{seed}(K_c, Onion)$), where $RREP$ indicates message type. Finally, the destination broadcasts the RREP packet.

When a node X receives the RREP packet, it first decrypts the $(K_{seed})_{pk_{one}}$ with the backuped

(during route request process) one-time private key sk_{one} to get K_{seed}. Then, it recovers K_c and *Onion* from the $K_{seed}(K_c, Onion)$ field of the RREP packet using K_{seed}. Afterwards, X decrypts the *Onion* with K_X (corresponding to sk_{one}) and checks whether N_X (corresponding to sk_{one}) is equal to the first field of the decryption result. If so, it knows that it is in the anonymous route and continues packet processing; otherwise, it simply discards the packet. If X is in the anonymous route, then X peels off the topmost layer of the *Onion*, and removes the first field of the result, and then gets a resulting onion *Onion'*. Afterwards, X computes $K'_{seed} = f(k_{seed})$ (f is a one-way function) and encrypts K'_{seed} with the prior hop's one-time public key pk'_{one}. Next, X computes $K'_{seed}(K_c, Onion')$ through a trapdoor one-way function. Finally, it replaces the $(K_{seed})_{pk_{one}}$ field with $(K'_{seed})_{pk'_{one}}$, and the $K_{seed}(K_c, Onion))$ field with $K'_{seed}(K_c, Onion'))$, and then broadcasts the modified RREP packet to its neighbors. We should mention that the K_{seed} in the original RREP is the route pseudonym for X and its next hop to exchange data packets while K'_{seed} is the route pseudonym for X and its prior hop to exchange data packets. When the source node receives the RREP packet, it can verify whether the destination has received the RREQ packet using the K_c in the RREP packet and its backuped one.

By above description, each intermediate node stores two route pseudonyms respectively for its prior and next hop, and the source (destination) has only one the route pseudonym for its next hop (prior hop). As a result, an anonymous route is successfully established.

3.3.2 A Dynamic Mix Route Algorithm

To simplify expression, the dynamic Mix route algorithm [39] is referred to as DMRA in this thesis. DMRA borrows the idea of MIX-nets [14] for anonymous data transfer. It assumes that there are a number of MIX nodes supporting anonymous communication in the wireless ad-hoc network, and that each MIX node has an asymmetric key pair. Any anonymous route between two nodes is established through a sequence of MIX nodes. Its route discovery is built on the top of an underlying routing protocol, e.g., DSR [40], and thus, DMRA is an application-layer routing algorithm. Since DMRA works in the context of wireless ad-hoc

networks where nodes (including MIX nodes) may be mobile, its main concern is how to maintain a dynamic anonymous route between two nodes.

In DMRA, every MIX node periodically broadcasts a MIX advertisement (MADV) message to claim its existence. A MADV message includes the initiator's ID, a sequence number and a hop count. The initiator's ID and the sequence number together uniquely identify a MADV message. The hop count indicates how far the message receiver is from the message initiator. A node may receive a MADV message from different MIX nodes, or it may receive a MADV message from the same MIX node multiple times, but it re-broadcasts only the one that is with minimal hop count and is received for the first time. In this case, a MADV message will not flood the entire network. Instead, it covers only a limited area. A node increments the hop count field of a MADV message before it re-broadcasts the message. By listening MADV messages, a node can find the closest MIX node. The MIX node closest to a node is taken by the node as its *dominator*. Because network topology is changing, a node's dominator may change over time accordingly.

When a source node wishes to set up an anonymous connection to a destination node, it sends a route request (RREQ) message to its dominator or a randomly selected MIX node, which in turn forwards the message to the destination. This process is finished using the underlying routing protocol, and the RREQ message is encrypted with the destination's public key. After the destination receives the RREQ message, it sends its dominator a destination registration (DREG) message if it is not yet registered with its dominator. A DREG message includes the initiator's ID and a sequence number. From then on, the destination periodically sends to its dominator a DREG message, and it each time increments the sequence number.

Every MIX node maintains a list of registered nodes, denoted by l. Each entry of l consists of a node ID and a corresponding DREG sequence number. For a MIX node, if it does not receive a DREG message from certain registered node for a pre-configured period of time, it removes the node from its l. As long as its l is not empty, the MIX node keeps periodically broadcasting a route update (RUPD) message throughout the network. A RUPD message

includes the initiator's ID, a sequence number, a node list, and a path list. The node list is just the initiator's l, while the path list contains the routes to the nodes in the node list.

Let us denote the node list and path list of a RUPD message respectively by *nl* and *pl*. When a node X receiving a RUPD message, it checks if it has some enqueued data packets that are designated to certain nodes in the *nl*. If so, it copies the corresponding MIX routes in the *pl* and uses the reverse MIX routes to deliver those data packets. Note that X may learn multiple distinct Mix routes to the same destination from the RUPD messages that it has received through different paths. If X is a MIX node, it checks if any node in the *nl* carries a higher DREG sequence number and updates its l accordingly. Then, X appends its own ID to each entry of the path list and re-broadcasts the modified RUPD message. For the same RUPD message, X re-broadcasts it only once.

In [39], the authors pointed out that the sender and receiver anonymity may be broken by attackers through global observation, and that attackers may learn route information during the MIX route update process. To reduce the risk, they suggested source nodes send dummy messages to confuse attackers and also make use of discovered multiple routes to deliver data packets. Besides, since RUDP messages are transmitted in plaintext, attackers can find who are involved in communication as communicating end although they may not find exactly who is talking to whom. And, if a node's dominator is compromised, the node will lose sender and/or receiver anonymity.

Chapter 4

SDAR: A Secure Distributed Anonymous Routing Protocol

In this chapter, we will present our secure distributed anonymous routing protocol (SDAR) for wireless ad-hoc networks. The major objective of SDAR is to allow trustworthy intermediate nodes to participate in path construction without jeopardizing the anonymity of the communicating nodes. The protocol has a number of characteristics, including non-source-based routing, flexible and reliable route selection, and resilience against path hijacking. SDAR is secured against passive and active attacks, but not against Denial-of-Service attacks. It maintains the anonymity of senders and receivers, and it is able to establish a route matching certain trust requirement if there exist enough nodes in the requested trust level between the source and destination. SDAR is developed based on the previous work of K. El-Khatib et al. [20]; the main terminologies used for describing SDAR are listed in Table 4.1.

4.1 Overview

In SDAR, to send data anonymously to a receiver node R, a sender node S has to discover and establish a reliable and anonymous path that connects them together. Both the path discovery and the path establishment processes should be carried out securely and without jeopardizing the anonymity of the communicating nodes. The protocol consists of three phases: the *path discovery* phase, the *path reverse* phase and the *data transfer* phase. Distributed information gathering about intermediate nodes that can be used along an anonymous path is carried out during the *path discovery* phase, while passing this information to the source node takes place during the *path reverse* phase. The official data exchange is processed during the *data transfer* phase after the construction of the route. A source node initiates a *path discovery* phase by locally broadcasting a *path discovery* message with certain trust requirement. Each intermediate node satisfying the trust requirement inserts into the *path discovery* message its

Name	Description
AK	The agent secret key
AL	The agent launcher
TRA	A trust and reputation assessor agent
ID_i	The identity of node i
PK_i	The public key of node i
TPK	A temporary one-time public key
TSK	The private (secret) key corresponding to TPK
K_i	A Symmetric (session) key generated by node i
PL_S	The padding length set by the sender
P_S	padding implemented by the sender
PL_R	The padding length made by the receiver node R
P_R	A padding made by the receiver node R
$E_{PK_i}(M)$	The message M is encrypted with a public key PK_i
$E_{K_i}(M)$	The message M is encrypted with the symmetric key K_i
$H(M)$	The message M is hashed with a hash function
$H_{K_i}(M)$	The message M is hashed and then encrypted with K_i
$Sign_S(M)$	The message M is signed by the source S
$SN_{session\_ID_i}$	A session key ID generated by node i
HCK_i	A high trust level community key generated by node i
MCK_i	A medium trust level community key generated by node i

Table 4.1: Notations for SDAR

own ID, a session key and a session key ID, all of which are together encoded with a temporary public key. Afterwards, the intermediate node encrypts the *path discovery* message with proper community key according to the trust requirement and locally broadcasts the message such that only intended neighbor nodes are able to decrypt it. Once the receiver node receives the message, it retrieves from the message the information about all intermediate nodes, encapsulates this information in a multi-layered message, and sends it along a reverse path in the dissemination tree back to the source node. Each intermediate node along the reverse path removes one encrypted layer from the message, and forwards the one-layer-less message to its ancestor node until the message reaches the source node. When the protocol terminates, the source node ends up with the information about all the qualified (in terms of trustworthiness) intermediate nodes on the discovered route as well as the session keys to encrypt the data transmitted through each of these nodes. The multicast mechanism and the layered encryption used in the protocol ensure the anonymity of the sender and receiver nodes.

4.2 Assumptions

Before we proceed further, we make the following assumptions in a wireless ad-hoc network.

- The links between wireless nodes are always bi-directional.
- Every wireless node has enough computation power to execute cryptography algorithms.
- There is a trusted CA outside the network, which issues certificates to the wireless nodes inside the network.
- Each wireless node uses one and only one unique ID for its communication.
- There are some nodes which are not willing to cooperate for routing and data delivery or are very likely to tamper with the routing protocol.

4.3 Trust Management Scheme

As we mentioned earlier, due to the openness of a wireless ad-hoc network, some nodes in the network are likely to defect and become harmful to the network, thereby necessitating a mechanism to identify these nodes and isolate them. In this section, we will introduce our trust management approach as well as the trust notion we choose to use in a wireless ad-hoc environment to select routing paths that meets certain trust requirements. The purpose of the trust management scheme is not only to motivate participating nodes to help each other relay data traffic, but also to identify malicious nodes and furthermore to avoid using them during route establishment. The identification of malicious nodes makes it easy to preclude unqualified nodes from route construction, thereby increasing route security and reliability. In our approach, we define the trust of a node as a cumulative value that is based on the past behavior of that node. The trust of a node increases as the node behaves exactly as it is supposed to (in our cases, follow reliably the steps of the routing protocol) or accordingly decreases as the node misbehaves. A node's trust is computed by each of its direct neighboring nodes based on their past experience with or observation of the node's behavior. These neighboring nodes, together with the evaluated node, form what we refer to as a *community*, as we will describe later.

4.3.1 Community Management

In our scheme, we define a node's community as the set of nodes that includes the node itself, referred to as *central node*, and all of its one-hop neighboring nodes. Figure 4.1 shows three communities. To build and maintain a node's community, SDAR employs a similar method used by AODV ad-hoc routing protocol [61]. That is, a node keeps track of its neighbors simply by listening for a HELLO message, which is broadcasted periodically by each node. The sender's public key is passed as part of the HELLO message. Upon receipt of a HELLO message from one of its neighboring nodes, a *central node* stores the neighboring

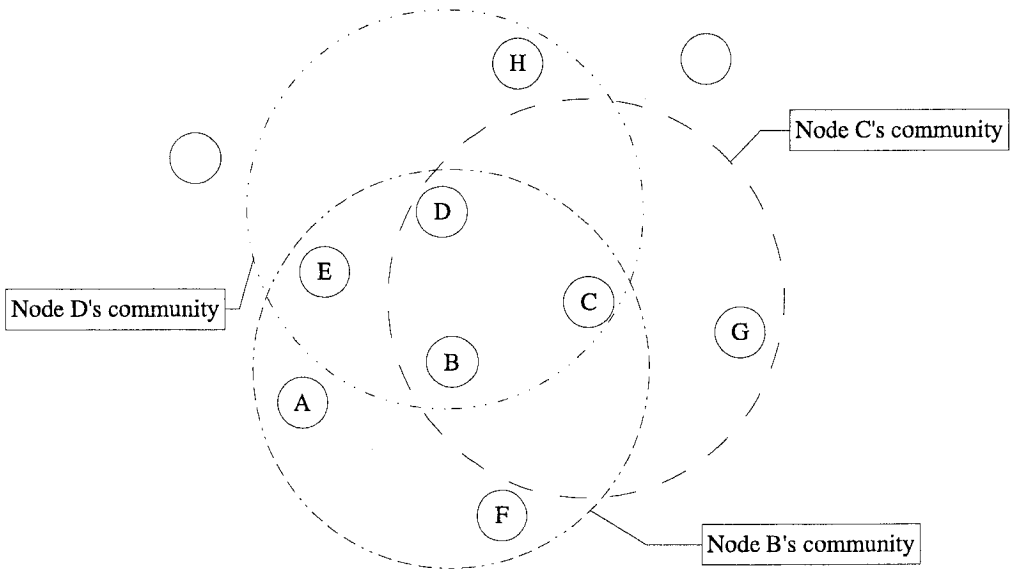


Figure 4.1: Three communities

node's the public key if it does not have it yet. Since nodes can move freely in a wireless ad-hoc network, some neighbors of the *central node* may leave while new neighbors may join the neighborhood of the central node from time to time. Thus, if a node did not receive for a certain period of time the HELLO message from one of its neighbors, it removes that neighbor from its neighbor list.

4.3.2 Community Key Management

In every community, the central node classifies its neighboring nodes into three classes, based on their trust level. The first and lowest trust level is for the nodes whose trust values are between 0 and a *Medium Trust Level Threshold* (MTLT), δ_1, while the second trust level, i.e., the medium level, contains the nodes whose trust values are between δ_1 and the *High Trust Level Threshold* (HTLT), δ_2. The trust level, corresponding to the high level, contains the nodes whose trust values are between δ_2 and 1. Each node independently selects the values for δ_1 and δ_2. In our experiments to be shown, both values have been determined empirically.

A *central node* generates two different symmetric keys, *High Trust Level Community Key* (HTLCK) and *Medium Trust Level Community Key* (MTLCK), and shares them with selected

neighbors. The two community keys are used only for the traffic going from the central node to its neighbors in a community during the *path discovery* phase. All the neighbors in the same trust level share the same key. The neighbors in high trust level will have both the HTLCK and the MTLCK. Whereas, the neighbors in medium trust level have only the MTLCK. As for the neighbors in low trust level, they do not share any community key at all.

Having detected a new neighbor, a *central node* assigns an initial trust value to it and updates its trust level later based on its behavior. The *central node* re-generates the corresponding community key when a neighbor's trust level goes down, and also when a neighbor leaves the community. Community key distribution is done through UPDATE messages. To protect a community key during distribution, the *central node* encrypts the key with the public key of the intended neighboring node before sending it. The *central node* repeats the same process with each node in its neighboring node set. For each received UPDATE message, a node sends an ACK message back to the sending node to confirm its successful receipt.

4.3.3 Trust-Based Distributed Route Selection

SDAR, as we shall see in the next section, requires each intermediate node, which receives a *path discovery* message, to forward this message to its neighboring nodes. But, in order to achieve route security and reliability, SDAR uses a distributed route selection algorithm that is based on the level of trust each intermediate node has with its neighboring nodes.

When a source node initiates a *path discovery* phase, it specifies the trust requirement in the *path discovery* message. Each intermediate node will propagate the message only to its selected neighboring nodes, depending on the trust level requested by the source node. If the requested trust level is *HIGH*, the node will use the HTLCK to encrypt the message; this will ensure that only highly trusted nodes will participate in the route construction. If the required trust level is *MEDIUM*, the node will use the MTLCK to encrypt the message. Using this approach restricts the nodes in the eventually established route only to the ones that are in the expected trust level.

4.4 Anonymous Route Construction

In this section, we will elaborate the *path discovery* phase and the *path reverse* phase of SDAR protocol.

4.4.1 Path Discovery Phase

The *path discovery* phase allows a source node S that wants to communicate securely and privately with a destination node R to discover and establish a routing path through a number of intermediate wireless nodes. An important characteristic of this phase is that none of the intermediate nodes that participated in the *path discovery* phase can discover the identity of the sending node S and the receiving node R.

The source node S initiates the *path discovery* phase by sending a *path discovery* message to all the nodes within its wireless transmission range. The *path discovery* message has five parts. The first part is an open part. It consists of message type, $TYPE$, trust requirement, $TRUSTREQ$, and a one-time public key, TPK. The trust requirement indicated by $TRUSTREQ$ could be either $HIGH$ or $MEDIUM$ or LOW. TPK is generated by the source node for each *path discovery* session and used by each intermediate node to encrypt routing information appended to the *path discovery* message. This key serves also as a unique identifier for the message. The second part contains the identity ID_R of the intended receiver, the symmetric key K_S generated by the source node, and the PL_S indicating the length of the third part, all encrypted with the public key PK_R of the receiver. The source node may learn about the public key PK_R of the target receiver through a number of ways including using the service of a certificate authority (CA). The symmetric key K_S is also called session key. It is used to encrypt the fourth part of the message. The third part is a padding P_S, generated by the source node and used to hide real routing information and to protect against message size attacks. The forth pare consists of ID_S, TSK, $SN_{SessionID_S}$, SEQ and $Sign_S(M_S)$, all encrypted with K_S. The intended receiver uses the temporary public key TPK and the corresponding private

$$
\begin{aligned}
&TYPE, TRUSTREQ, TPK,\\
&E_{PK_R}(ID_R, K_S, PL_S),\\
&P_S,\\
&E_{K_S}(ID_S, TSK, SN_{SessionID_S}, SEQ, Sign_S(M_S))
\end{aligned}
$$

Figure 4.2: Path discovery message just sent by the source

key TSK to decrypt and verify the routing information in the message. $SN_{SessionID_S}$ is referred to as session key ID. It is a random number generated by the source node and is mapped to the session key K_S to use with the message. SEQ is a monotonically increased sequence number maintained by the source node. It is incremented every time when the source node broadcasts a *path discovery* message and is used to prevent *replay attacks*. $Sign_S$ protects the integrity of the message. The fifth part of the message contains information about the intermediate nodes prior to current node in the route along which the message ever traveled. A *path discovery* message just sent by a source node has the format shown in Figure 4.2 where M_S is defined as $H(TYPE, TRUSTREQ, TPK, ID_R, K_S, PL_S, P_S, ID_S, TSK, SN_{SessionID_S}, SEQ)$.

Each node keeps an internal table for mapping the randomly generated session key ID of a session to the session key for the session, as well as to the immediate ancestor and successor nodes along the anonymous path for the session. Given an encrypted message and a session key ID, a node can use this mapping table to know which session key to use to decrypt the message. Only the session key ID, the session key, and the ancestor node entry are added to the table during the *path discovery* phase, while the successor node entry is added during the later *path reverse* phase.

When $node_i$ receives a *path discovery* message, it processes the message according to the following steps:

1. Check if the message has already been received from other nodes within its wireless transmission range using the TPK as the unique identifier for the message. If the message was received previously, drop it silently and stop; otherwise, continue.

2. Check if it is the intended next hop by finding the community key corresponding to the trust requirement in the message. If the key is found, then decrypt the message using that key and go to the next step; otherwise, stop.

3. Check if it is the destination by decrypting $E_{PK_R}(ID_R, K_S, PL_S)$ with its private key and comparing the ID_R with its ID.

4. If it is NOT the destination, then

   (a) Add the following information to the message, all encrypted with the TPK: its own identity ID_i, a session key K_i (shared encryption key generated by the node), a randomly generated session key ID $SN_{SessionID_i}$ for the session, and the signature for the original message together with ID_i, K_i and $SN_{SessionID_i}$.

   (b) Forward the modified message to the neighbors whose trust levels meet the trust requirement claimed by the source node.

   (c) Add $< SN_{SessionID_i}$, ID of the ancestor node, $K_i >$ to the local mapping table.

5. If it is the destination, then

   (a) Recover PL_S and K_S from $E_{PK_R}(ID_R, K_S, PL_S)$; use PL_S to find out the offset of the forth part of the message; decrypt the forth part of the message with K_S and afterwards extract TSK and SEQ from it; use the SEQ to verify the freshness of the message; use the TSK to retrieve session keys for all the nodes in the path along which the message ever traveled.

   (b) Put all the session keys and session key IDs of the intermediate nodes in one message; encrypt the message several times, each time with the session key of a node in the discovered path. Use the reverse order of the keys in the message (same as the data flow in Onion Routing)

   (c) Send the newly-generated message to the node from which it received the *path discovery* message

$$
\begin{array}{l}
TYPE, TRUSTREQ, TPK, \\
E_{PK_R}(ID_R, K_S, PL_S), \\
P_S, \\
E_{K_S}(ID_S, TSK, SN_{SessionID_S}, SEQ, Sign_S(M_S)), \\
E_{TPK}(ID_1, K_1, SN_{SessionID_1}, Sign_{ID_1}(M_{ID_1})), \\
E_{TPK}(ID_2, K_2, SN_{SessionID_2}, Sign_{ID_2}(M_{ID_2})), \\
\qquad\qquad \vdots \\
E_{TPK}(ID_i, K_i, SN_{SessionID_i}, Sign_{ID_i}(M_{ID_i}))
\end{array}
$$

Figure 4.3: Path discovery message just processed by node i

A *path discovery* message that just passed by nodes i on its way from the sender S to the receiver R would have the format shown in Figure 4.3, where $M_S = H(TYPE, TRUSTREQ, TPK, ID_R, K_S, PL_S, P_S, ID_S, TSK, SN_{SessionID_S}, SEQ)$, and $M_{ID_i} = H(M_{prev}, ID_i, K_i, SN_{SessionID_i})$, and M_{prev} is the cumulative message that $node_i$ gets from its ancestor $node_{i-1}$.

4.4.2 Path Reverse Phase

The *path discovery* message is forwarded from one node to the other in the network until it reaches the receiver R, which then triggers the *path reverse* phase. After the receiver R gets the *path discovery* message, it can use its private key to retrieve K_S. Then using K_S, it can obtain the temporary private key TSK and the source sequence number SEQ embedded in the fourth part of the message. By comparing the SEQ with the most recent sequence number received from the source, the receiver R is able to preclude stale *path discovery* messages. Using TSK, R can retrieve the session key and session key ID of each intermediate node in the fifth part of the *path discovery* message. Integrity check is performed during above process. If the *path discovery* message is fresh and is not forged, and if it has not been tampered with, the receiver R then composes a message that contains all the obtained session key IDs and the corresponding session keys, and recursively encrypts the message with the session keys of all the nodes

$$
\begin{aligned}
&TYPE,\\
&E_{K_i}(E_{K_{i-1}}(E_{K_{i-2}}\cdots(E_{K_2}(E_{K_1}(E_{K_S}\\
&PL_R, P_R, SN_{SessionID_1}, K_1, \cdots, SN_{SessionID_i}, K_i, SN_{SessionID_R}),\\
&SN_{SessionID_S}, SN_{SessionID_{S-2}}, H(M_{S-2}), H_{K_S}(N_S)),\\
&SN_{SessionID_1}, SN_{SessionID_{S-1}}, H(M_{S-1}), H_{K_1}(N_1)),\\
&SN_{SessionID_2}, SN_{SessionID_S}, H(M_S), H_{K_2}(N_2)),\\
&\qquad\vdots\\
&SN_{SessionID_{i-2}}, SN_{SessionID_{i-4}}, H(M_{i-4}), H_{K_{i-2}}(N_{i-2})),\\
&SN_{SessionID_{i-1}}, SN_{SessionID_{i-3}}, H(M_{i-3}), H_{K_{i-1}}(N_{i-1})),\\
&SN_{SessionID_i}, SN_{SessionID_{i-2}}, H(M_{i-2}), H_{K_i}(N_i)
\end{aligned}
$$

Figure 4.4: Path reverse message

along the path to the source node. With each encryption, the receiver R adds a layer that contains the session key ID generated by an intermediate node and the session key ID generated by the node's next-next-hop node along the reverse path to the sender S. If the node from which the receiver R got the *path discovery* message is node i, the encrypted message constructed by the receiver R will have the format shown in Figure 4.4, where $M_i = (E_{K_i}(M_{i-1}), SN_{SessionID_i}, SN_{SessionID_{i-2}}, H(M_{i-2}), H_{K_i}(N_i))$, and $N_i = (E_{K_i}(M_{i-1}), SN_{SessionID_i}, SN_{SessionID_{i-2}}, H(M_{i-2}), H_{K_i}(N_i))$, and M_{S-1} and M_{S-2} are a randomly generated padding, and $SN_{SessionID_{S-1}}$ and $SN_{SessionID_{S-2}}$ are a random number with the same number of bits as a regular session key ID.

Each intermediate node that receives the *path reverse* message uses the session key ID to retrieve the corresponding session key for the session, and removes one encryption layer, and then forwards the one-layer-less message to the next node along the reverse path to the source node. The ID of the node from which the *path reverse* message was received is also added into the mapping table as the successor node entry corresponding to the session key ID. After the source node receives the message, it decrypts the message, gets the complete information about the route, and then starts the *data transfer* phase.

4.5 Anonymous Data Transfer

SDAR uses a similar approach to the Onion Routing protocol's for the data transfer. When the source node gets the *path reverse* message, it first checks whether or not the message is correct, and then uses the shared session keys of the intermediate nodes to make the layer encryption for the data that it wants to transfer to the receiver. By using the mapping table, each intermediate node decrypts one encryption layer with proper session key and forwards the rest of the message to the next node until the message reaches the target receiver.

4.6 Identification of Malicious Behavior

In this section, we will describe how each node can compute and constantly update the trust it holds in its neighboring nodes. Our approach is based on the ability of a node to identify proper/malicious behavior of neighboring nodes, and hence updating the their trust levels accordingly. A behavior is proper if it confirms to the specification of the routing protocol, or malicious otherwise. For SDAR protocol, a malicious behavior happens when a node drops the packet without forwarding it or adversely alerts the packet before forwarding it. We call these two types of malicious behavior *Malicious Dropping* and *Malicious Modification*. A node can identify these two types of behavior of its neighboring nodes simply by overhearing their communication. Note that for the destination node of a message to protect its anonymity without jeopardizing its trust, it must also forward a copy of the message. The following is the detailed description about how a node can identify these two kinds of malicious behavior of its neighboring nodes.

4.6.1 Finding Malicious Dropping Behavior

During the *path discovery* phase, a unique identifier, TPK, is carried in every *path discovery* message. A node can find malicious dropping simply by expecting to overhear the message with the same TPK from the intended neighboring node.

During the *path reverse* phase and the *data transfer* phase, the same check is performed except that the session key ID is used instead of the TPK. In these two phases, a node's next hop's session key ID is obtained from path reverse message or data transfer message and locally stored by the node's predecessor. By observing the *path reverse* message format shown in Figure 4.4, we can see that $node_i$ can get not only its own session key ID, $SN_{SessionID_i}$, but also $SN_{SessionID_{i-2}}$, the session key ID of its next-next-hop node, $node_{i-2}$. The predecessor then expects that it should overhear the message carrying the session key ID of the node's next hop from the node.

4.6.2 Finding Malicious Modification Behavior

We will show here how to identify *malicious modification* during the *path reverse* phase and the *data transfer* phase taking the node $node_{i-1}$ in Figure 4.4 as an example. When $node_{i-1}$ receives the message, M_{i-1}, from $node_i$, it processes M_{i-1} according to the following steps:

1. $node_{i-1}$ retrieves $SN_{SessionID_{i-1}}$, $SN_{SessionID_{i-3}}$ and $H(M_{i-3})$ from M_{i-1}, stores $SN_{SessionID_{i-3}}$ and $H(M_{i-3})$ locally, and then gets the session key K_{i-1} corresponding to $SN_{SessionID_{i-1}}$.

2. $node_{i-1}$ retrieves N_{i-1} from M_{i-1}, hashes it, and then encrypts the hash value with K_{i-1}.

3. $node_{i-1}$ checks if the encryption result of previous step is equal to the $H_{K_{i-1}}(N_{i-1})$ contained in M_{i-1}. If it is not, $node_i$ must have done *malicious modification* to M_{i-1}, and the checking stops; otherwise, the message, M_{i-1} , sent by $node_i$ is not modified, and continue.

4. $node_{i-1}$ retrieves the $E_{K_{i-1}}(M_{i-2})$ from M_{i-1}, decrypts it with K_{i-1}, and sends the decryption result, M_{i-2}, to $node_{i-2}$ with no modification.

5. Having overheard M_{i-3} from $node_{i-2}$ (recall that $node_{i-2}$ is supposed to process M_{i-2} in the same way to get M_{i-3} and then send M_{i-3} to $node_{i-3}$), $node_{i-1}$ hashes it and checks whether the hash value is equal to the previously stored $H(M_{i-3})$ in Step 1. If they are

equal, the message, M_{i-3}, sent by $node_{i-2}$ is not modified; otherwise, $node_{i-2}$ must have had malicious modification behavior on M_{i-3}.

4.7 Protocol Characteristics

The proposed SDAR protocol has a number of characteristics, including non-source-based routing, trust-based route selection and resilience against path hijacking.

4.7.1 Non-Source-Based Routing

SDAR has a major advantage over the legacy routing protocol for anonymous communication systems in the way routing paths are constructed. That is, it does not require a global view of network topology. In standard Onion Routing [66], the source onion node must know the topology and link state of the network as well as the public keys of all onion nodes on the path before it can establish a path. In our protocol, each node in the network contributes to the final path by forwarding *path discovery* and *path reverse* messages. This approach eliminates the need for managing routing centrally. Moreover, similar to other dynamic routing protocols, the proposed protocol gathers routing information only when the session is started or when the path breaks. Information about nodes that have joined or left the wireless ad-hoc network need not be propagated to all the other nodes. In standard Onion Routing, this information is needed in order for source onion nodes to build a routing path with viable nodes.

4.7.2 Trust-Based Route Selection

In SDAR, route selection decision is made by neither source nodes nor destination nodes. Instead, every intermediate node along a route makes contribution to the route selection by selecting its neighbors based on the trust relationship between them and according to the trust requirement claimed by the source node. In this case, a node is able to intentionally establish a route in a given trust level by including a proper trust requirement in the *path discovery*

message it originates. Since trust reflects a node's cooperation performance in the past, every two neighboring nodes in the eventually established route can guarantee the communication happening between them to certain extent. As a result, the entire route can offer certain degree of guarantee to the traffic going through it. The trust requirement claimed by the source node for the route and the distributed route selection scheme together make SDAR protocol more flexible and more reliable.

4.7.3 Resilience against Path Hijacking

During the *path discovery* phase, each intermediate node receives a *path discovery* message and forwards the message to its neighbors. While a well-behaving node forwards the message to the neighboring nodes in an expected way, a malicious node might forward the message only to its neighboring malicious nodes, resulting in a path with only malicious nodes. We refer to this situation as "path hijacking". The proposed protocol proves to be resilient against path hijacking. To confirm that, note that the protocol terminates only after the intended receiver initiates the *path reverse* phase, and after the *path reverse* message has made its way successfully to the source node. If malicious nodes keep on forwarding a *path discovery* message among themselves, the message will never get to the intended receiver and the source node will never get a *path reverse* message triggered by the *path discovery* message. Although in this case, the protocol may fail to return a suitable path, it is still resilient to path hijacking in the sense that the actual hijacking does not occur (note that other *path discovery* messages might still have made their way to the intended receiver and triggered a successful *path reverse* phase). If on the other hand, a malicious node decides to break the cycle and forwards the message to a non-malicious node, and if this message gets to the intended receiver and triggers a *path reverse* message, the path will be constructed through a number of malicious nodes. But this case does not threaten the anonymity of the data traffic as was shown in [65], although it is a partial path hijacking. In any case, we only claim resilience to path hijacking, not immunity to it.

4.8 Proof of Correctness

Theorem 4.8.1 *SDAR is secured against passive and active attacks, but not against Denial-of-Service attacks.*

Proof:

1. SDAR provides protection against passive attacks. In the *path discovery* message, the identifier ID_S of the sender are encrypted with a one-time session key K_S. K_S and the identifier ID_R of the intended receiver are encrypted with the public key PK_R of the intended receiver. The one-time public key is used to encrypt the identities of intermediate nodes and the shared session keys. Thus, an adversary cannot find the real sender, receiver, or all intermediate nodes just by looking at the *path discovery* message. The same conclusions can be made for the *path reverse* message.

2. SDAR provides protection against active attacks like *replay attacks* and *message modification attacks*. Using the SEQ in the *path discovery* message, the receiver can discover a replayed *path discovery* message. Additionally, if some adversaries want to change the *path discovery* message or impersonate the sender or some intermediate nodes, the receiver can easily find out by verifying the signature since the sender and intermediate nodes have hashed the message, which is cumulative from its ancestor node, and signed the hash value in the *path discovery* message. The same conclusions can be made for the *path reverse* message.

3. A Denial-of-Service (DoS) attack would be a very dangerous attack on the protocol. The protocol itself does not provide a mechanism against this kind of attack. For instance a powerful adversary may simply flood the network with *path discovery* messages. Additionally, the small computational power on all wireless carry-on devices makes the protocol more vulnerable to this attack. This problem is common though to all the routing protocols for wireless ad-hoc networks.

Theorem 4.8.2 *SDAR maintains the anonymity of the sender and receiver.*

Proof:

1. During *path discovery* phase processing,

   - If all the neighboring nodes of the sender joined in collusion, they would know which message originally came from the sender and which message was just forwarded by the sender, i.e., they would find the sender but will not know who the intended receiver is.
   - If all the neighboring nodes of the target receiver were in collusion together, they would know which message terminated in the receiver and which message was just forwarded by the receiver, i.e., they would find the potential receiver but would not know the identity of the sender.
   - If some intermediate nodes were in collusion together, they would only know that the message was forwarded. They therefore cannot confirm which node is the sender and which node is the receiver.

2. During *path reverse* phase processing,

   - If all the neighboring nodes of the receiver joined in collusion, they would be able to determine who the receiver is. The collusion of all neighboring nodes can reveal the fact that the circled node is the node that started the *path reverse* phase, and hence it must be the intended receiver in the *path discovery* phase.
   - If all neighboring nodes of the sender were in collusion, they would be able to determine who the sender is. The situation is same as the above.
   - If some or all of the intermediate nodes were in collusion together, although they would know part of the path chain, they still would not be sure who the sender and receiver are since they would not know if the end node of the *path reverse* message is the sender or just another intermediate node, and the start node of the *path reverse* message is the receiver or just another intermediate node.

Theorem 4.8.3 *SDAR is able to identify malicious nodes and to avoid using them to establish routes.*

Proof:

Due to the nature of ad hoc wireless network and the bi-direction of the link between two wireless nodes, a node can be always monitored by its neighboring nodes. Whenever a node behaves adversely, its neighboring nodes will update their trust in the node. When a neighboring node receives later a request for a new path, it will avoid using the misbehaving node, and ensure that the path is established with only trusted nodes.

Theorem 4.8.4 *SDAR is able to establish a route matching certain trust requirement if enough nodes in requested trust level exist between the source and the destination.*

Proof:

During the *path discovery* phase, each intermediate node broadcasts the route request message to its neighboring nodes that satisfy the trust requirement set by the source node. These nodes in turn, pass the *path discovery* message to qualified neighboring nodes. Thus, starting from the source node, all the nodes in the requested trust level are passed the *path discovery* message, any path of qualified nodes, starting at the source and ending at the destination node will be searched and definitely be found.

Additionally, even if the source node sets the trust requirement for the route, each intermediate node is responsible for selecting its next hop on the path. Having each intermediate node select its next hop ensures that any malicious behavior by a node is observed by all neighboring nodes and will be reflected later during the route selection.

Chapter 5

Experimental Results and Performance Evaluation

We implemented SDAR protocol within the Network Simulator, NS-2[1] and carried out an extensive set of simulation experiments. During our experiments, we wish to be able to study the behavior of SDAR closely. In this chapter, we are going to present our simulation experiments and evaluate the performance of SDAR based on the experimental results. The performance evaluation on SDAR is done in two aspects: the overhead when compared with the basic version of DSR [40] and the effect from the change in the percentage of malicious nodes.

[1]NS-2 and its relevant information can be found in NS-2 official website "www.isi.edu/nsnam/ns"

5.1 Simulation Setup

The experimental results we have obtained are based on a wireless ad-hoc network deployed in a flat rectangle area of $670mx670m$. There are 30 mobile nodes randomly scattered in the network, which kept moving around according to the random waypoint mobility model [10]. Except for the experiments in Section 5.3.1 where we changed the speed value, the nodes moved at a maximum speed of $5m/s$ in a randomly chosen direction. The nodes had to pause for a pre-configured period of time, which we set up to $20s$, before they changed their moving direction.

We have used an IEEE 802.11 MAC layer and a 914MHz Lucent WaveLAN DSSS radio interface with the transmission range of $250m$ for each node. The traffic was generated by CBR sources over UDP protocol. Source nodes continuously generated data packets of 512 bytes at the rate of 4 packets per second in each flow. In addition, SDAR was simulated using 512 bit keys for authentication with the RSA algorithm, 64 bit community keys, and 64 bit session keys.

During our experiments, we found that the trust-based node classification would not be stable if the simulation was run less than $4000sec$, and that the simulation results would be irrelevant to simulation time if we run the simulation more then $4000sec$; therefore, we run each simulation five times, and each time, the simulation was run for $4000sec$ of simulated time.

In every simulation, half of the route requests had *High Trust Requirement* (HTR) for the intermediate nodes and the other half had *Low Trust Requirement* (LTR). The values of the *Medium Trust Level Threshold* (MTLT), δ_1, and the *High Trust Level Threshold* (HTLT), δ_2, were selected as 0.6 and 0.95.

5.2 Performance Metrics

We have evaluated our SDAR protocol using the following performance metrics:

- *Connectivity*:

  The *Connectivity* metric is defined as the percentage of the path request successfully answered by the destination, out of the route requests sent by the source.

- *Number of Packets*:

  In our simulations, we kept track of the number of data packets that were prepared to be sent, the number of data packets actually sent, and the number of data packets successfully received.

- *Routing Overhead*:

  The *Routing Overhead* metric represents the number of routing packets required to send one hundred data packets.

- *End-to-End Delay*:

  The *End-to-End Delay* metric is defined as the average time difference between the time a data packet is sent from the source and the time it is successfully received by the destination.

- *Average Route Length*:

  The *Average Route Length* metric indicates the average number of hops in a route.

- *Security Overhead*:

  The *Security Overhead* metric indicates the number of UPDATE messages sent by all nodes. These messages, as we mentioned in Section 4.3.2, are used to distribute community keys.

5.3 Experimental Results

In this section, we first investigate the overhead of SDAR routing protocol when compared with the DSR routing protocol [40]. Then, we present the effect from the change in the percentage of malicious nodes on the behavior of SDAR protocol.

5.3.1 SDAR and DSR – A Comparison

The reason why we chose the Dynamic Source Routing (DSR) routing protocol [40] stems from the fact that both DSR and SDAR protocols share the main routing idea. During the course of our simulations experiments, both protocols were run under identical mobility and traffic scenarios. We used a basic version of DSR, which does not include any optimization mechanisms, such as route caching, which, as one may expect, goes against the anonymity concept. This allowed us, also, to have a fair comparative study.

Figure 5.1 and Figure 5.2 show that SDAR protocol has a lower connectivity rate and a higher end-to-end delay. This variation is mainly due to the computational overhead of the security functions required for anonymously computing the path; intermediate nodes have to spend more time processing each single message, leading to a larger amount of waiting time and hence longer end-to-end delay.

5.3.2 The Effect From Malicious Nodes on SDAR

In the experiments we described in the previous section, we have mainly compared the security overhead incurred by SDAR, in comparison with the basic version of DSR [40]. In these experiments, we have also assumed that all the nodes in the network are well-behaved nodes. In this section, we will present the additional experiments we have conducted to determine the effect from the change in the percentage of malicious nodes on the behavior of SDAR protocol.

Figure 5.3 shows the change in the percentage of connectivity in relation to the change in the percentage of malicious nodes. The graph shows clearly that as the number of malicious nodes increases in the network, the percentage of successfully established route decreases. This is mainly due to the fact that the higher the percentage of malicious nodes, the higher probability that these nodes will drop the *path request* messages, leading to a lower connectivity rate. The graph shows also the difference between path requests with HTR and LTR for the intermediate nodes. The difference in the degree of connectivity between HTR and LTR is

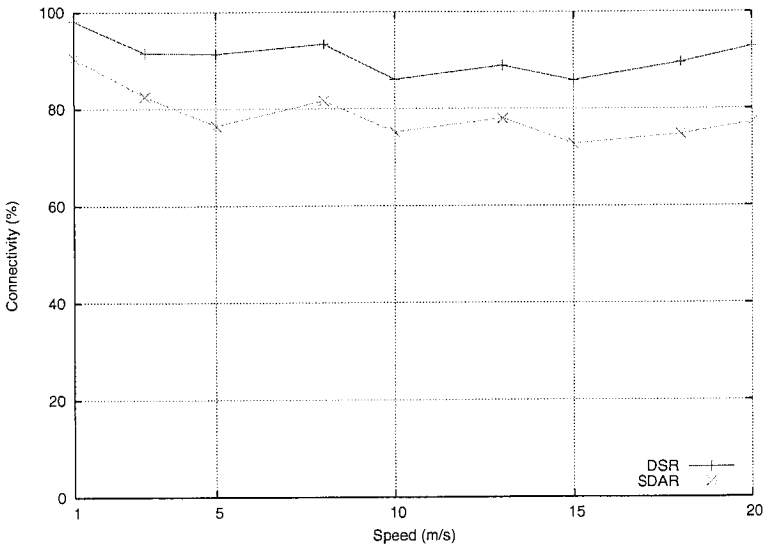


Figure 5.1: Connectivity of SDAR vs. DSR

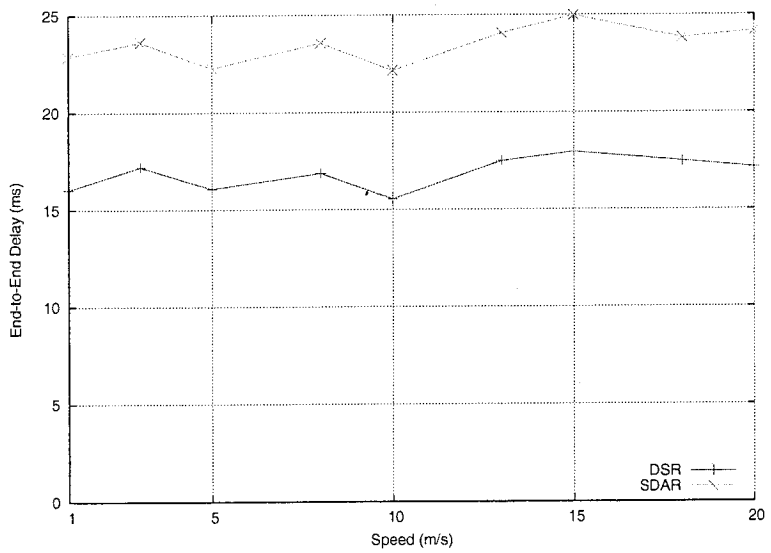


Figure 5.2: End-to-end delay of SDAR vs. DSR

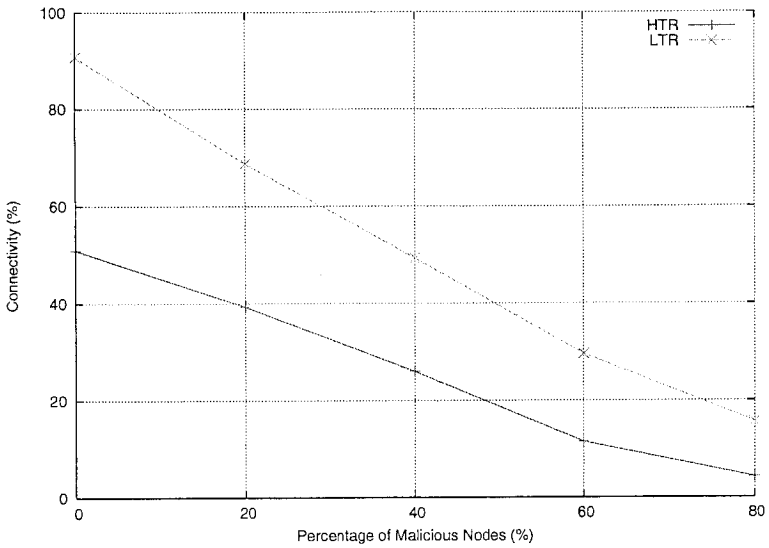


Figure 5.3: Connectivity vs. percentage of malicious nodes

mainly due to the higher trust requirement of HTR, which restricts the number of intermediate nodes that can participate in the route construction, leading to a lower connectivity; LTR route requests use intermediate nodes even if they have low trust value.

Figure 5.4 illustrates the routing overhead as a function of the percentage of malicious nodes for both HTR and LTR path requests. The graph shows that as the number of malicious nodes increases, the number of times the source node has to run the path discovery algorithm increases, and hence an increase in the routing overhead. This is also the result of intermediate malicious nodes dropping the path request or reply messages, resulting in an increase in the number of path requests. HTR has also higher routing overhead, since the probability of path requests is higher because of the restriction on the trust level of intermediate nodes, hence requiring a larger number of path requests.

The results in Figure 5.5 show the change in route length as a result of the change in the percentage of malicious nodes in the network, and also for HTR and LTR path requests. For both HTR and LTR, the average route length decreases with the increase of the percentage of malicious nodes. This can be explained by the fact that, with the fixed node density, the probability of constructing a path, which is composed of many nodes and is satisfying certain

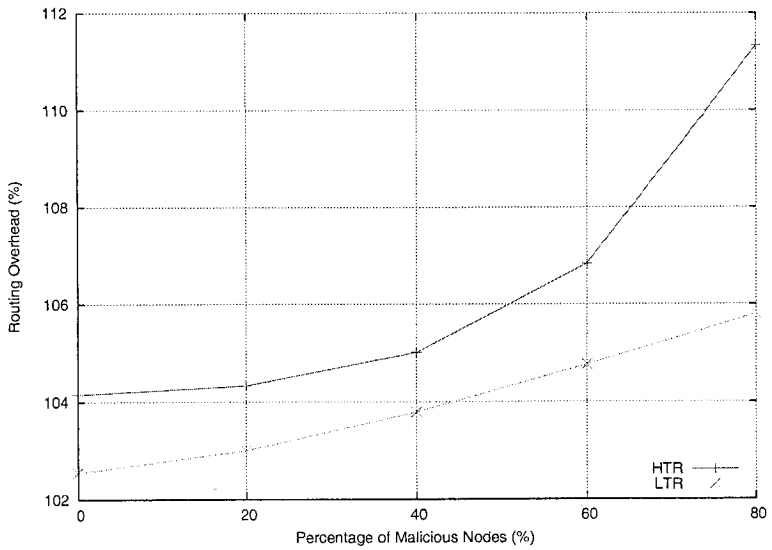


Figure 5.4: Routing overhead vs. percentage of malicious nodes

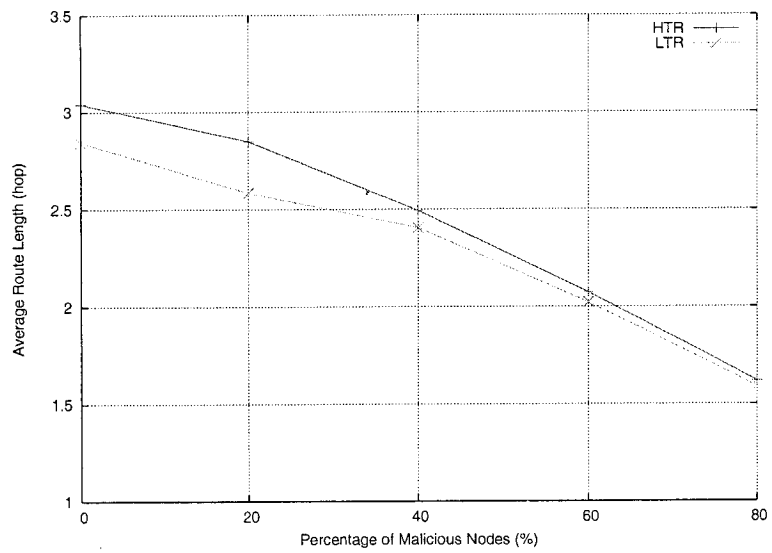


Figure 5.5: Average route length vs. percentage of malicious nodes.

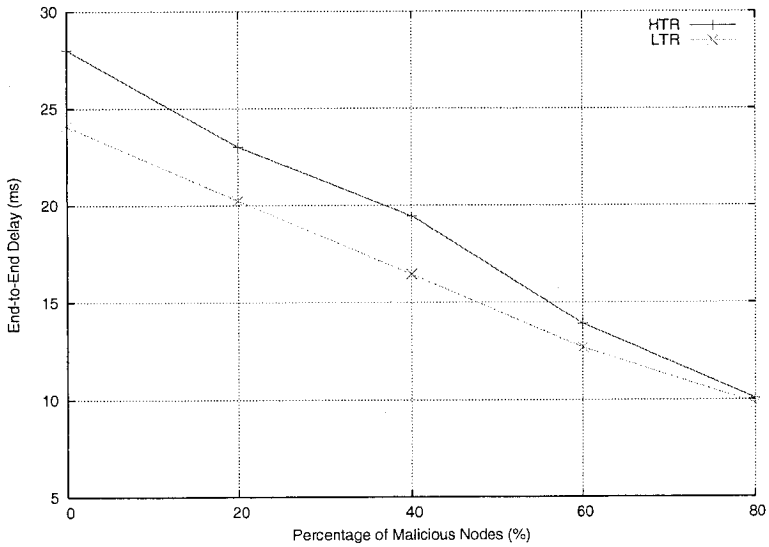


Figure 5.6: End-to-end delay vs. percentage of malicious nodes.

trust requirement, decreases as the percentage of malicious nodes increases. The graph shows also that when there are few malicious nodes in the network, the average path length for the HTR is higher than the average path length for the LTR. This is due mainly to the fact that the HTR path request adds more constraints on the path discovery algorithm, but since qualifying paths do exist between communicating nodes, the path would eventually be found, but it will be longer.

Figure 5.6 shows the end-to-end delay for the HTR and LTR, as a function of the percentage of malicious nodes. As explained in the previous paragraph, a when node density is fixed, higher percentage of malicious nodes means a shorter route between the source and the destination (if any), and hence short delay in the packet delivery. Additionally, since HTR routes are longer than LTR routes, this also means the packets going through HTR routes experience higher end-to-end delay than those using LTR routes.

Figure 5.7 and 5.8 illustrates the difference between the number of data packets prepared to be sent, the number of data packets actually sent, and the number of data packets that were received at the destination, as the percentage of malicious nodes in the network changes. As noticed from the graph, using the LTR generates a higher throughput, especially when the

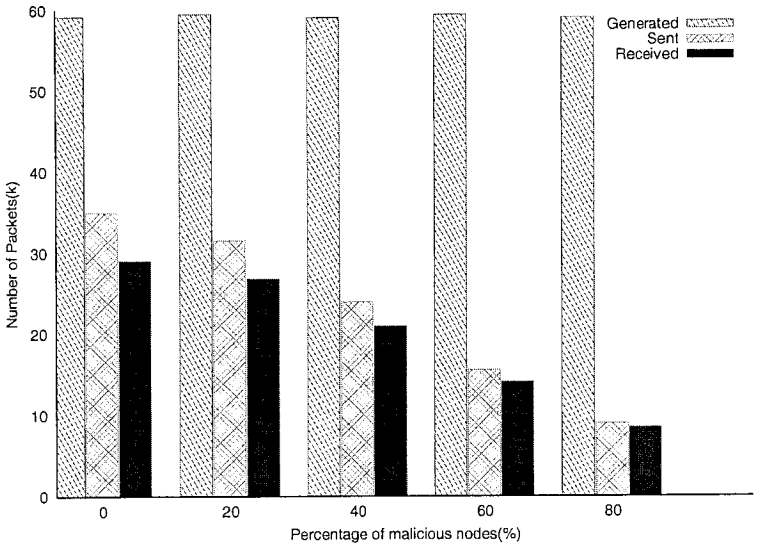


Figure 5.7: Number of packet for HTR vs. percentage of malicious nodes.

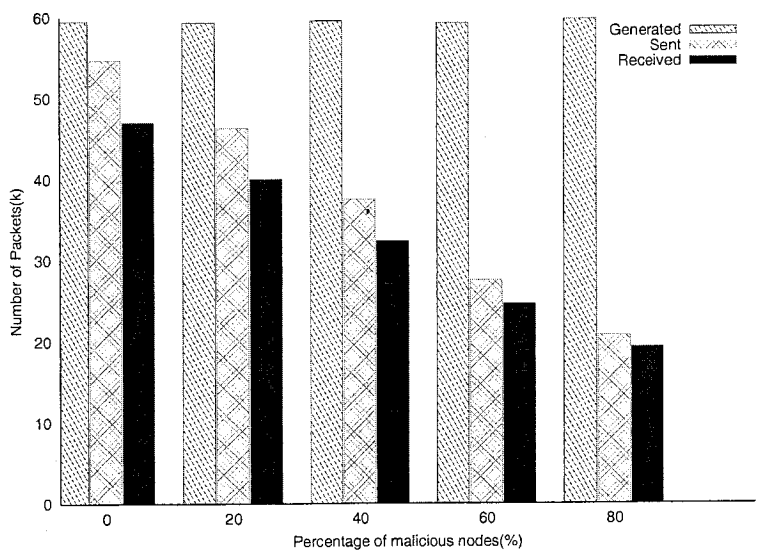


Figure 5.8: Number of packet for LTR vs. percentage of malicious nodes.

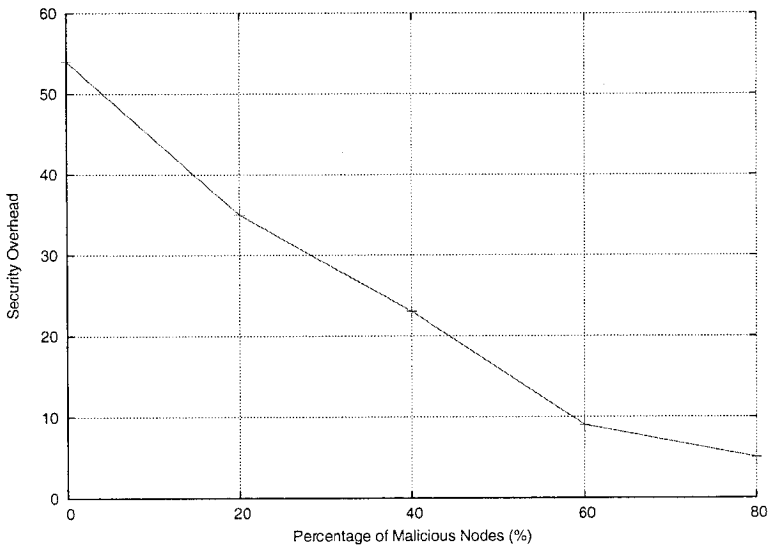


Figure 5.9: Security overhead vs. percentage of malicious nodes.

percentage of bad nodes is low; this is mainly due to the fact that all nodes are considered initially as malicious nodes, and their status is updated during the course of the simulation, thus causing less HTR data packets can be sent.

Figure 5.9 shows the number of exchanged UPDATE messages used to establish and update the trust relationship between neighboring wireless nodes. This number drops as the percentage of malicious nodes increases in the network. This number is expected to decline, since nodes send UPDATE messages only when the trust levels of their neighboring nodes change. In particular, the demotion of the trust levels of nodes cause more UPDATE messages. But as the percentage of malicious nodes increases in the network, more and more nodes remains in malicious status (recall "malicious" is default status), and thus fewer and fewer UPDATE messages are sent. The issue of collusion between malicious nodes is not included in our current work.

Chapter 6

An Agent-based Trust and Reputation Management Scheme

Rating nodes' trust and reputation have proven to be an effective approach to improve security, to support decision-making and to promote node collaboration in both wired and wireless networks. Although considerable research has been done on trust and reputation management, they focus mostly on trust and reputation modeling but seldom shed light on the overhead incurred by their proposed schemes. In this chapter, we propose a novel Agent-based Trust and Reputation Management scheme (ATRM) for WSNs only from a system design point of view. Our scheme is based on a clustered wireless sensor network with a backbone structure and on a mobile agent system providing trust and reputation management service. The scheme objective is to manage trust and reputation with minimal overhead in terms of extra messages and time delay. The main contribution of our work is the introduction of a localized trust and reputation management strategy, which makes trust aggregation and reputation acquisition quite straightforward without requiring network-wide flooding, and thereby reducing both communication cost and acquisition latency. Throughout the entirety of this chapter, we describe our scheme, prove its correctness, and give its initial performance evaluation.

6.1 Introduction

Trust, as an integrative component of human society, is an abstract matter that we are dealing with in our everyday life, but its relevant research is quite narrowed down, and therefore causing a lack of coherence in its definitions between different disciplines [50]. In computer science, there are many definitions and models for trust, such as the ones given in [1, 2, 26]. Based on the previous work as described in [1, 2, 26], the notion of trust to be used throughout this chapter is briefly defined as: *trust is the degree of belief about the future behavior of other entities, which is based on the one's the past experience with and observation of their actions*; and the properties of trust are summarized as: subjectivity [52, 68], non-transitivity [16], temporalness [4], contextualness and dynamicity as well as non-monotonicity [2]. Additionally, we can derive from subjectiveness and non-transitiveness another characteristic of trust, that is unidirection.

Reputation is another complex notion across multiple disciplines. It is quite different from but easily confused with trust. Although, in some research papers, e.g., [2, 11, 30] to mention just a few, authors attempt to define and model reputation in certain ways, most of them share a common agreement. That is, the basis of an entity's reputation is the trust the others hold in that entity. Likewise, reputation in the ATRM is defined as: *in a community, reputation of an entity is the global perception about the entity's behavior norms based on the trust that other entities hold in the entity.* In this definition, the word "global" implies that reputation is objective rather than subjective. Though some previous work [47, 53] points out that reputation should be subjective, we believe it is reasonable to consider that reputation is objectivity [31] since one's reputation is derived from *all the others'* opinions. Other characteristics of reputation include temporalness [47], contextualness [30, 53], and dynamicity as well as non-monotonicity.

Considerable research on trust and reputation management can be found in literature, and simulation results show that rating nodes' trust and reputation is an effective approach in distributed environments to improve security [18, 48, 63], to support decision-making [3, 42, 85],

and to promote node collaboration [12, 33, 51, 58]. Nowadays, trust and reputation management is attracting more and more attention from researchers and becoming an essential part of many network-based applications. A solution to trust and reputation management problem generally consists of both modeling and system design. In brief, modeling is the process of converting trust and reputation from an abstract concept to a mathematical expression, while system design is the process during which the following questions are answered: how the trust and reputation system works, what components the system consists of, and how these components cooperate. Obviously, the system design depends on the trust and reputation model used as well as the underlying network model. Thus, there is no "one-fit-all" trust and reputation management system. In our work, we focus our interest only on the system design part of the solution of the trust and reputation management problem in a particular type of networks, wireless sensor networks (WSNs), where bandwidth and energy are highly restricted.

Searching nodes' reputation in a network with a central authority is not difficult. The absence of any centralized authority in wireless sensor networks makes it challenging to trace nodes' reputation accurately. Flooding the network with request messages is a useful tool for data-searching in a fully distributed environment. However, since message transfer consumes both bandwidth and energy, the trust and reputation management schemes causing large amounts of traffic by flooding the network with request messages are not desirable in wireless sensor networks known for their bandwidth and energy constraints. In addition, because trust and reputation information are usually requested by nodes before they start communicating with each others, the trust and reputation management schemes with poor trust and reputation acquisition latency are not acceptable in situations with strict real-time requirements, like battle fields and emergency rescue. Under these circumstances, we propose a novel Agent-based Trust and Reputation Management scheme (ATRM) for wireless sensor networks; the main objective of our scheme is to manage trust and reputation with minimal overhead in terms of extra messages and time delay. The ATRM is based on a clustered WSN with backbone and on a mobile agent system; it introduces a trust and reputation local management strategy with

the help from the mobile agents running on each node. That is, a node's trust and reputation information is stored on the node itself and managed by the local mobile agent of the node. Benefiting from the local management of trust and reputation, no centralized repositories are needed, and nodes themselves are able to provide their own reputation information whenever requested; therefore, reputation propagation requires neither network-wide flooding nor long acquisition-latency.

The remainder of this chapter is organized as follows: Section 6.2 briefly introduces the background knowledge about wireless sensor networks, hierarchical wireless networks and mobile agent systems. Section 6.3 reviews some related work on trust and reputation management in literature. Section 6.4 gives a detailed presentation of the ATRM. Section 6.5 proves the correctness of the ATRM. Section 6.6 theoretically analyzes the complexity of the ATRM. Section 6.7 shows some preliminary experimental results of the ATRM.

6.2 Background

In this section, we are going to present some background knowledge about wireless sensor networks, hierarchical wireless networks and mobile agent systems.

6.2.1 Wireless Sensor Networks

Wireless sensors are small and cheap devices powered by low-energy batteries, equipped with radio transceivers, and responsible for responding to physical stimuli, such as pressure, magnetism and motion, by producing radio signals. They are featured with resource (e.g., power, storage, and computation capacity) constraints and low transmission rate. Wireless sensor networks (WSNs) are collections of such wireless sensors deployed (e.g., using aircraft) in strategic areas to gather data about the changes in their surroundings and to report these changes to a data-processing center (which is also called data sink). The processing center can be a specialized device or just one of the sensors, and its function is to analyzes the collected

data to determine the characteristics of the environment or to detect events. Mass-produced intelligent sensors and pervasive networking technology enable WSNs to be widely applied to various applications ranging from military to civilian fields; the examples of these applications include target tracking and traffic monitoring.

WSNs may experience topological change although nodes are fixed; this is because nodes' components and radios are usually turned OFF and ON from time to time to reduce node interaction and to prolong network lifetime, and also because nodes are very likely to be destroyed under harsh environmental conditions. In addition, differing from mobile ad-hoc networks (MANETs), WSNs are characterized with the mentioned dense deployment. They usually consist of thousands of nodes densely scattered over the monitored areas. The dense deployment causes a high degree of interaction between nodes, which in turn consumes more battery energy and complicates networking protocols.

6.2.2 Hierarchical Wireless Networks

For a wireless network with n nodes capable of transmitting at Wbits/sec, according to [32], the throughput, T, for each node under optimal conditions is

$$T = \Theta(\frac{W}{\sqrt{n}}) \tag{6.1}$$

According to the above formula, the number of nodes should be kept as small as possible in order to gain efficient throughput performance. But in WSNs, there are usually thousands of nodes, and scalability is therefore an important issue.

Clustering is an effective approach to improve network scalability and has been studied by a number of researchers, and many clustering algorithms, e.g., [5, 6, 28, 46], are proposed for wireless networks. By clustering algorithms, nodes are grouped into some clusters, and within every cluster, a node is elected as a cluster head. Cluster heads together form a higher-level network, upon which clustering could again be applied. In this way, a multi-level structure is constructed in the network after recursive clustering. This structure facilitates communication

and makes it possible to restrict bandwidth-consuming network operations like flooding only to intended clusters. This can lead to reduced network traffic and hence improved network scalability. When cluster heads have the same communication capability as regular nodes, communication between cluster heads has to be done through a sequence of intermediate nodes, and thus the improvement in scalability can be limited. This problem has led to the proposal of mobile backbone networks, which will be introduced in the following paragraphs.

The main difference between traditional clustered wireless networks and mobile backbone networks is the application of radios of different ranges. In mobile backbone networks [86], cluster heads are required to be pre-deployed backbone nodes, which are capable of communicating through radios of different range and are connected directly through long-range radios instead of by message relay. The nodes in the same level share the same communication channel, but radios in different levels use different frequency and channel resources. Nodes communicate only with other nodes belonging to the same cluster, and inter-cluster communication has to be done through cluster heads in the higher-level backbone network. Using this approach, each cluster can be considered an independent small-sized wireless network, and the higher-level backbone network is another wireless network linking the clusters together.

Assume that there is a two-level mobile backbone network where the number of nodes is n and the number of clusters is m. Furthermore, denote by W_1 the transmission rate in the lower-level network and by W_2 the transmission rate in the upper-level network. Then, the number of nodes in each cluster is n/m on average, and by Formula 6.1, the per node throughput of clusters (in the lower-level) $T_1 = \Theta(\frac{W_1}{\sqrt{n/m}})$, and the per node throughput of the backbone network (in the upper-level) $T_2 = \Theta(\frac{W_2}{\sqrt{m}})$. When n is fixed, it is clear that T_1 increases with the growth of m while T_2 decreases, and thus it is necessary to determine the optimal m which makes the mobile backbone network achieve optimal overall throughput. Since cluster heads are required to be backbone nodes, the optimal m implies the optimal number, referred to as M, of backbone nodes needed. According to [86], M is computed as:

$$M = \frac{W_2}{W_1}\sqrt{n} \tag{6.2}$$

Since this result is achieved under optimal conditions, the number of backbone nodes is actually less than it is in the real world. A backbone network construction algorithm RCC, and its performance evaluation is presented in [86].

6.2.3 Mobile Agent Systems

Mobile agents are the programs launched by network users to accomplish certain given tasks while migrating from one computation environment (a computer) to another in networks. They are featured with autonomy, asynchrony, adaptivity and communicability. After a mobile agent lands on a computer, it accesses the local resources, interacts with the local execution environment, senses the dynamic changes of the network and acts accordingly, following the self-contained rules, and if necessary, it can communicate with other mobile agents by means of messages to collaboratively achieve a common goal. Since the required data is locally accessed and computation logic is encapsulated inside mobile agents, the transfer of control and data messages is less necessary, and thus, the possibility of network congestion decreases. Using mobile agents to access distributed data creates an efficient method for data distribution, aggregation and sharing in distributed network environments with bandwidth constraints. Mobile agents can be widely employed in many fields like electronic commerce, secure brokering and distributed information retrieval and telecommunication network services.

The future of mobile agents is so promising that a large number of researchers are attracted to it. So far, a lot of research activities on mobile agents have been carried out, and many mobile agent systems, such as Grasshopper, Anima, Odyssey and Planet, have already been developed by different institutes and companies. However, security and fault-tolerance problems always remain, since mobile agents are designed to run on remote computers and to travel over large-scale networks. On the one hand, mobile agents' computation logic and accumulated data both risk the attacks from their host computers, and on the other hand, mobile

agents roaming around in the network can be used by their hosts through embedding malicious codes to assault other machines. Therefore, mobile agent systems should be secured against unauthorized analysis and modification. Additionally, since both node and communication failure are common phenomena, mobile agents are very likely to be lost during their trips; due to the software faults of mobile agents or of their execution environment, mobile agents may crash or have an incorrect status. In these cases, mobile agent systems should be able to detect the loss and failure of mobile agents, to recover mobile agents from incorrect status, and to ensure mobile agents' arrival at their destination. Although many security mechanisms [73, 76, 79, 80] and fault-tolerance models [13, 64, 72, 75] have been proposed and developed for mobile agents systems, there are still a number of issues that need to be addressed before having a robust mobile agent system.

6.3 Literature Review on Trust and Reputation Management Systems

Early-existing systems dealt only with certain aspects of trust management, for example, PKI X.509 [35] and PGP [90] are designed to solve the problem of finding a suitably trustworthy copy of the public key of someone. After Blaze, Feigenbaum, and Lacy formally introduced "trust management" as a separate component of security in network services and gave an overall definition of trust management problem in [8], a great deal of comprehensive research focusing especially on trust and reputation management follows. Trust and Reputation systems can be classified as centralized or decentralized. However, we are going to review decentralized systems.

Blaze, Feigenbaum, and Lacy [8] proposed a unified decentralized trust management system, PolicyMaker, based on a simple language for describing security policies, credentials and relationships. PolicyMaker works like a database engine. Its basic function is to process queries, each of which is a request to determine if a particular public key or a set of particular

public keys are permitted to perform a particular action according to local policy. A PolicyMaker query has the following form: $key_1, key_2, ..., key_n$ REQUESTS *ActionString*, where *ActionString* describes a proposed trusted action and is application-specific. Depending on processing results, PolicyMaker may accept/reject the requested actions or return additional suggestion to make the requested actions acceptable. As we can see, PolicyMaker does not require a mapping between public keys and keyholders' IDs, so it quite meets the anonymity requirement (if any). Similar systems include KeyNote [7], RT [45] and TPL [34].

Gupta, Judge, and Ammar [31] proposed a partially distributed reputation system for P2P networks. This system involves a central reputation computation agent (RCA) which is presumably trusted by every other nodes. In order to secure reputation computation and to prevent nodes' modification to their locally-stored reputation information, public-key cryptography is utilized. The RCA's public key is required to be available to every node in the network, and each node generates a (PublicKey, PrivateKey) pair and registers it with the RCA. The digest of the public key of a node is used by the RCA as the node's identification. A node's reputation computed by the RCA is encrypted with RCA's private key and locally stored by the node itself. The RCA does not participate in nodes' transaction process, but it is periodically requested by nodes for credits for their contribution in the system based on two proposed reputation computation schemes, Debit-Credit (DCRC) and Credit-Only (CORC).

Kamvar, Schlosser, and Garcia-Molina [42] proposed a reputation management algorithm, called EigenTrust, for P2P networks. In EigenTrust, there is a distinction between a node's local trust and its global reputation. For any two nodes n_i and n_j, the local trust n_i holds in n_j is normalized as $\frac{max(s_{ij},0)}{\sum_j max(s_{ij},0)}$ where s_{ij} represents the local trust rating depending on the Quality of Service (QoS) n_j provided. The reputation of n_j is based on the local trust values, which are assigned to n_j by other nodes (e.g., node n_i) and weighted according to the trust of the assigning nodes. Trust aggregation is done using a transitive trust mechanism: a requester asks its friends for their opinions about the evaluated node, and then it asks for the opinions of its friends' friends, and then it asks for the opinions of its friends' friends' friends, and so on.

This kind of asking process ends after a pre-determined number of iterations.

Jurca and Faltings [41] proposed an incentive-compatible reputation mechanism, which introduces a side-payment scheme and cryptographically protects the integrity of reputation information. Side payments are organized through a set of always-existing broker agents, named R-agents, which sell reputation information to and buy reputation information from agents. The payoff for an agent selling reputation information to an R-agent depends on whether the provided information is consistent with the future reports on the same agent or not. Two agents that are about to trade select by a certain algorithm an R-agent, ask the selected R-agent for the reputation of their partners, and pay for the reputation information. An agent that loses all its reputation money can not use the reputation service any more. Having known the reputation of their partners, agents will decide if they should have the transaction with their partners. After the transaction is over, agents get transaction payoff from which they can exactly determine the behavior of their partners and submit a report to the selected R-agent. Based on the outcome of the transaction, agents also update their opinions about the effectiveness of different R-agents.

Taking into account the dynamicity of MANETs, Ren and his colleagues [68] proposed a certificate-based trust establishment scheme. They addressed the trust establishment with the dynamically joining and leaving node under the assumption of existing sparse social relationships among nodes. The scheme requires a bootstrapping phase during which a secret dealer is required. In the first part of the bootstrapping phase, every member node obtains its secret short list from the secret dealer, which includes k bindings of a node identifier and its corresponding public key. The value of k is carefully chosen depending on the group size and may vary slightly from node to node. The second part of the bootstrapping phase is a certificate issuance process. All the member nodes generate certificates for the received bindings from their own domain and store the certificates locally. After the bootstrapping phase, network becomes fully functional with no need of the secure dealer, and the trust establishment for any two nodes turns into a certificate chain detection problem in a certificate graph.

Theodorakopoulos and Barasa [81] proposed a trust evaluation mechanism for MANETs without the need of centralized infrastructure (e.g., the secure dealer in [68]). In their mechanism, trust evaluation is viewed as a generalized shortest path problem on a weighted directed graph where vertices represent nodes and weighted edges correspond to the *opinions* that one end node has about the other end node in the edge direction. *Opinions* are assigned to edges by nodes based on their local observation and on their own criteria. Every *opinion* consists of two values, trust value and confidence value. The former is a node's trust estimation while the latter reflects the accuracy of the trust value assignment. In such a graph, an indirect trust relation without previous immediate experience is established by the theory of semirings.

Buchegger and Boudec [11] presented a reputation system for MANETs and P2P networks, which is featured with the elimination of the affection on reputation from incompatible recommendation. In their system, each node n_i maintains three types of data about everybody else, e.g., node n_j, that it cares about. These three types of data are the trust rating T_{ij}, the reputation rating R_{ij} and the first-hand observation summary F_{ij}. T_{ij} represents n_i's confidence in how possible it is that the first-hand observation provided by n_j is true. R_{ij} reflects n_i's opinion about how trustworthy n_j is in transactions. F_{ij} is a summary record of n_i's immediate observation about n_j. When n_i makes first-hand observation about n_j's behavior, R_{ij} and F_{ij} are updated accordingly. And, n_i periodically publishes F_{ij} as its recommendation to n_j. When n_i receives another node n_k's first-hand information about n_j, F_{kj}, it may or may not update R_{ij} based on T_{ik} (i.e., if n_k is trustworthy enough) and R_{ij} (i.e., how different R_{ij} and F_{kj} are) by a modified Bayesian approach and on a linear model merging heuristic. According to the decision, T_{ik} is also updated. In this algorithm, only first-hand information is published, and in order to reduce traffic, the publication is confined to a subset of nodes.

Liu and Issarny [47] introduced another reputation mechanism for MANETs, where service reputation and recommendation reputation is differentiated. By using recommendation reputation, this mechanism is able to distinguish truth-telling and lying nodes and thus to be resilient against attacks of defame and collusion. Reputation is managed distributedly by an

experience manager, a recommendation manager and a reputation manager running on each node. The experience manager takes the responsibility to store the previous experiences with other nodes. The recommendation manager is in charge of storing other nodes' recommendations, exchanging reputation information with other nodes, and managing a table of recommendation reputation of recommenders. The reputation manager computes and manages the service reputation of a node, referring to the inputs from both the experience manager and the recommendation manager. Periodically, the recommendation manager contacts with other nodes for reputation information exchange. If a recommendation manager receives a recommendation exchange request from another node, it may accept or reject the request depending on if the requester's recommendation reputation is above a pre-defined threshold value.

Some proposed schemes like [8, 31, 41, 42, 90] are designed originally for wired networks (e.g., P2P) and thus not suitable for wireless sensor networks where nodes have resource constraints. The ones such as [11, 47, 68, 81] are developed for wireless networks, but they focus mostly on trust and reputation modeling and seldom emphasize the network performance problem. The commonly used approaches by these schemes to acquire reputation can be classified as two groups, on-demand and periodical. As trust information is distributed into the entire network, reputation computation requires network-wide flooding for trust aggregation. Whether reputation computation is periodical or on-demand, it will results in packet collision and energy loss. Although the scheme presented in [11] restricts flooding to a subnet of nodes, how to determine the subset such that it covers all the nodes (or a sufficient number of nodes) holding the required trust information becomes a problem.

6.4 The Proposed Scheme ATRM

In this section, we propose a novel Agent-based Trust and Reputation Management scheme (ATRM) for WSNs. The main objective of the ATRM is to effectively manage trust and reputation with minimal overhead in terms of extra messages and time delay. The notations to

be used for describing the ATRM can be found in Table 6.1.

6.4.1 Overview

The ATRM is based on a clustered WSN with backbone, and its core is a mobile agent system. Differing from traditional trust and reputation management systems, ATRM requires that a node's trust and reputation information be stored respectively in the forms of t-Instrument and r-Certificate by the node itself. Obviously, nodes can not manage and compute their own

Name	Description
n_i	An arbitrary node in the original network
n'_j	An arbitrary node in the backbone network
$C_\star$	An arbitrary context
$ID(n_i)$	The ID of node n_i
TRA	A trust and reputation assessor agent
AK	The symmetric key held by a TRA
AL	The agent launcher
t-Instrument	A trust instrument
r-Certificate	A reputation certificate
$Tran_\star$	An arbitrary transaction
n_{req}	The requester of transaction $Tran_\star$
n_{pro}	The provider of transaction $Tran_\star$
Tab_{eval}	Trust evaluation table
Tab_{instr}	t-Instrument table
Buf_{cert}	r-Certificate buffer
$CNTER$	Received *LowVersion* message counter
$t_{i,j}$	The trust evaluation made by n_i on n_j
$H(M)$	The hash value of the message M
$E_{AK}(M)$	The Message M is encrypted with AK
$TI(n_i, n_j, C_\star)$	The t-Instrument issued by n_i to n_j for $C_\star$
$RC(n_i)$	The r-Certificate of node n_i
δ_{cert}	r-Certificate acquisition timeout
δ_{instr}	t-Instrument issuance timeout
δ_{valid}	The lifetime of a trust evaluation
θ_{cnt}	The threshold value of $CNTER$
θ_{ack}	The maximum number of acknowledgment retries
θ_{cert}	The maximum number of r-Certificate acquisition retries
θ_{instr}	The maximum number of t-Instrument issuance retries

Table 6.1: Notations for ATRM

trust and reputation. So, ATRM further requires that every node locally hold a mobile agent which is in charge of administrating the trust and reputation of its hosting node. In this sense, mobile agents provide nodes a "one-to-one" trust and reputation management service.

In ATRM, an arbitrary transaction is defined as the process of interaction between two nodes, the *requester* and the *provider*, and it is triggered by the *requester* and may be accepted/rejected by the *provider*. Before starting any transaction, the *requester* asks its local mobile agent to obtain the *r*-Certificate of the *provider* by directly querying the *provider*'s local mobile agent. Based on the *provider*'s *r*-Certificate, the *requester* decides whether to start the transaction or not. After a transaction is finished, the *requester* makes a trust evaluation on the *provider* according to the QoS it gets from the *provider* during the transaction, and then it submits the evaluation to its local mobile agent which then accordingly generates a *t*-Instrument for the *provider* and sends the *t*-Instrument to the *provider*'s local mobile agent. Using its collected *t*-Instruments, a mobile agent periodically issues its hosting node *r*-Certificates.

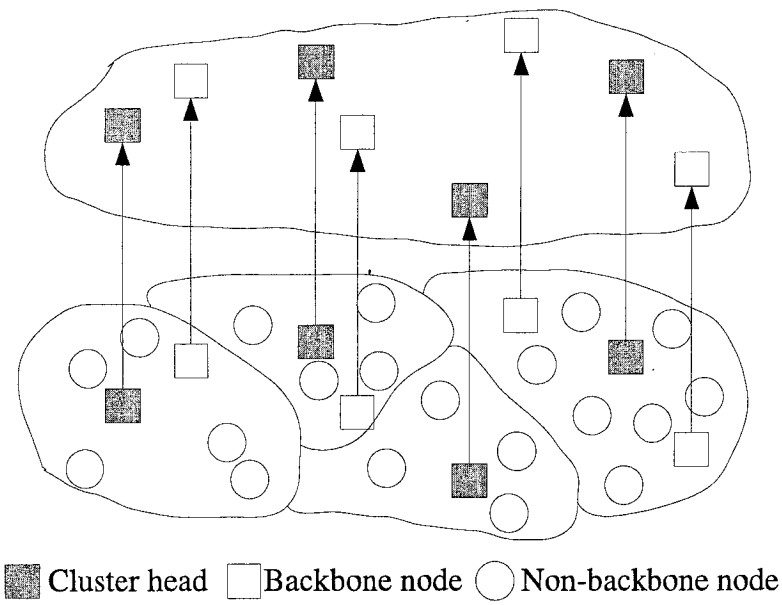


Figure 6.1: A clustered WSN with backbone

6.4.2 Network Model

Sufficient backbone nodes with multiple long-range radios are randomly scattered in the WSN. Every node has a unique ID by which it can be distinguished from others. Through a secure routing protocol, all the nodes (including backbone nodes) together compose a low-level network using a short-range radio, which is referred to as the *original network*. Likewise, the backbone nodes additionally constitute a high-level network, *backbone network*, via a long-range radio. The original network is dynamically partitioned by an effective clustering algorithm running on every node into a number of clusters, each of which has a backbone node elected as cluster head. Figure 6.1 shows an example of such 2-level WSNs. Both backbone and non-backbone nodes may fail or become unavailable due to system crash, power exhaust or any other reason, however, the rest of the original network and backbone network is still connected.

6.4.3 Assumptions

Since mobile agents are designed to travel over the entire network and run on remote nodes, they must be launched by trusted entities. And clearly, compromised mobile agents may not provide expected services and can actually constitute a threat to network security. Therefore, in ATRM, we assume 1) that there is a trusted authority which is responsible for generating and launching mobile agents and 2) that mobile agents are resilient against unauthorized analysis and modification towards their computation logic.

6.4.4 System Architecture

ATRM consists of four key components, which are an Agent Launcher (AL), Trust and Reputation Assessors (TRAs), Trust Instruments (t-Instruments), Reputation Certificates (r-Certificates). We are going to introduce these four components in this section.

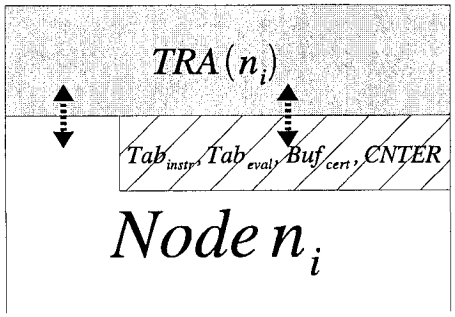


Figure 6.2: A node and its local replica TRA

Agent Launcher (AL)

An AL is an authority responsible for generating and launching TRAs into the network. It may be a piece of software, a node or an organization, and it could be either inside or outside the network. In ATRM, we assume there is only one AL that is always-existing and trusted by every node. The AL launches one TRA each time in a broadcast fashion into the backbone network. Before launching a TRA, the AL associates it with a symmetric secret key AK and a monotonically increased version number (starting from 1). The purpose of AK is to secure trust aggregation and reputation propagation, while the version number is used to support agent consistency verification. In case the AK of the current *TRA* is stolen or broken, the AL may periodically launch a new TRA with a higher version number and a fresh AK to replace old ones according to application-specific security requirements.

Trust and Reputation Assessor (TRA)

A TRA is a mobile agent generated by the AL. It is designed to be distributed into every node and to provide its hosting node with trust and reputation management service. Each node will hold a replica of the TRA of current version. For an arbitrary node n_i, its replica TRA, $TRA(n_i)$, locally maintains four data structures, i.e., a trust evaluation table Tab_{eval}, a t-Instrument table Tab_{instr}, a r-Certificate buffer Buf_{cert}, and a *LowVersion* message counter $CNTER$. The trust evaluations that n_i recently made on other nodes are kept in Tab_{val}, while the t-Instruments issued to n_i by the local replica TRAs of other nodes are stored in Tab_{instr}.

Buf_{cert} accommodates n_i's r-Certificate last issued by $TRA(n_i)$. As for $CNTER$, it is incremented whenever $TRA(n_i)$ receives a *LowVersion* message from a node for the first time since the last $CNTER$ resetting. The relation between a node and its local TRA is shown in Figure 6.2, where the dashed lines with double arrows represent the interaction between the TRA and its host.

As illustrated in Table 6.2, Tab_{eval} is composed of four fields, ID, CTX, $EVAL$, and TS, among which ID and CTX together constitute the primary key of the table. Field ID contains the IDs of the evaluated nodes; field CTX implies trust contexts; field $EVAL$ stores the trust evaluation values; field TS holds the time when evaluations are made. For any node n_j, its entry for context $C_\star$ in the Tab_{eval} of node n_i is denoted by $Entry_{eval}(n_j, C_\star)$.

ID*	CTX*	EVAL	TS
$ID(n_j)$	$C_\star$	t_{ij}	T

Table 6.2: The Tab_{eval} of node n_i

Table 6.3 shows the structure of the Tab_{instr} of n_i. Tab_{instr} contains five fields, including ID, CTX, $INSTR$, TS and ACK, of which and ID and CTX together constitute the primary key of the table. Field ID contains the IDs of t-Instrument issuers; field CTX implies trust contexts; field $INSTR$ stores t-Instruments. Field TS holds the time when t-Instruments are issued; field ACK reflects how may times a t-Instrument issuance is acknowledged, and its default value is 1. For any node n_j, its entry for context $C_\star$ in the Tab_{instr} of node n_i is denoted by $Entry_{instr}(n_j, C_\star)$.

ID*	CTX*	INSTR	TS	ACK
$ID(n_j)$	$C_\star$	$TI(n_j, n_i, C_\star)$	T	1

Table 6.3: The Tab_{instr} of node n_i

A replica TRA stays on its host until it is replaced by the replica of a higher-version TRA, and in the meantime, it offers its host the trust and reputation management service. When TRA replacement takes place, the new local TRA will take over all the data structures maintained

by the old one and reset $CNTER$ to 0. Detailed description about TRA's functionality can be found in Section 6.4.5.

Trust Instruments (t-Instruments)

A t-Instrument is a segment of data organized with a special structure and issued by the local replica TRA of a node (*issuer*) to another node (*issuee*). It is stored in the Tab_{instr} on its issuee node. Considering any two nodes n_i and n_j, the t-Instrument issued by $TRA(n_i)$ to n_j under context $C_\star$ is defined as:

$$TI(n_i, n_j, C_\star) = E_{AK}(D, H(D)) \tag{6.3}$$

where $D = (ID(n_i), ID(n_j), C_\star, T, t_{i,j})$ and T is a timestamp implying the time when the t-Instrument is issued. By above definition, we can see that a t-Instrument implicitly indicates the temporalness property of trust by the use of a timestamp T. t-Instrument issuance is driven by transactions, and it involves message transmission between the local replica TRAs of *issuers* and *issuees*. A more detailed description about t-Instrument issuance can be found in Section 6.4.5.

Reputation Certificates (r-Certificates)

A r-Certificate is a segment of data organized with a special structure and issued by a replica TRA to its host. A node's r-Certificate is locally stored in the Buf_{cert} on the node and periodically refreshed by the local replica TRA. If there are k concerned contexts, for any node n_i, its r-Certificate is defined as

$$RC(n_i) = E_{AK}(R, H(R)) \tag{6.4}$$

where $R = (ID(n_i), T, ((r_1, C_1), (r_2, C_2), \cdots, (r_k, C_k)))$ and T is a timestamp implying the time when the r-Certificate is issued. This formula is explained as follows: n_i's reputation is r_1

under context C_1, r_2 under the context C_2, $\cdots$, and r_k under context C_k at time point T. This definition implicitly reflects the contextualness and temporalness properties of reputation. Description about r-Certificate acquisition and issuance can be found in Section 6.4.5.

6.4.5 How It Works

The execution of ATRM involves two phases, i.e., the *network initialization* phase and the *service-offering* phase. As soon as ATRM starts, the *network initialization* phase is initiated. The purpose of this phase is to distribute a TRA to every node. What follows is the *service-offering* phase during which the trust and reputation service is provided. In this subsection, we are going to go through the detail of these two phases.

Network Initialization Phase

The *network initialization* phase consists of two stages. In the first stage, the AL launches a TRA in the backbone network in a broadcast fashion. Considering an arbitrary node n'_j in the backbone network, when it receives a TRA for the first time, n'_j makes a replication of the TRA and then forwards the TRA to all its immediate neighbors in the backbone network. If n'_j receives an already-received TRA, it just discards the TRA and pretends nothing happened. Once n'_j has a replica TRA, it enters the second stage. In the second stage, n'_j checks if it itself is a cluster head. If so, n'_j broadcasts its replica TRA within its cluster in G to distribute the replica TRA to all its cluster members, or keeps silent otherwise. The *network initialization* phase is run at the beginning of the execution of ATRM, and it may also be re-run later from time to time to update replica TRAs depending on application-specific security requirements.

Service-offering Phase

As long as a node has a local replica TRA, the trust and reputation service provided by the replica TRA is available to the node, and thus we say that the node is in the *service-offering* phase. The trust and reputation service is composed mainly of four types of sub-services,

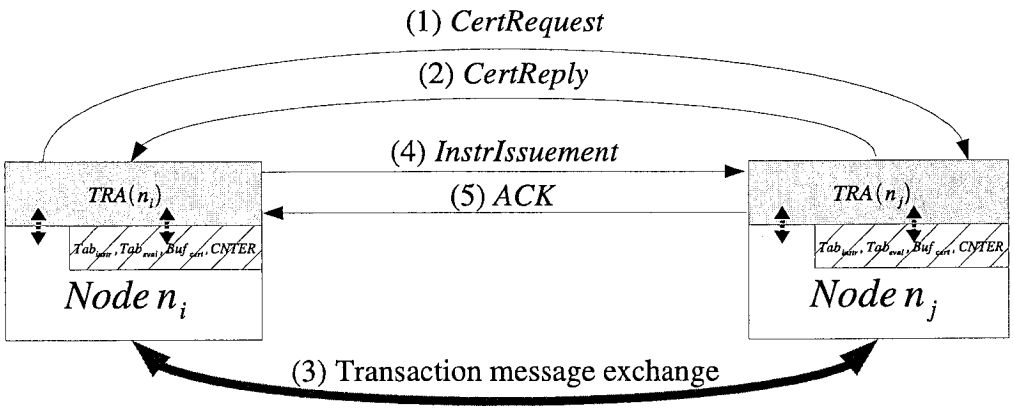


Figure 6.3: The interaction between TRAs in the service-offering phase

r-Certificate acquisition, *t-Instrument issuance*, *r-Certificate issuance* and *trust management routine*. The first two sub-services are transaction-driven and involve the message transmission between transaction *requester* and *provider*, whereas, the other two sub-services involve merely local processing periodically performed by the replica TRA of each node.

Because of the asynchronous execution, nodes are unlikely to enter the *service-offering* phase at the same time. Namely, it is possible that some nodes do not yet have a local replica TRA or have an inconsistent version of TRA when they are requested by other nodes for *r*-Certificates. The inconsistency of two replica TRAs means that the two replica TRAs derive from different versions of the TRA and thus have distinct secret keys. Inconsistent replica TRAs are not able to correctly process the *t*-Instruments and *r*-Certificates issued by each other. Clearly, the asynchronous execution may lead to the failure of the trust and reputation management service. In the following paragraphs, we are going to exploit how the four types of sub-services are delivered in spite of the presence of failures.

***r*-Certificate Acquisition** is initiated by $TRA(n_{req})$ in the pre-transaction process just before the official start of $Tran_\star$. Its objective is to obtain the reputation of n_{pro}, based on which n_{req} is able to decide whether to really start $Tran_\star$ with n_{pro}. Because the *r*-Certificate reflecting n_{pro}'s reputation is locally stored by n_{pro} and managed by $TRA(n_{pro})$, n_{req} just commissions its own replica TRA, $TRA(n_{req})$, to directly ask $TRA(n_{pro})$ for n_{pro}'s *r*-Certificate. In Figure 6.3,

Step (1) and (2) indicate the process of r-Certificate acquisition happening between n_{req} and n_{pro}. The detail of r-Certificate acquisition is shown below:

At n_{req} side

1. $TRA(n_{req})$ sends a *CertRequest* message carrying its version number to $TRA(n_{pro})$ and then expects a reply message from it.

2. If $TRA(n_{req})$ does not get any reply from $TRA(n_{pro})$ during a given period of time δ_{cert}, it will retry by sending a *CertRequest* message again to $TRA(n_{pro})$. If still no response from $TRA(n_{pro})$ after θ_{cert} (a pre-configured number of) retries, $TRA(n_{req})$ considers that there is no $TRA(n_{pro})$ at all and notifies n_{req} of the absence of $TRA(n_{pro})$. (This is referred to as Case 1.)

3. If $TRA(n_{req})$ gets a reply from $TRA(n_{pro})$ within θ_{cert} retries, in case that the reply is

   (a) A *LowVersion* message,

      i. If, since the last $CNTER$ resetting, it is the first time that $TRA(n_{req})$ realizes n_{pro} has a high version of TRA, $TRA(n_{req})$ increments $CNTER$ and notifies n_{req} that it need to be updated. (This is referred to as Case 2.)

      ii. Otherwise, $TRA(n_{req})$ pretends nothing happened.

   (b) A *HighVersion* message, $TRA(n_{req})$ notifies n_{req} that $TRA(n_{pro})$ is out-of-date. (This is referred to as Case 3.)

   (c) A *CertReply* message,

      i. $TRA(n_{req})$ validates the $RC(n_{pro})$ contained in the message by the following steps:

         A. Decrypt the $RC(n_{pro})$, and retrieve the R. Recall $RC(n_i) = E_{AK}(R, H(R))$.

         B. Compute the hash digest of R, $H'(R)$.

C. Retrieve $H(R)$ from $RC(n_{pro})$, and compare it with $H'(R)$. If they are equal, the validity test is passed, or fails otherwise.

ii. If the validity check is passed, $TRA(n_{req})$ computes the trust of n_{pro}, based on the trust evaluation on n_{pro} in L_{val} (if any) and the reputation data retrieved from $RC(n_{pro})$, using its self-contained computation logic, and it then passes n_{req} the computation result. (This is referred to as Case 4.)

iii. Otherwise, it considers that the r-Certificate is illegally modified by n_{pro} and notifies n_{req} of the situation. (This is referred to as Case 5.)

In Case 2, if $CNTER$ is equal to or greater than a threshold value θ_{cnt}, n_{req} asks its cluster head for the latest version of the TRA; otherwise, it goes for other transaction partner instead of n_{pro}. In Case 4, based on the computation result, n_{req} makes the decision on whether it starts transaction $Tran_{\star}$ with n_{pro}. In the other cases, what n_{req} is going to do depends on n_{req}'s local policy.

At n_{pro} side

1. Upon receiving the *CertRequest* message from $TRA(n_{req})$, $TRA(n_{pro})$ retrieves the version number of $TRA(n_{req})$ from the message to check if it itself is of the same version as $TRA(n_{req})$.

2. If their version numbers are consistent, $TRA(n_{pro})$ encapsulates the r-Certificate of n_{pro}, $RC(n_{pro})$ (stored in Buf_{cert}), into a *CertReply* message and sends the message to $TRA(n_{req})$. (This is referred to as Case 6.)

3. Otherwise,

   (a) If the version number of $TRA(n_{pro})$ is higher than that of $TRA(n_{req})$, $TRA(n_{pro})$ sends a *LowVersion* message to $TRA(n_{req})$. (This is referred to as Case 7.)

   (b) Otherwise,

i. If, since last $CNTER$ resetting, this situation ever happened with n_{req}, or it already received a *LowVersion* message from n_{req}, $TRA(n_{pro})$ pretends nothing happened.

ii. Otherwise, $TRA(n_{pro})$ sends a *HighVersion* message to $TRA(n_{req})$, increments $CNTER$, and notifies its host n_{pro} that it need to be updated. (This is referred to as Case 8.)

In case 6, if $TRA(n_{pro})$ later receives n_{req}'s transaction request, it could accept or reject it. In Case 7, n_{pro} pretends nothing happened. In Case 8, if $CNTER$ is equal to or greater than θ_{cnt}, n_{pro} asks its cluster head for the latest version of the TRA; otherwise, it pretends nothing happened.

t-Instrument Issuance is triggered by $TRA(n_{req})$ in the post-transaction process right after the termination of $Tran_{\star}$. In Figure 6.3, Step (4) and (5) indicate the process of t-Instrument issuance between n_{req} and n_{pro}. The detail of t-Instrument issuance is presented as follows:

At n_{req} side

1. Upon the termination of $Tran_{\star}$, based on the QoS obtained from n_{pro} during $Tran_{\star}$, n_{req} makes a trust evaluation on n_{pro} for every related context $C_{\star}$. A trust evaluation has the following form: $(ID(n_{pro}), C_{\star}, t_{req,pro}, T)$ where T is the time when the evaluation is made. Then, n_{req} submits these trust evaluations to $TRA(n_{req})$.

2. For every trust evaluation $(ID(n_{pro}), C_{\star}, t_{req,pro}, T)$ submitted by n_{req}, $TRA(n_{req})$,

   (a) Updates $Entry_{eval}(n_{pro}, C_{\star})$ with $(t_{req,pro}, T)$ (if no such an entry, $TRA(n_{req})$ creates one first), and then

   (b) Generates a t-Instrument $TI(n_{req}, n_{pro}, C_{\star}, T)$ accordingly, sends it to $TRA(n_{pro})$, and then

   (c) Expects an ACK message from $TRA(n_{pro})$.

3. If $TRA(n_{req})$ does not receive from $TRA(n_{pro})$ the ACK message corresponding to a issued t-Instrument in a certain period of time δ_{instr}, it will re-send the t-Instrument and expect an ACK message again. The t-Instrument issuance can be retried maximally θ_{instr} (a pre-configured number of) times.

At n_{pro} side

1. After receiving $TI(n_{req}, n_{pro}, C_\star)$ from n_{req}, $TRA(n_{pro})$ verifies its validity in the same way as a r-Certificate is validated.

2. If $TI(n_{req}, n_{pro}, C_\star)$ is invalid, it is simply discarded by $TRA(n_{pro})$.

3. Otherwise, $TRA(n_{pro})$ first retrieves the timestamp T from $TI(n_{req}, n_{pro}, C_\star)$ and then looks up $Entry_{instr}(n_{req}, C_\star)$.

   (a) If $Entry_{instr}(n_{req}, C_\star)$ (shortly $Entry_{instr}$) exists,

      i. In case of $Entry_{instr}.TS < T$, $TRA(n_{pro})$ first updates $Entry_{instr}.INSTR$ and $Entry_{instr}.TS$ respectively with $TI(n_{req}, n_{pro}, C_\star)$ and T, and then sends an ACK message back to $TRA(n_{req})$, and afterwards resets $Entry_{instr}.ACK$ to 1; or,

      ii. In case of $Entry_{instr}.TS = T$ and $Entry_{instr}.ACK < \theta_{ack}$, $TRA(n_{pro})$ sends an ACK message back to $TRA(n_{req})$ and then increments $Entry_{instr}.ACK$; or,

      iii. In all other cases, $TRA(n_{pro})$ discards $TI(n_{req}, n_{pro}, C_\star, T)$ and pretends nothing happened.

   (b) Otherwise, $TRA(n_{pro})$ first creates in Tab_{instr} a new entry $Entry_{instr}(n_{req}, C_\star)$ with $TI(n_{req}, n_{pro}, C_\star)$ and T, and then sends an ACK message back to $TRA(n_{req})$.

r-Certificate Issuance is executed periodically by replica TRAs based on the t-Instruments of their hosts. It involves two types of operations, computing reputation and generating r-Certificate. For any node n_i, $TRA(n_i)$ computes the reputation of its host n_i based on the old

r-Certificate in Buf_{cert} and the t-Instruments in Tab_{instr} using its self-contained computation logic. Since t-Instruments are context-specific, the computation result will not be a single value but a set of such values each of which represents n_i's reputation in a specific context at the time T when it is computed. Afterwards, $TRA(n_i)$ generates a r-Certificate with timestamp T for n_i based on the computation result of previous step, and replaces the old r-Certificate in Buf_{cert} with the new one, and then empties Tab_{instr}.

Trust Management Routine is periodically carried out by every replica TRA to maintain the Tab_{eval} on its hosting node. Because of the temporalness of trust and the limit of local storage space, stale trust evaluation values should be removed from the Tab_{eval}. In each run of the routine, the replica TRA checks the difference between the current time and the timestamp of each entry in the Tab_{eval}. If the difference is bigger than a threshold value δ_{valid}, the entry is removed.

6.5 Proof of Correctness

In this section, we are going to present several lemmas and theorems about the TARM, and we will also give the proof of their correctness.

Lemma 6.5.1: *Every active node will eventually hold a replica of the latest TRA.*

Proof:

During the *network initialization* phase, in the first stage, the AL broadcasts a TRA in the backbone network, and it is guaranteed that all living backbone nodes receive the TRA since the backbone network is connected despite the backbone-node failure (by assumption); in the second stage, every cluster head (which is a backbone node) broadcasts the received TRA within its own cluster in the original network, but cluster members' receipt of the TRA can not be guaranteed due to node failure or cluster re-organization. Under these circumstances, after the *network initialization* phase, all living backbone nodes hold a replica of the most-recently

launched TRA while some non-backbone nodes may not. For any non-backbone node n_i, its failure in receiving the latest TRA results in two possible cases: 1) n_i does not have a replica TRA at all; 2) n_i has a out-of-date TRA. For the first case, n_i is aware of its TRA absence and will actively ask its cluster head for the latest version of the TAR. In the second case, n_i will finally be notified of the staleness of its replica TRA when it does not succeed later in acquiring other nodes' r-Certificates in the *service-offering* phase, and when the number of such failures is beyond θ_{cnt}, n_i will ask its cluster head for the latest version of the TRA. Therefore, n_i is able to have a replica of the latest TRA in either of the above two cases. Hence, sooner or later, every active node will eventually hold a replica of the latest TRA.

Lemma 6.5.2: *All the trust evaluations on a node are aggregated on the node itself.*

Proof:

In the post-transaction process of any transaction $Tran_\star$, n_{req} commissions $TRA(n_{req})$ to issue n_{req} a t-Instrument based on the QoS n_{req} provides during $Tran_\star$. The t-Instrument is then stored (in the Tab_{instr}) on n_{req} and managed by $TRA(n_{req})$. Therefore, a node will hold all the trust evaluations, in the form of t-Instrument, from the nodes it as *provider* ever had a transaction with, and thus Lemma 6.5.2 holds its correctness.

Theorem 6.5.1: *A node's reputation computed in ATRM reflects the overall trust evaluation of other nodes on the node.*

Proof:

In ATRM, by choosing the accurate reputation model, the computed reputation can accurately reflect a node's trustworthiness in a global view if sufficient individual trust evaluations on the node are provided. According to Lemma 6.5.2, for any node n_i, it holds all its t-Instruments issued by other nodes. Therefore, the reputation computed by $TRA(n_i)$ using this full set of t-Instruments will truly indicate the overall trust evaluation of n_i from other nodes' point of view. Hence, Theorem 6.5.1 is correct.

Theorem 6.5.2: *The reputation of a node can be directly acquired by any other node without network-wide flooding.*

Proof:

The correctness of Theorem 6.5.2 is derived from the fact that every node's reputation is locally stored in the form of r-Certificate by the node itself. Remember that a node's reputation is objective and can be shared by all the other nodes in ATRM. For any transaction $Tran_{\star}$, n_{req} knows the fact that n_{pro} locally stores its own r-Certificate. Hence, n_{req} can get n_{pro}'s r-Certificate simply by querying n_{pro} itself instead of by exhaustively asking every other node in the network.

Theorem 6.5.3: *A node's locally-stored trust and reputation are readable to all the replica TRAs (of proper version) but confidential to any node including the node itself.*

Proof:

When a replica TRA generates a t-Instrument/r-Certificate, it encrypts the t-Instrument/r-Certificate with its secret key AK. By Lemma 6.5.1, all the replica TRAs are eventually derived from the same original TRA and share the same AK, and thus they are sooner or later able to successfully decrypt the t-Instruments/r-Certificates generated by each other. Since AK is securely kept only by replica TRAs, nodes are unable to read either their locally-stored t-Instruments/r-Certificates or received ones. In addition, by using strong cryptography algorithms and sufficiently long secret keys, successful attacks to AK via cryptanalysis and brute force guessing can be precluded. Hence, Theorem 6.5.3 holds.

Theorem 6.5.4: *A node's unauthorized modification to its locally-stored trust and reputation information can be identified.*

Proof:

Every t-Instrument/r-Certificate is encrypted with the secret key of a replica TRA. According to the underlying network model, nodes are connected through secure links, and thus the data passed all the way to a replica TRA in application layer is the original one sent from the sender

node. Moreover, before t-Instrument issuance and r-Certificate provision happen between two nodes, their local replica TRAs are required to pass the consistency test. For any transaction requester, it has no reason to tamper with its received confidential r-Certificates since they are critical for its decision-making; and it has no reason to tamper with the t-Instruments it issues since it itself is the actual creator of these t-Instruments. Under these circumstances, the failure of a replica TRA in validating the received t-Instruments/r-Certificates must be due to the host/sender nodes' unauthorized modification to the t-Instruments/r-Certificates. Hence, Theorem 6.5.4 holds its correctness.

6.6 Complexity Analysis

The complexity of ATRM is the sum of that of the *network initialization* phase and that of the *service-offering* phase. Because the *network initialization* phase (which is a broadcast process) is run only once at the beginning of the execution of ATRM (unless special security requirements are specified), in this section, we will analyze the complexity only of the service-offering phase. The complexity analysis is done under two assumptions: reliable transmission and transaction-qualified nodes.

According to Section 6.4.5, only r-Certificate acquisition and t-Instrument issuance involve message transmission during the *service-offering* phase, and for each r-Certificate acquisition and t-Instrument issuance process, there are two ATRM packets injected into the network respectively. In the case, the number of ATRM packets per transaction is four. Let $N_{transaction}$ denote the number of transactions. Then, the message complexity of the *service-offering* will be exactly:

$$C_{msg} = 4 \times N_{transaction} \tag{6.5}$$

Based on this formula, we can see that the number of ATRM packets is directly proportional to the number of transactions during the *service-offering* phase but irrelevant to the number of

nodes in the network. The solid lines in Figure 6.4(a) and 6.5(a) clearly show this relation.

Now, let us analyze the average r-Certificate acquisition delay, T_{delay}, which is defined as the average time difference between the time when a *CertRequest* message is sent from a transaction *requester* and the time when the corresponding *CertReply* message is successfully received by the *requester*. By Section 6.4.5, during a r-Certificate acquisition process, a *CertRequest* message is first sent from the *requester* to the *provider*, and the *provider* then sends a *CertReply* message back to the *requester* after it finishes processing the received *CertRequest* message. Let us denote by T_{e2e} the average end-to-end delay of the underlying routing protocol and by $T_{process}$ the average delay for processing a *CertRequest* message. Then, we have the following formula:

$$T_{delay} = 2 \times T_{e2e} + T_{process} \tag{6.6}$$

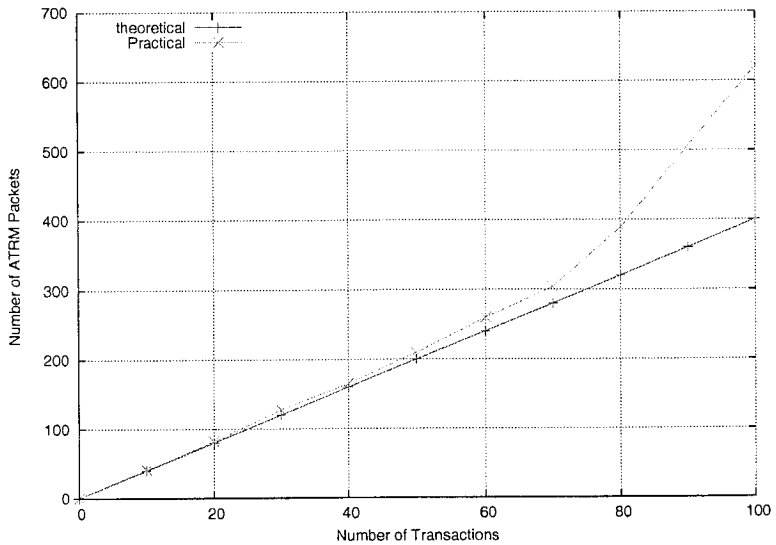
For a particular WSN, $T_{process}$ is a constant factor, and thus, T_{delay} actually depends on the routing protocol employed.

6.7 Performance Evaluation

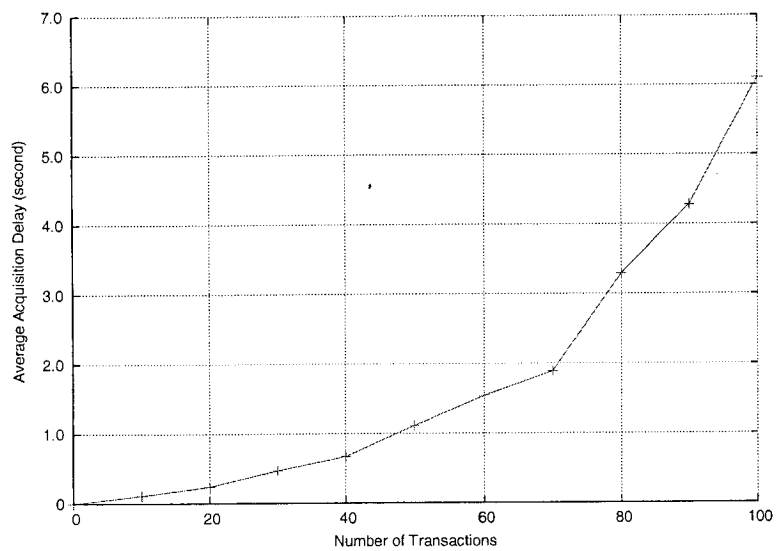
In this section, we evaluate the performance of ATRM in two aspects: 1) the message overhead added by ATRM to the underlying network during the *service-offering* phase; and 2) the average r-Certificate acquisition delay, based on the experiments we have carried out using the Network Simulator, NS-2. Since the backbone network is used only for agent distribution in the *network initialization* phase, it was not simulated in our experiments.

During our experiments, we use a flat rectangle area of $100m \times 100m$. Nodes are randomly deployed and are fixed throughout the simulation. They communicate with each other in the WSN using a radio of $60m$. Every node is transaction-qualified and is pre-assigned a TRA with 64-bit secret key. The system parameters δ_{cert} and δ_{instr} are set to 20 seconds while θ_{cert} and θ_{instr} are configured to 5. Some nodes spontaneously trigger transactions, each of which

lasts for a randomly decided period of simulated time. During a transaction, the *requester* constantly sends 300-byte packets at the transmission rate of 2 packets per second. To have an average result, we run the simulation for each scenario 5 times. During each simulation run, we kept track of the number of ATRM packets and the average r-Certificate acquisition delays. Based on the above setup, we conducted two sets of experiments. In the first set, where the



(a) Number of ATRM packets VS. Number of transactions



(b) Average acquisition delay VS. Number of transactions

Figure 6.4: Experiment set 1 with 100 nodes

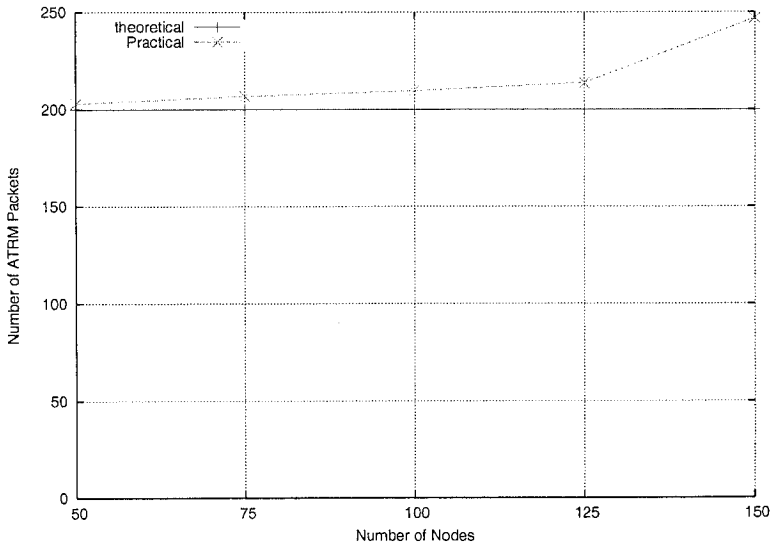
number of nodes were set to 100 in order to simulate the high density of WSNs, we aimed at studying the performance of ATRM in the situations with fixed number of nodes and changing numbers of transactions. In the second set, we investigated the performance of ATRM with fixed number of transactions (50) and varying numbers of nodes.

Figure 6.4(a) shows the change in the number of ATRM packets in relation to the change in the number of transactions. The dashed line stems from experimental results while the solid one is based on theoretical analysis. Following either of these two lines, we can see that the number of ATRM packets rises as the number of transactions increases. With the increase of the number of transactions from 0 to 70, the two lines stay very close to each other, and the gap between them grows slowly. However, after the number of transactions is beyond 70, the gap between the two lines increases at a very fast speed. This situation is due to the impact of transactions on network performance. In fact, the moderate increase of traffic load can be comfortably adapted by the underlying network. Whereas, if the traffic load is beyond the limit that the network can adequately handle, network congestion and consequent packet loss will occur, furthermore, the heavier the traffic load, the more serious the situation. In our experiments, 70 transactions make traffic load reach that limit, and therefore, when the number of transactions is greater than 70, packet loss happens very often during the *r*-Certificate acquisition and *r*-Instrument issuance, thus causing frequent failure of the two processes. By the definition of ATRM, when the two processes fail, transaction *requesters* will retry. Frequent *r*-Certificate acquisition and *r*-Instrument issuance retries cause the injection of more ATRM packets into the network and thus widen the distance between the experimental results and theoretical analysis.

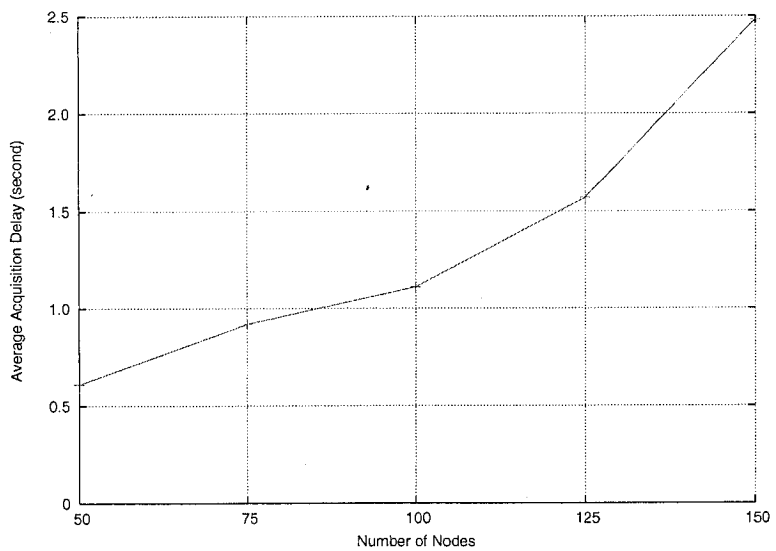
Figure 6.4(b) illustrates the average delay of *r*-Certificate acquisition as a function of the number of transactions. In busy networks, network packets are enqueued before being processed, and the heavier the traffic load, the longer they stay in queue, and thus, the longer the transmission delay. Thereby, the line goes up as the number of transactions increases.

Figure 6.5(a) depicts the variation in the number of ATRM packets in relation to the change

in the number of nodes when the number of transactions is fixed. The dashed line stems from experimental results while the solid one is based on theoretical analysis. Following the dashed line, we actually find that the number of ATRM packets increases as the number of nodes goes up. Clearly, these experimental results deviates from theoretical analysis. The reason for the difference is that per node throughput declines as the number of nodes increases [86] (note



(a) Number of ATRM packets VS. Number of nodes



(b) Average acquisition delay VS. Number of nodes

Figure 6.5: Experiment set 2 with 50 transactions

that channel capacity is fixed in our experiments). Poor throughput causes packet loss and retransmission and thus results in the increase of the total number of ATRM packets.

Figure 6.5(b) exhibits the average delay of r-Certificate acquisition as a function of the number of nodes. As we explained previously, per node throughput goes down as the number of nodes increases in a wireless network with fixed channel capacity. Therefore, when the number of nodes becomes larger and larger, packets have to stay in queue for an increasingly longer period of time before they are processed. This makes the average delay of r-Certificate acquisition rise.

Chapter 7

Conclusions and Future Work

Security and anonymity are the most challenging issues in wireless ad-hoc networks. In this thesis, we proposed a novel secure distributed anonymous routing protocol (SDAR) for wireless ad-hoc networks, which anonymously creates routes on the fly to support onion routing without the originator knowing either the keys of the mix nodes nor the topology of the network. We elaborated the algorithm, and presented an extensive set of simulation experiments to evaluate its performance. The simulation results indicated clearly that the anonymity in wireless ad-hoc network is feasible and could be incorporated within an ad-hoc routing protocol, and that SDAR provides a good solution to achieve anonymity in wireless ad-hoc networks at a reasonable additional cost when compared to the well-known DSR routing protocol [40].

In the future, we will study the performance of SDAR based on other routing protocols such as AODV [61]. Additionally, as we know, power-conservation and scalability are big issues in wireless ad-hoc networks as well. Since SDAR requires quiet large amount of computational power for authentication/encryption and uses much network-wide broadcast messages for path discovery, we are going to investigate its impact on the lifetime and scalability of the network and try to find the optimized solution.

Trust and reputation management is another important issue in distributed autonomous environments such as wireless ad-hoc networks. Its solution should be carefully developed

in accordance with the underlying network model in order to gain optimal network performance. We proposed a novel agent-based trust and reputation management scheme (ATRM) for wireless sensor networks (WSNs). ATRM introduces a trust and reputation local storage strategy, which makes trust aggregation and reputation propagation become straightforward without requiring network-wide flooding, and thus reducing reputation acquisition latency. We proved the correctness of ATRM and made preliminary evaluation on the performance of ATRM based on experimental results.

ATRM is an ongoing project and is still in its initial stage. There are many relevant problems that need to be studied and solved. First of all, the effectiveness of ATRM is based on the assumption of secure mobile agents, which is not studied in our current work. Secondly, since we used only a single model for trust and reputation management, our future work may also include studying different models and finding the one that is the best suitable for WSNs.

Appendix A

Trademarks

Product and brand names used in this thesis may be trademarks or registered trademarks of their respective owners. Any such trademarks or registered trademarks are the properties of their respective owners.

Bluetooth is a trademark of Bluetooth SIG, Inc.

IDEA is a registered trademark of Asom Systec AG.

PGP is a registered trademark of PGP Corporation.

RC4, RSA, MD5 are registered trademarks of RSA Data Security, Inc.

Java is a registered trademark or trademark of Sun Microsystems, Inc.

Appendix B

Pseudocodes

The following is the pseudo-codes for processing path request messages in SDAR routing protocol.

```
// Input:  M, data packet; ID, sender's address
// Output: none
void route_request_handler (string M, address ID)
{
    part1 = extract_part1(M);
    msgtype = get_from_part1(part1, TYPE);
    if (msgtype != ROUTEREQUEST)
        return;
    tpk = get_from_part1(part1, TPK);
    if (tpk is already locally stored)
        return;          // The request was already processed
    trustrequirement = get_from_part4(part4, TRUSTREQ)
    if (!get_community_key(ID, trustrequirement, &communitykey))
        return;          // I am not the intended next hop
    if (get_trust_level_of(ID) != trustrequirement)
```

```
    return;          // There is not mutual trust
cipthedpart2 = extract_part2(M);
part2 = decrypt(cipthedpart2, myprivatekey);
destinationid = get_from_part2(part2, IDR);
if (destinationid == myownid)
{//I am actually the destination
    sessionkey = get_from_part2(part2, KS);
    paddinglength = get_from_part2(part2, PLS);
    part3 = extract_part3(M, paddinglength);
    cipthedpart4 = extract_part4(M, paddinglength);
    part4 = decrypt (cipthedpart4, sessionkey);
    if (!header_integrity_check(part1, part2, part3, part4))
        return;      //The packet header has been  modified
    tsk = get_from_part4(part4, TSK);
    cipthedpart5 = extract_part5(M, paddinglength);
    //Store encrypted intermediate node info in a list, ciphedtext.
    split(cipthedpart5, ciphedtext);
    //Decrypt intermediate node info into a list, plaintext.
    decrypt_into (ciphedtext, tsk, plaintext);
    if (!node_info_integrity_check(ciphedtext, plaintext))
        return;      //The intermediate node info has been modified
    // Store source node information
    store_source_node_info(part1, part2, patr4);
    // Store intermediate nodes information
    store_intermediate_node_info(plaintext);
    // Initiate path reverse phase
    path_reverse(tpk);
```

```
    }else
    {//I am not the destination
        sk = generate_session_key();
        skid = assign_id_to(sk);
        store_predecessor_info(tpk, ID, sk, skid);
        M' = M + myownid + sk + skid;
        hashedmsg = hash(M');
        signedhmsg = sign(hashedmsg, myprivatekey);
        myowninfor = encrypt_node_info(myownid,sk,skid,singedhmsg,tpk);
        // Append my own information to M
        M' = M + myowninfor;
        // Store the hash result of M' locally for future use.
        make_history_record(tpk, timestamp, hash(M'));
        // Broadcast M' to selected neighborhood
        broadcast(M', trustrequirement);
    }
}
```

Appendix C

Acronyms

ACK: Acknowledgment

AK: Agent Key

AL: Agent Launcher

ANODR: ANonymous On Demand Routing

AODV: Ad-hoc On-demand Distance Vector Routing

ARAN: Authenticated Routing for Ad-hoc Networks

ATRM: Agent-based Trust and Reputation Management

C: Context

CA: Certificate Authority

CCITT: International Telegraph and Telephone Consultative Committee

CORC: Credit-Only

CPU: Central Processing Unit CRL: Certificate Revocation List

DC: Dinning Cryptographer

DCRC: Debit-Credit

DES: Data Encryption Standard

DMRA: Dynamic mix Route Algorithm

DoS: Denial of Service

DREG: Destination Registration

DSA: Digital Signature Algorithm

DSDV: Destination-Sequenced Distance-Vector Routing

DSR: Dynamic Source Routing

ETSI: European Telecommunications Standards Institute

FIPS: Federal Information Processing Standards

HTLCK: High Trust Level Community Key

HTLT: High Trust Level Threshold

HTR: High Trust Requirement

ID: Identity

IDEA: International Data Encryption Algorithm

IEEE: Institute of Electrical & Electronic Engineers

INRT: Intermediate Node Reply Token

IP: Internet Protocol ITU: International Telecommunication Union

K: Key

LAN: Local Area Network

LTR: Low Trust Requirement

MA: Mobile Agent

MAC: Message Authentication Code

MADV: MIX Advertisement

MANET: Mobile Ad-hoc Network

MD5: Message Digest (version 5)

MRL: Message Retransmission List

MTLCK: Medium Trust Level Community Key

MTLT: Medium Trust Level Threshold

MTR: Medium Trust Requirement

NPDU: Network Protocol Data Unit

NRC: National Research Council Canada

NS-2: Network Simulator (version 2)

OSI: Open Systems Interconnection

P2P: Peer-to-Peer

PGP: Pretty Good Privacy

PKI: Public Key Infrastructure

PK: Public Key

QoS: Quality of Service

R: Receiver

RA: Registration Authority

RCA: Reputation Computation Agent

RERR: Route Error

RREQ: Route Request

RREQ-DSE: Route Request Double Signature Extension

RREQ-SSE: Route Request Single Signature Extension

RREP: Route Reply

RREP-DSE: Route Reply Double Signature Extension

RREP-SSE: Route Reply Single Signature Extension

RSA: Rivest Shamir Adleman

RUPD: Route Update

r-Certificate: Reputation Certificate

S: Sender/Source

SA: Security Association

SAODV: Secure AODV

SAR: Security-aware Ad-hoc Routing

SDAR: Secure Distributed Anonymous Routing

SEAD: Secure Efficient Distance Vector Routing

SEAL: Software-optimized Encryption Algorithm

SHA-1: Secure Hash Algorithm 1

SRP: Secure Routing Protocol

TESLA: Transparent Extensible Session-Layer Architecture

TPK: Temporary Public Key

TRA: Trust and Reputation Assessor

TSK: Temporary Secure key

t-Instrument: Trust Instrument

WRP: Wireless Routing Protocol

WSN: Wireless Sensor Network

References

[1] A. Abdul-Rahman and S. Hailes. "A distributed trust model". In *Proceedings of the 1997 workshop on New security paradigms*, pages 48–60, Langdale, U.K., 1997.

[2] A. Abdul-Rahman and S. Hailes. "Supporting Trust in Virtual Communities". In *Proceedings of the 33rd Ann. Hawaii Int'l Conf. System Sciences (HICSS 33)*, volume 6, pages 6007–6016, 2000.

[3] B. Alunkal, I. Veljkovic, G. V. Laszewski, and K. Aminand. "Reputation-based Grid Resource Selection". In *Proceedings of Workshop on Adaptive Grid Middleware*, New Orleans, U.S., Sep. 2003. (Available at http://www-unix.mcs.anl.gov/ laszewsk/papers /vonLaszewski–reputation.pdf on May 06, 2005).

[4] F. Azzedin and M. Maheswaran. "Towards Trust-Aware Resource Management in Grid Computing Systems". In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, pages 452–457, Berlin, Germany, May 2002.

[5] S. Banerjee and S. Khuller. "A Clustering Scheme for Hierarchical Control in Multi-hop Wireless Networks". In *IEEE Infocom 2001*, pages 1028–1037, Anchorage, U.S., Apr. 2001.

[6] S. Basagni. "Distributed Clustering for Ad hoc Networks". In *Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '99)*, pages 310–315, Fremantle, Australia, Jun. 1999.

[7] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. "RFC 2704 - The KeyNote Trust Management System Version 2", Sep. 1999. (Available at http://www.faqs.org /rfcs/rfc2704.html on May 06, 2005).

[8] M. Blaze, J. Feigenbaum, and J. Lacy. "Decentralized Trust Management". In *Proceedings of 1996 IEEE Conference on Privacy and Security*, pages 164–173, Oakland, U.S., 1996.

[9] A. Boukerche. "Performance evaluation of routing protocols for ad hoc wireless net-

works". *ACM/Kluwer Mobile Networks and Applications*, 9(4):333–342, 2004.

[10] J. Broch, D. A. Maltz, D. B. Johnson, Y. Hu, and J. Jetcheva. "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols". In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98)*, pages 85–97, Dallas, U.S., 1998.

[11] S. Buchegger and J. Boudec. "A robust reputation system for P2P and mobile ad-hoc networks". In *Proceedings of the 2nd Workshop on the Economics of Peer-to-Peer Systems*, Cambridge, U.S., Jun. 2004. (Available at http://www.eecs.harvard.edu /p2pecon/confman/papers /s2p2.pdf on May 06, 2005).

[12] S. Buchegger and J. L. Boudec. "Nodes Bearing Grudges: Towards Routing Security, Fairness, and Robustness in Mobile Ad Hoc Networks". In *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pages 403–410, 2002.

[13] N. Budhiraja and K. Marzullo. Tradeoffs in Implementing Primary-Backup Protocols. In *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing*, pages 280–288, San Antonio, U.S., Oct. 1995.

[14] D. Chaum. "Untraceable Electronic mail, Return Addresses, and Digital pseudonyms". *Communications of the ACM*, 24(2):84–88, Feb. 1981.

[15] D. Chaum. "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability". *Journal of Cryptography*, 1(1):65–75, 1988.

[16] B. Christianson and W. S. Harbison. "Why Isn't Trust Transitive?". In *Proceedings of Security Protocols International Workshop*, pages 171–176. University of Cambridge, 1996.

[17] A. W. Dent and C. J. Mitchell. *"User's Guide To Cryptography And Standards"*. Artech House, 2004.

[18] P. Dewan and P. Dasgupta. "Securing P2P Networks Using Peer Reputations: Is there a silver bullet?". In *Proceedings of IEEE Consumer Communications*

and Networking Conference (CCNC 2005), Las Vegas, U.S., 2005. (Available at http://cactus.eas.asu.edu/partha/Papers-PDF/2005/Dewan-CCNC.pdf on May 06, 2005).

[19] D. Eastlake. "RFC 3174 - US Secure Hash Algorithm 1 (SHA1)", Sep. 2001. (Available at http://www.faqs.org/rfcs/rfc3174.html on May 06, 2005).

[20] K. El-Khatib, L. Korba, R. Song, and G. Yee. "Secure Dynamic Distributed Routing Algorithm for Ad Hoc Wireless Networks". In *Proceedings of the 1st International Workshop on Wireless Security and Privacy*, pages 359–366, Kaohsiung, Taiwan, Oct. 2003.

[21] D. L. Evans, P. J. Bond, and A. L. Bement. "The Keyed-Hash Message Authentication Code (HMAC), Jun. 2002. (Available at http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf on May 06, 2005).

[22] A. Fiat and A. Shamir. "How to prove to yourself: practical solutions to identification and signature problems". *Advances in CryptologyCCrypto 86*, pages 186–194, 1987.

[23] FIPS-PUB-186. "DIGITAL SIGNATURE STANDARD (DSS)", May. 1994. (Available at http://www.itl.nist.gov/fipspubs/fip186.htm on May 06, 2005).

[24] FIPS-PUB-46-3. "DATA ENCRYPTION STANDARD (DES)", Oct. 1999. (Available at http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf on May 06, 2005).

[25] L. Ford and D. Fulkerson. *"Flows in Networks"*. Princeton University Press, 1962.

[26] D. Gambetta. "Can We Trust Trust?". In D. Gambetta, editor, *Trust: Making and Breaking Cooperative Relations*, chapter 13, pages 213–237. Department of Sociology, University of Oxford, electronic edition edition, 1990.

[27] J.-E. Garcia, A. Kallel, K. Kyamakya, K. Jobmann, J.-C. Cano, and P. Manzoni. "A novel DSR-based energy-efficient routing algorithm for mobile ad-hoc networks". In *Proceedings of 2003 IEEE 58th Vehicular Technology Conference*, pages 2849–2854, Orlando, U.S., Oct. 2003.

[28] M. Gerla and J. T. Tsai. "Multicluster, mobile, multimedia radio network". *ACM-Baltzer Journal of Wireless Networks*, 1(3):255–265, Oct. 1995.

[29] I. Goldberg and A. Shostack. "Freedom network 1.0 architecture", Nov. 1999. (Available at http://www.homeport.org / adam/zeroknowledgewhitepapers /arch-notech.pdf on May 06,2005).

[30] V. S. Grishchenko. "A fuzzy model for context-dependent reputation". In *Proceedings of Trust, Security and Reputation Workshop at ISWC 2004*, Hiroshima, Japan, 2004. (Available at http://trust.mindswap.org/trustWorkshop/papers/1-s.pdf on May 06, 2005).

[31] M. Gupta, P. Judge, and M. Ammar. "A Reputation System for Peer-to-peer Networks". In *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, pages 144–152, Monterey, U.S., Jun. 2003.

[32] P. Gupta and P. R. Kumar. "The Capacity of Wireless Networks". *IEEE Transactions on Information Theory*, IT-46(2):388–404, Mar. 2000.

[33] Q. He, O. D. Wu, and P. Khosla. "SORI: A Secure and Objective Reputation-based Incentive Scheme for Ad-hoc Networks". In *Proceedings of IEEE Wireless Communications and Networking Conference 2004*, 2004. (Available at http://www.wu.ece.ufl.edu /mypapers/WCNC04_incentive.pdf on May 06, 2005).

[34] A. Herzberg, Y. Mass, J. Michaeli, D. Naor, and Y. Ravid. "Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers". In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 2–14, Berkeley, U.S., May 2000.

[35] R. Housley, W. Polk, W. Ford, and D. Solo. "RFC 3280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Apr. 2002. (Available at http://www.ietf.org/rfc/rfc3280.txt on May 06, 2005).

[36] Y. Hu, D. Johnson, and A. Perrig. "SEAD: Secure Efficient Distance Vector Routing in Mobile Wireless Ad Hoc Networks". In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 02)*, pages 3–13, 2002.

[37] Y. Hu, A. Perrig, and D. B. Johnson. "Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks". In *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking(MobiCom 2002)*, pages 12–23, Sep. 2002.

[38] ITU-T. "X.800 (03/91) Security Architecture for Open Systems Interconnection for CCITT Applications", 1991. (Available at http://fag.grm.hia.no/IKT7000/litteratur/ paper/x800.pdf on May 06, 2005).

[39] S. Jiang, N. Vaidya, and W. Zhao. "A mix route algorithm for mix-net in wireless ad hoc networks". In *Proceedings of the 1st IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, pages 406–415, Fort Lauderdale, U.S., Oct. 2004.

[40] D. B. Johnson and D. A. Maltz. "Dynamic source routing in ad hoc wireless networks". In T. Imielinski and H. Korth, editors, *Mobile computing*, chapter 5, pages 153–181. Kluwer Academic, 1996.

[41] R. Jurca and B. Faltings. "An incentive compatible reputation mechanism". In *Proceedings of IEEE International Conference on E-Commerce 2003*, pages 285–292, Jun. 2003.

[42] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. "The Eigentrust Algorithm for Reputation Management in P2P Networks". In *Proceedings of the 12th International World Wide Web Conference*, pages 640–651, Budapest, Hungary, 2003.

[43] J. Kong and X. Hong. "ANODR: anonymous on demand routing with untraceable routes for mobile ad-hoc networks". In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking and computing*, pages 291–302, Annapolis, U.S., 2003.

[44] L. Korba, R. Song, and G. Yee. "Anonymous Communications for Mobile Agents". In *Proceedings of the 4th International Workshop on Mobile Agents for Telecommunication Applications*, pages 171–181, Barcelona, Spain, 2002.

[45] N. Li and J. Mitchell. "RT: A Role-based Trust-management Framework". In *Proceedings of the 3rd DARPA Information Survivability Conf. and Exposition (DISCEX'03)*, pages 201–212, Apr. 2003.

[46] C. R. Lin and M. Gerla. "Adaptive Clustering for Mobile Networks". *IEEE Journal on Selected Areas in Communications*, 15(7):1265–1275, Sep. 1997.

[47] J. Liu and V. Issarny. "Enhanced reputation mechanism for mobile ad hoc networks". In

Proceedings of the 2nd International Conference on Trust Management, volume 2995, pages 48–62, Mar. 2004.

[48] Z. Liu, A. W. Joy, and R. A. Thompson. "A Dynamic Trust Model for Mobile Ad Hoc Networks". In *Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04)*, pages 80–85, Suzhou, China, May 2004.

[49] I. Mantin. "Analysis of the Stream Cipher RC4". Master's thesis, Weizmann Institute of Science, 2001.

[50] D. H. McKnight and N. L. Chervany. "The Meanings of Trust". Technical Report 94-04, Carlson School of Management, University of Minnesota, 1996.

[51] P. Michiardi and R. Molva. "Core: A collaborative reputation mechanism to enforce node cooperation in mobile AD HOC networks". In *Proceedings of the 6th IFIP Communications and Multimedia Security Conference*, pages 107–121, Portorosz, Slovenia, 2002.

[52] B. A. Misztal. *"Trust in Modern Societies: The Search for the Bases of Social Order"*. Polity Press, Polity Press, Dec. 1995.

[53] L. Mui, M. Mohtashemi, and A. Halberstadt. "A Computational Model of Trust and Reputation". In *Proceedings of the 35th Hawaii International Conference on System Sciences*, pages 188–196, 2002.

[54] S. Murthy and J. J. G. Aceves. "An Efficient Routing Protocol for Wireless Networks". *ACM/Baltzer Journal on Mobile Networks and Applications*, 1(2):183–197, Oct. 1996.

[55] R. Naik, S. Biswas, and S. Datta. "Distributed Sleep-Scheduling Protocols for Energy Conservation in Wireless Networks". In *Proceedings of the 38th Hawaii International Conference on System Sciences 2005*, pages 285b–294b, Hawaii, U.S., 2005.

[56] M. Oliveira, J. Crowcroft, and M. Slater. "An innovative design approach to build virtual environment systems". In *Proceedings of the workshop on Virtual environments 2003*, pages 143–151, Zurich, Switzerland, May. 2003.

[57] P. Papadimitratos and Z. J. Haas. "Secure Routing for Mobile Ad hoc Networks". In

Proceedings of SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002), pages 193–204, San Antonio, U.S., Jan. 2002.

[58] T. G. Papaioannou and G. D. Stamoulis. "Effective Use of Reputation of Peer-to-Peer Environments". In *Proceedings of IEEE/ACM CCGRID 2004, GP2PC workshop*, Chicago, U.S., Apr. 2004. (Available at http://nes.aueb.gr/publications /2004.p2p_policies.GP2PC.pdf on May 06, 2005).

[59] M. Perillo, Z. Ignjatovic, and W. Heinzelman. "An energy conservation method for wireless sensor networks employing a blue noise spatial sampling technique". In *Proceedings of the third international symposium on Information processing in sensor networks*, pages 116–123, Berkeley, U.S., 2004.

[60] C. Perkins and P. Bhagwat. "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers". In *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[61] C. E. Perkins and E. M. Royer. "Ad hoc On-Demand Distance Vector Routing". In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, New Orleans, U.S., Feb. 1999.

[62] A. Pfitzmann and M. Waidner. "Networks Without User Observability – Design Options". In *Proceedings of EUROCRYPT 1985*. Springer-Verlag, LNCS 219, 1985. (Available at http://www.semper.org/sirene/publ/PfWa_86anonyNetze.html on May 06, 2005).

[63] A. A. Pirzada and C. McDonald. "Establishing Trust In Pure Ad-hoc Networks". In *Proceedings of the 27th conference on Australasian computer science*, ACM International Conference Proceeding Series, pages 47–54, Dunedin, New Zealand, 2004.

[64] S. Pleisch and A. Schiper. "Modeling Fault-Tolerant Mobile Agent Execution as a Sequence of Agreement Problems". In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems*, pages 11–20, Nuremberg, Germany, Oct. 2000.

[65] J. Raymond. "Traffic Analysis: Protocols, Attacks, Design Issues, and Open problems". In *Proceedings of International Workshop on Design Issues in Anonymity and Unob-*

servability, volume 2009 of *Lecture Notes in Computer Science*, pages 10–29, Berkeley, U.S., Jul. 2000.

[66] M. Reed, P. Syverson, and D. Goldschlag. "Proxies for anonymous routing". In *Proceedings of the 12th Annual Computer Security Applications Conference*, pages 95–104, Dec. 1995.

[67] M. K. Reiter and A. D. Rubin. "Crowds: Anonymity for Web Transactions". *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[68] K. Ren, T. Li, Z. Wan, F. Bao, R. H. Deng, and K. Kim. "Highly reliable trust establishment scheme in ad hoc networks". *Eslevier Jaournal of Compuater Networks*, pages 687–699, Apr. 2004.

[69] R. Rivest. "RFC 1321 - The MD5 Message-Digest Algorithm", Apr. 1992. (Available at http://www.faqs.org/rfcs/rfc1321.html on May 06, 2005).

[70] R. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". *Communications of the ACM*, 21(2):120–126, Feb. 1978.

[71] P. Rogaway and D. Coppersmith. "A software-optimized encryption algorithm". *Journal of Cryptology*, 11(4):273–287, 1998.

[72] K. Rothermel and M. straber. "A Fault-Tolerant Protocol for Providing the Exactly-Once Property of Mobile Agents". In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, pages 100–108, 1998.

[73] T. Sander and C. Tschudin. Protecting mobile agents against malicious hosts. In *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 44–60, Heidelberg, Germany, 1998.

[74] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields, and E. M. Belding-Royer. "A Secure Routing Protocol for Ad Hoc Networks". In *Proceedings of the 2002 IEEE International Conference on Network Protocols (ICNP)*, pages 78–87, Nov. 2002.

[75] F. B. Schneider. "Towards Fault-Tolerant and Secure Agentry". In *Proceedings of the 11th International Workshop on Distributed Algorithms*, pages 1–14, Saarbrucken, Ger-

many, Sep. 1997.

[76] D. Scott, A. Beresford, and A. Mycroft. "Spatial Security Policies for Mobile Agents in a Sentient Computing Environment". In *Proceedings of FASE 2003*, volume 2621 of *Lecture Notes in Computer Science*, pages 102–117, Warsaw, Poland, Apr. 2003.

[77] B. Strulo, J. Farr, and A. Smith. "Securing Mobile Ad hoc Networks – A Motivational Approach". *BT Technology Journal*, 21(3):81–89, 2003.

[78] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. "Anonymous connections and onion routing". In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 44–54, Oakland, U.S., May. 1997.

[79] Y. Tahara, A. Ohsuga, and S. Honiden. "Mobile agent security with the IPEditor development tool and the mobile UNITY language". In *Proceedings of the 5th international conference on Autonomous agents*, pages 656–662, Montreal, Canada, 2001.

[80] J. Tardo and L. Valente. "Mobile Agent Security and Telescript". In *Proceedings of the 41st International Conference of the IEEE Computer Society (CompCon '96)*, pages 58–63, Feb. 1996.

[81] G. Theodorakopoulos and J. S. Baras. "Trust evaluation in ad-hoc networks". In *Proceedings of the 2004 ACM workshop on Wireless security*, pages 1–10, Philadelphia, U.S., 2004.

[82] TheOpenGroup. "Architecture for Public-Key Infrastructure (APKI) - Draft 1", 1997. (Available at http://www.opengroup.org /public/tech/security/pki /apki_1-0.pdf on May 06, 2005).

[83] L. Venkatraman and D. Agrawal. "A novel authentication in ad hoc networks". In *Proceedings of the 2nd IEEE Wireless Communications and Networking Conference*, volume 3, pages 1268–1273, Chicago, U.S., Sep. 2000.

[84] L. Venkatraman and D. Agrawal. "Strategies for enhancing routing security in protocols for mobile ad hoc networks". *Journal of Parallel and Distributed Computing*, 63(2):214–227, 2003.

[85] Y. Wang and J. Vassileva. "Trust and Reputation Model in Peer-to-Peer Networks". In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing (P2P'03)*, pages 150–157, Linkoping, Sweden, Sep. 2003.

[86] K. Xu, X. Hong, and M. Gerla. "Landmark routing in ad hoc networks with mobile backbones". *Journal of Parallel and Distributed Computing*, 63(2):110–122, Feb. 2003.

[87] S. Yi, P. Naldurg, and R.Kravets. "Security-Aware Ad Hoc Routing Protocol for Wireless Networks". In *Proceedings of the 2nd ACM Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc'01)*, pages 299–302, 2001.

[88] M. G. Zapata and N. Asokan. "Securing Ad Hoc Routing Protocols". In *Proceedings of ACM Workshop on Wireless Security (WiSe)*, pages 1–10, 2002.

[89] B. Zhu, Z. Wan, M. S. Kankanhalli, F. Bao, and R. H. Deng. "Anonymous Secure Routing in Mobile Ad-Hoc Networks". In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN'04)*, pages 102–108, Tampa, U.S., Nov. 2004.

[90] P. R. Zimmermann. *"The Official PGP Users Guide"*. MIT Press, 1995.