Available online at www.sciencedirect.com
ScienceDirect
Journal of Network and Computer Applications 30 (2007) 937–954
Journal of NETWORK and COMPUTER APPLICATIONS
www.elsevier.com/locate/jnca

Secure and efficient key management in mobile ad hoc networks

Bing Wu[a,*], Jie Wu[a], Eduardo B. Fernandez[a], Mohammad Ilyas[a], Spyros Magliveras[b]

[a]*Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431, USA*
[b]*Department of Mathematics, Florida Atlantic University, Boca Raton, FL 33431, USA*

Received 28 July 2005; accepted 28 July 2005

Abstract

In mobile ad hoc networks, due to unreliable wireless media, host mobility and lack of infrastructure, providing secure communications is a big challenge. Usually, cryptographic techniques are used for secure communications in wired and wireless networks. Symmetric and asymmetric cryptography have their advantages and disadvantages. In fact, any cryptographic means is ineffective if its key management is weak. Key management is also a central aspect for security in mobile ad hoc networks. In mobile ad hoc networks, the computational load and complexity for key management are strongly subject to restriction by the node's available resources and the dynamic nature of network topology. We propose a secure and efficient key management (SEKM) framework for mobile ad hoc networks. SEKM builds a public key infrastructure (PKI) by applying a secret sharing scheme and using an underlying multi-cast server groups. We give detailed information on the formation and maintenance of the server groups. In SEKM, each server group creates a view of the certificate authority (CA) and provides certificate update service for all nodes, including the servers themselves. A ticket scheme is introduced for efficient certificate service. In addition, an efficient server group updating scheme is proposed. The performance of SEKM is evaluated through simulation.
© 2005 Elsevier Ltd. All rights reserved.

Keywords: Mobile ad hoc networks; Key management; Secret sharing; Security; Server group

*Corresponding author.
E-mail addresses: bwu@fau.edu (B. Wu), jie@cse.fau.edu (J. Wu), ed@cse.fau.edu (E.B. Fernandez), ilyas@fau.edu (M. Ilyas), spyros@fau.edu (S. Magliveras).

1084-8045/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2005.07.008

1. Introduction

A mobile ad hoc networks is a special type of wireless network in which a collection of mobile hosts with wireless network interfaces may form a temporary network, without the aid of any fixed infrastructure or centralized administration. In mobile ad hoc networks, nodes within their wireless transmitter ranges can communicate with each other directly (assume that all nodes have the same transmission range), while nodes outside the range have to rely on some other nodes to relay messages. Thus a multi-hop scenario occurs, where the packets sent by the source host are relayed by several intermediate hosts before reaching the destination host. Every node functions as a router. The success of communication highly depends on the other nodes' cooperation.

While mobile ad hoc networks can be quickly and inexpensively set up as needed, security is a more critical issue compared to wired or other wireless counterparts. Many *passive* and *active* security attacks could be launched from the outside by malicious hosts or from the inside by compromised hosts (Luo and Fang, 2003; Ilyas, 2003).

Cryptography is an important and powerful tool for security services, namely authentication, confidentiality, integrity, and non-repudiation. Cryptography has two dominant approaches, namely *symmetric-key* (secret-key) and *asymmetric-key* (public-key). There is a variety of symmetric or asymmetric algorithms available, such as DES, AES, IDEA, RSA, and EIGamal (Salomaa, 1996; Tanenbaum, 2003; Burnett and Paine, 2001). *Threshold cryptography* (Shamir, 1979) is a scheme quite different from the above two approaches. In Shamir's (k, n) *secret sharing* scheme, a secret is split into n pieces according to a random polynomial. The secret can be recovered by combining k pieces based on *Lagrange interpolation*. These cryptographic approaches are widely used in wired and wireless networks; obviously they could also be used in mobile ad hoc networks.

Key management is a basic part of any secure communication. Most cryptosystems rely on some underlying secure, robust, and efficient key management system. Secure network communications normally involve a key distribution procedure between communication parties, in which the key may be transmitted through insecure channels. A framework of trust relationships needs to be built for authentication of key ownership in the key distribution procedure. While some frameworks are based on a centralized *trusted third party* (TTP), others could be fully distributed. For example, a *certificate authority* (CA) is the TTP in PKI, a *key distribution center* (KDC) is the TTP in the symmetric system, while in PGP no such a trusted entity is assumed.

We introduce here a secure and efficient key management (SEKM) scheme. The major contribution of our scheme is that SEKM is designed to provide efficient share updating among servers and to quickly respond to certificate updating, which are two major challenges in a distributed CA scheme. The basic idea is that server nodes form an underlying service group for efficient communication. For efficiency, only a subset of the server nodes initiates the share update phase in each round. A ticket-based scheme is introduced for efficient certificate updating. Normally, because of share updating, recently joining servers could be isolated from the system if they carry outdated certificates. Our scheme does not isolate new servers, and is open for regular nodes for easy joining and departing. SEKM creates a view of CA and provides secure and efficient certificate service in the mobile and ad hoc environment. The framework of SEKM is described in Section 4.

This paper is organized as follows: Section 2 reviews related work. Section 3 discusses the key management and trust model in mobile ad hoc networks. Details of the SEKM

scheme are described in Section 4. A performance evaluation of the proposed approach is conducted in Section 5. In Section 6, we conclude the paper and discuss possible future work. Throughout the paper, we use the terms node and host interchangeably.

2. Related work

Recently, security has become a hot research topic in mobile ad hoc networks. Several secure routing protocols have been proposed in the literature. For example, SRP (Papadimitratos and Haas, 2002), SEAD (Hu et al., 2002), and SAODV (Zapata, 2002) address security attacks in routing protocols and propose different means to counter particular threats. However, almost all of them rely on the existence of a public-key management system. Even in TESLA (Perrig et al., 2000), delivery and authentication of the first element in a hash chain requires an asymmetric key management framework. So, the existence of an effective key management framework is fundamental to secure routing protocols. There are some other research papers which focus on either secure data transmission, intrusion detection, or key management in mobile ad hoc networks. Although these topics are closely related, we emphasize key management and ignore the rest of them here. We will address those topics in future work.

Zhou and Haas (1999) presented a secure key management scheme by employing (t, n) *threshold cryptography*. The system can tolerate $t-1$ compromised servers. However, this scheme does not describe how a node can contact t servers securely and efficiently when the servers are scattered in a large area. A share refreshing scheme is proposed to counter mobile adversaries. However, efficient and secure distribution of secret shares is not addressed.

Luo and Lu (2004) proposed a localized key management scheme called URSA. In their scheme *all* nodes are servers. The advantage of this scheme is the efficiency and secrecy of local communication as well as system availability; on the other hand, it reduces system security, especially when nodes are not well protected physically. One problem is that when the threshold k is much larger than the network degree d, nodes will have to keep moving to get their certificates updated. The second critical issue is convergence in the share updating phase. Another critical issue is that too much off-line configuration is required before accessing the networks.

Yi et al. (2002) put forward a scheme called mobile certificate authority (MOCA) key management. In their approach, certificate service is distributed to MOCA nodes, which are physically more secure and powerful than other nodes. In their scheme, a node could locate $k+\alpha$ MOCA nodes either randomly, through the shortest path, or based on the freshest path in its route cache. But the critical question is how nodes can discover those paths securely since most secure routing protocols are based on the establishment of a key service.

Capkun et al. (2003) considered a fully distributed scheme that has the advantage of configuration flexibility. However, it lacks any trusted security anchor in the trust structure. Many certificates need to be generated. Every node should collect and maintain an up-to-date certificate repository. Certificate chaining is used for authentication of public keys. The *certificate graph*, which is used to model this web of trust relationship, may not be strongly connected, especially in the mobile ad hoc scenario. In that case nodes within one component may not be able to communicate with nodes in different components. Certificate conflict is just another example of a potential problem in this scheme.

Recently, Yi and Kravets (2004) provided a composite trust model. In their scheme they combine the central trust and the fully distributed trust models. This scheme takes advantage of the positive aspects of two different trust systems. Actually, it is a compromise between security and flexibility. Some authentication metrics, such as confidence value, are introduced in order to glue two trust systems. However, proper assignment of confidence values is a challenge.

In summary, the schemes proposed in (Zhou and Haas, 1999; Yi et al., 2002; Luo and Lu, 2004) are based on the secret sharing technique. Zhou and Haas (1999) focuses on the share updating procedure. Yi's scheme (Yi et al., 2002) emphasizes efficient communications among MOCA nodes. Luo's approach (Luo and Lu, 2004) addresses the problem of share updating and certificate service in a localized environment. (Capkun et al., 2003) discusses the problem of key repository maintenance and certificate chaining in a fully distributed way.

3. Key management in ad hoc networks

Key management is a basic part of any secure communication structure. Most secure communication protocols rely on a secure, robust, and efficient key management system. General key management primitives and trust models are described below.

3.1. Key management primitives

The key is a piece of input information for cryptography algorithms. First, if the key is discovered, the encrypted information can be revealed. The secrecy of the private key must be assured locally. The *key encryption key* (KEK) approach could be used at local hosts.

Second, key distribution and key agreement over an insecure channel are risky and suffer from potential attacks. In the traditional digital envelope approach, a session key is generated at one side encrypted by the public-key algorithm, and then delivered and recovered at other end. In the *Diffie–Hellman* (DH) scheme, the communication parties at both sides exchange some public information and generate a common session key. Several enhanced DH schemes have been proposed to counter the *man-in-the-middle* attack. Many complicated key exchange or distribution protocols and frameworks have been designed and built. However, in mobile ad hoc networks the computational load and complexity of the key agreement protocol are strongly restricted by the node's available resources, the dynamic network topology, and network synchronization difficulty.

Third, key integrity and ownership should be protected from key attacks. Digital signatures, message digests and hashed message authentication codes (HMAC) are techniques used for data authentication or integrity purposes. Similarly, the public key is protected by public-key certificates, in which a trusted entity, called a certificate authority in PKI, vouches the binding of the public key with the owner's identity. In systems which lack a TTP, public-key certificates are vouched by peer nodes in a distributed manner, such as is done in *pretty good privacy* (PGP). Obviously, a certificate cannot prove whether an entity is "good" or "bad", but only the ownership of a key, i.e., it is for key authentication purposes.

Fourth, the key could be compromised or disposed after a certain period of usage. Since the key should no longer be usable after its disclosure, some mechanism is required to enforce this rule. In PKI, this can be done implicitly or explicitly. A certificate contains a

lifetime of validity; it is not useful after an expiration date. But in some cases, the private key could be compromised during the validity period, which would require the CA to revoke a certificate explicitly and notify the network by posting it onto a *certificate revocation list* (CRL) to prevent its usage.

3.2. Trust models

The authentication of key ownership is the first step for secure communication. Otherwise, it is easy to forge or spoof someone's key. Some trusted framework must be present to verify the key ownership. For PKI in the public-key cryptosystem, there are two dominating trust models, namely, centralized and web-of-trust trust models. For network scalability, the centralized trust model could be a hierarchical trust structure instead of a single CA entity. Multiple CA roots could be necessary for a large network, such as the Internet. There are two major variations proposed in ad hoc networks, which we name CA-view and hybrid trust models. The hybrid model glues the centralized and the distributed trust together (Yi and Kravets, 2004). See Figs. 1(a)–(d) for different trust models.

In the figures, all nodes within the circle form a network domain. In Fig. 1(a), there is one entity (in black) who is trusted by all nodes within the domain. In Fig. 1(b), there is no entity trusted by all hosts in the network domain, instead peer nodes trust each other and produce "certificates" based on local trust. Fig. 1(c) shows that quorum nodes (in gray) collaboratively create a view of the CA, which functions as the CA within the domain. The quorum nodes jointly produce the certificate. Fig. 1(d) shows a combination of (a) and (b) where some nodes are certified by the central CA (in black), and some are certified by peer nodes. For example, nodes 8 and 12 are CA certified, node 9 is not certified by the CA but by node 8. Node 13 is not trusted by any node within the domain. The confidence value of CA trust is higher than the value of the peer trust. For example, the value of a solid trust line is higher than that of a dashed line. Each trust line could have different values. Of course, this hybrid trust mode could have further variations. For example, the central CA could be distributed to a quorum of nodes.

Obviously, in mobile ad hoc networks, a framework for key management built on a fully centralized mode is not feasible, not only because of the difficulty to maintain such a

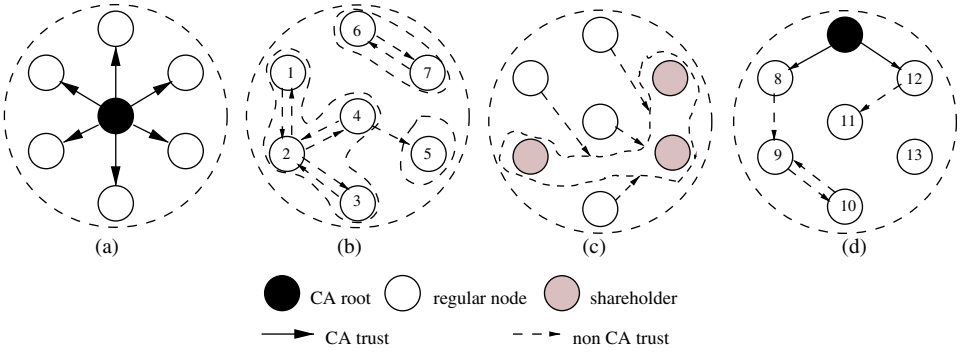


Fig. 1. Trust models.

globally trusted entity but also because the central entity could become a hot spot of attacks. Meanwhile a completely distributed model may not be acceptable because there is no well-trusted security anchor available in the whole system. One feasible solution is to distribute the central trust to multiple (or entire network) entities based on a *secret sharing* scheme. In SEKM, the system public key is distributed to the whole network, while the system private key is split between all server nodes. The server group creates a view of a CA in PKI for mobile ad hoc networks.

3.3. Secret sharing

In a mobile ad hoc network environment, a single CA node could be a security bottleneck if it is not well protected. Multiple replicas of CA are fault tolerant, but the network is as vulnerable to break in as single CA or even worse since breaking one CA means breaking all CAs, while it could be much easier for attackers to locate a target. An elegant secret sharing scheme is proposed in mobile ad hoc networks with different implementations. To better understand this scheme, a short overview is given here. A system-wide secret is distributed to multiple nodes. No single node knows or can deduce the secret from the piece it holds. Only a threshold number of nodes can deduce the secret. The study and proof of the basic algorithms are in (Shamir, 1979; Herzberg et al., 1995; Felman, 1987; Stadler, 1996). Some algorithms have been proposed to enhance basic secret sharing schemes (Zhou and Haas, 1999; Desmedt and Jajodiay, 1997; Wong et al., 2002; Shoup, 2000). For example, providing a way for a shareholder to verify the validity of a received share, periodically updating shares, share recovery and partial certificates, etc., which are implemented in SEKM with proper modification and are given in Section 4. In summary, the secret sharing algorithms make it feasible to reduce trust and adapt to the distributed and unreliable environment of mobile ad hoc networks. This is the main reason that we adopt these techniques in SEKM.

4. Secure and efficient key management (SEKM) scheme

4.1. Notations and assumptions

Some notation used in SEKM is introduced below. We assume that every node carries a valid certificate from off-line configuration before entering the network. A smart card can be used for this preconfiguration. The format of a certificate is similar to the X.509 structure with two extra attributes defined as *server flag* and *share version*. The server flag is set to 1 for servers and 0 for non-servers. The share version is also set to 1 for servers and 0 for non-servers. Version is increased by 1 after every share updating. Each server has its secret share stored in an encrypted format such as in password-based or KEK schemes. Each server also has a copy of the encrypted share verification parameters $\{g^{K_{ca}^{-1}} \bmod p, g^{\alpha_1} \bmod p, g^{\alpha_2} \bmod p, \ldots, g^{\alpha_{k-1}} \bmod p\}$. Some notations are listed in Table 1.

The structure of a certificate is:

ID_i	T_{valid}	K_i	*flag*	*ver*	*sign.*	*issuer*	*algo.*

Table 1
Some notation

t_s	Timestamp
n_s	Nonce, one time random number
ID_i	Node i's identity
K_i/K_i^{-1}	Node i's public key/private key pair
K_{ca}/K_{ca}^{-1}	CA's public key/private key pair
m	Control message or data
$(m)^{K_i}$	m is encrypted by node i's public key
$(m)^{K_i^{-1}}$	m is signed by node i's private key
$h(m)$	The digest of m
k	Threshold value
S_i	Node i's secret share
$Cert_i$	Certificate of node i's public key

4.2. Overview of SEKM

In the SEKM framework, K_{ca}^{-1} is distributed to m shareholders. Normally, the number of shareholders is significantly less than the total number of nodes (n) in the network. For example, 20–30% nodes are secret shareholders. We name these shareholders as CA-view or server nodes in short. They are basically normal nodes except that they hold a system private key share and are capable of producing partial certificates. A quorum of $k(1<k\leqslant m)$ servers can produce a valid certificate. It is easier to connect all servers and form a special group rather than to search each one of them separately and frequently. This arrangement is communication-efficient, bandwidth-saving, and easy for management. From a node point of view, it is easier to locate the server "block" rather than each "point". From the server point of view, it is easier to coordinate within the group rather than the entire network. We name this special group as a multicast server group, or server group in short, though it is quite different from the traditional source–receiver multicast groups. This server group consists of server nodes and forwarding nodes. The forwarding nodes within the group are regular nodes. The framework of SEKM consists of several phases, namely server group formation phases, group maintenance phases, share updating phases, certificate renew/revocation phases, and handling new server nodes phases. A substructure snapshot of a server group is illustrated in Fig. 2. The substructure of a server group in essence creates a view of CA for certificate services and efficient share updating. For simplicity, we state that server groups produce certificates without explicitly excluding the non-server forwarding nodes.

4.3. Secure server group formation and maintenance

In the server group formation phase, the SEKM scheme is similar to the existing on-demand multicast routing protocol (ODMRP) (Lee et al., 2002). ODMRP is an on-demand protocol, where a source-rooted or receiver-rooted forwarding group is formed based on periodical Join-Data and Join-Table messages. Rather than a tree structure, a mesh structure is maintained to forward multicast data. However, the difference is that the group formation phase is secure and there is no specific source and receiver in SEKM; instead, only server nodes initiate the group formation and become members of the group

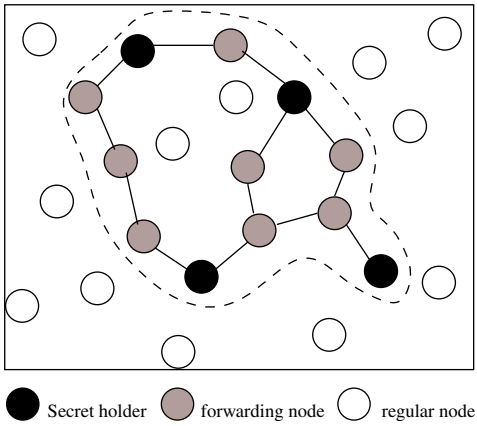


Fig. 2. Server group substructure snapshot in SEKM.

in their lifetime. A subset of non-server nodes could be forwarding nodes in a certain period, and become part of the group. The structure of the server group is a mesh with a soft state maintenance scheme. In general, the mesh structure is more stable than the traditional tree structure. Although the tree-based structure is more communication efficient, it is easy to break in a high dynamic situation and incurs excessive control traffic for link recovery. Maintaining the connection of the server group is essential for the normal operation of SEKM. It is necessary to maintain at least a quorum of server nodes connected by periodically sending control packets before some link is broken. So, it is a *soft state* maintenance scheme rather than a *hard state* approach. In this paper, we assume that the network is a connected graph and one server group is maintained. In our future work we will consider the scenario where there are multiple server groups.

4.3.1. Group creation

The server group formation procedure consists of a request phase and a reply phase. When a secret shareholder enters the network, it broadcasts a server advertising packet, which is called *JoinServeReq* and is done in a scoped flooding way. Only the server nodes can initiate the *JoinServeReq* packet, which is enforced by the server *flag* attribute in certificates. By doing this we can prevent malicious nodes from flooding the join request packet. The *JoinServeReq* packet contains message m which includes $\{ID_i, SEQ_i, TTL\}$ together with its hashed signature $\{[h(ID_i, SEQ_i)]^{K_i^{-1}} | (TTL)^{K_i^{-1}}\}$ (symbol | denotes concatenation). Node i could attach its certificate $\{Cert_i\}$ for the first time. When a node receives a non-duplicate *JoinServerReq* packet, it needs to verify that the packet is from the authenticated source, and without any change except for the TTL field. The TTL value decreases by 1 as the packet leaves the node. The change of TTL is signed by intermediate nodes and verified by neighbors. The packet is discarded if any of those conditions is not satisfied. After verification, the routing table is updated based on the information contained in the message and through the route backward learning process. The server certificate could also be stored in this table. Nodes that receive a valid *JoinServerReq* will rebroadcast the *JoinServerReq* packet if TTL is >0. A compromised node could modify the TTL field unpredictably but the misbehavior is assumed to be monitored by neighbors.

Also, the (ID_i, SEQ_i) pair can help to identify and discard the duplicate packet. If the node is a server, it will send a *JoinServerReply* packet as well as forwarding the request packet. Similar to the *JoinServerReq* packet, a *JoinServerReply* packet is also protected by the replier's signature. The server could delay for a while before it sends out a reply message so that a better path could be selected based on certain metrics, or the server could build multiple paths by sending reply messages to multiple upstream neighbors.

When a node receives a *JoinServerReply* it checks the validity of the packet first. After verification, the node could update its routing table based on the forwarding learning process. If the next hop field matches its own ID it will mark itself as a forwarding node and forward the reply based on the routing table. Note that a server node could be a forwarding node as well if it is on the shortest path between a pair of servers. The procedure continues until the reply reaches the initial request server. Thus, all server nodes together with the forwarding nodes form a mesh structure. Detailed examples are described below.

4.3.2. Examples

In the example, nodes 1, 2, 16, and 22 are servers. Fig. 3(a) shows the *JoinServeReq* initiated by server node 1 and gives a snapshot of the dissemination of the request and reply messages. When a node receives a request packet, it checks first the validity of the packet before taking any further actions. It also discards duplicate, non-authenticated, or illegally altered packets. In the example, we assume that the validity of all packets in the process are verified. After neighbor nodes 14, 18 and 20 receive the request they rebroadcast it. This process continues at other nodes. When server 16 receives the packet from node 21 first, it could send back a *JoinServerReply* message to node 21 instantly, or it could delay for a while until it receives the same request from node 12. Server 16 could send replies to both nodes 21 and 12 in order to enforce multiple paths. Node 16 rebroadcasts the join request message if the TTL is more than 0. When node 21 receives the join reply packet from node 16 it learns that (1) node 16 is a server node and (2) it is on the

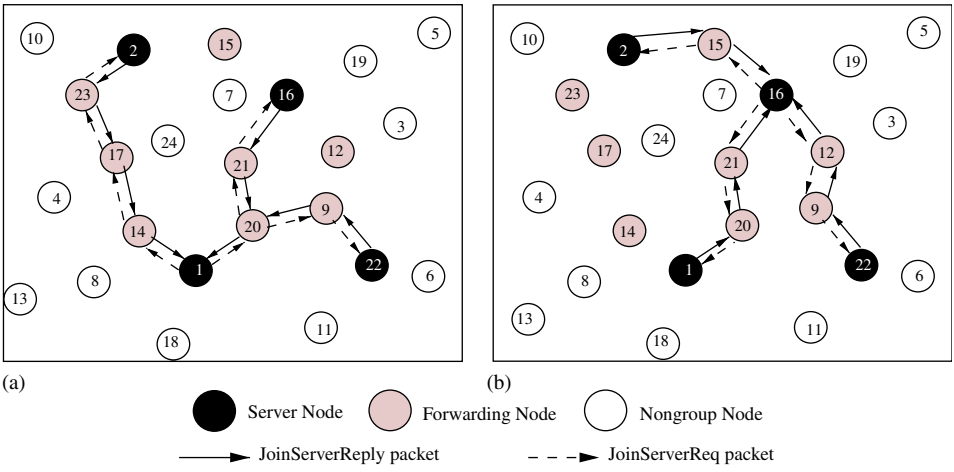
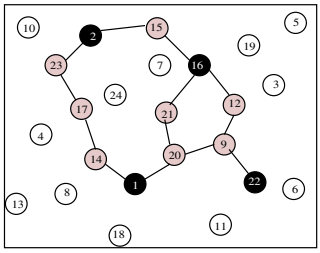


Fig. 3. Server group setup illustration. (a) Node 1 initiates the join request packet; (b) node 16 initiates the join request packet.



(a)

Server 16's table

Server	Next Hop
1	12, 21
2	15
16	16
22	12, 21

(b)

Server Node Forwarding Node Nongroup Node

Fig. 4. Server group mesh and table snapshot.

selected path between server 1 and server 16, and thus (3) sets its forwarding *flag* and updates its forwarding table. The same process happens on nodes 12 and 20. Eventually, the join reply packet from node 16 reaches server node 1, which is the join request initiator. When server 1 receives the reply packet it learns that server 16 is reachable through neighbor 20. It updates the routing table entry. After a certain amount of time, replies from all servers arrive at node 1. Node 1 has the knowledge of all reachable servers. Fig. 3(b) shows the join procedure initiated from server 16. After all servers finish the join procedure the group mesh structure is formed, and each server has a routing table established. Fig. 4(a) shows the server group mesh and Fig. 4(b) shows a snapshot of the table created by node 16. Every server node maintains a table and the table is referenced for subsequent certificate service phases and share updating phases.

4.3.3. Group maintenance

The server group structure should be maintained during the entire lifetime of the network. The mesh structure is more reliable than the conventional tree structure where there is only one path available between any pair of servers. However, for a mesh structure, there are multiple possible paths between pairs of servers. Thus if one link is broken, an alternative link could be utilized instead of launching a costly recovery procedure. In SEKM, the periodical message *JoinServerRequest* and *JoinServerReply* are sent out in order to refresh the server group. Thus, a soft state scheme is adopted to react to the dynamic network topology and possible link breaks. Since this soft state scheme is quite expensive for a large network, the frequency for refreshing should be scheduled carefully according to node mobility.

4.4. Secret share updating

Every server node has a piece of the system secret key K_{ca}^{-1}. Although the S_i is stored at local storage protected in encrypted format by some means, there is still the risk that the node is captured and compromised, revealing the secret share. Once a *mobile attack* compromises enough shares the system secret is disclosed. In order to counter these types of attacks, a periodical share updating scheme is proposed in some papers using different implementations. In SEKM, updating the shares held by servers is quite simple. The idea is that only threshold k servers within the server group initiate the share update phase.

We name these servers *active servers* for convenience. Active servers generate new shares and send them to the corresponding servers in the group. Obviously, active servers consume more energy than non-active ones. In order to avoid some servers continuing to work as active servers, this phase is operated in rounds and teams of servers are "selected" as active servers alternatively. Similar to the scheme in Heinzelman et al. (2000), the update phase is broken into rounds. A percentage is defined as $\gamma = \lfloor \frac{k}{m} \rfloor$. At every round every server i generates a random number β_i between 0 and 1 and a threshold value τ_i. τ_i is defined as

$$\tau_i = \begin{cases} \dfrac{\gamma}{1 - \gamma * (r \bmod \frac{1}{\gamma})} & \text{if } i \in SG, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where r is the current round, and SG is the set of server nodes that have not ever initiated update phases in the $\frac{1}{\gamma}$ rounds. During round $0(r = 0)$, each server has a probability γ to initiate the share update. If $\beta_i < \tau_i$, this server will become an *active server*. In each round there are about k active servers. The algorithm for the share update phase is shown below.

Share Updating:

1. Each active server i randomly selects a $(k-1)$-degree polynomial $g_i(x) = (\beta_{i,1}x + \beta_{i,2}x^2 \cdots + \beta_{i,k-1}x^{k-1}) \bmod p$, or $g_i(x) = \sum_{d=1}^{k-1} \beta_{(i,d)}(x)^d \bmod p$ in short. Note: $g_i(0) = 0$.
2. Server i broadcasts the witness for polynomial coefficients $\{g^{\beta_{i,d}} : |1 < d < k\}$ and its hashed signature $\{[h(g^{\beta_{i,d}})]^{K_i^{-1}} : |1 < d < k\}$ to the server group.
3. Each active server i computes a share for server j, with $S_{i \to j} = g_i(j) \bmod p$, which is encrypted with $j's$ public key K_j, then sent to the corresponding server $j(1 \leqslant j \leqslant k)$ in the form of $\{[S_{i \to j}]^{K_j}\}$.
4. Each server will receive about k new shares. It decrypts each new share, checks its validity and combines k new shares with its old share to produce the final new share. Server j's new share $S_j' = S_j + \sum_{i=1}^{k} S_{i \to j}$. The new share will replace the old share as the new partial certificate signing key.

4.5. Handling new servers

In SEKM, new servers can join the network while some servers may leave the network. In case a server leaves the network, the soft state server group maintenance mechanism can handle the change of server group topology. However, when a new server joins the group, some mechanism is required to handle a possible share inconsistency. As we know the server group updates shares periodically, a new joining node could carry an outdated share from off-line configuration. In order to handle this situation the new node r needs to contact at least k servers to "catch up" with the latest server group with a renewed share. As we described above, a new node sends the *JoinServerReq* message the first time it enters the network. The server group checks the incoming join group request. A message could be sent out to notify requesting node r by checking the *version* field in the certificate. After that, a share renewing process will be launched. The algorithm is shown in the next page.

4.6. Certificate updating

There is an attribute, T_{valid}, defined in the public-key certificate. A certificate is only valid for a period of time after being issued. Each node (including servers) needs to periodically update its certificate before expiration. A node needs to get at least a threshold (k) number of *partial certificates* to reconstruct a valid certificate. It is advantageous to update certificates based on the server group structure in SEKM. Once a server node receives a *CertUpdateReq* and verification of the request, instead of sending the request to all server group nodes, it attaches a ticket and just sends the request to *sufficient* $k+\Delta$ servers. Δ is the marginal safety value in case some partial certificates are corrupted. Since each server knows the path to all other server group members, it is wise to utilize the ticket scheme. Here the ticket is basically used as a counter. The ticket could be split at intermediate nodes. For a small server group, broadcasting certificate requests within the group is good enough. But for a large server group with $m \gg k$, broadcast requests to all servers cause significant processing and bandwidth waste.

Handling a new server:

1. The receiving server in the server group locates a subset of servers ($\omega : |\omega| \geqslant k$). Each server $i (i \in \omega)$ randomly chooses a polynomial $f_i(x)$ with degree $k-1$, where $f_i(r) = 0$ and $f_i(0) \neq 0$.
2. Each server node i broadcasts the witnesses for coefficients while it distributes share $f_i(j)$ to the corresponding server $j (j \in \omega)$, encrypted with K_j.
3. Server j receives shares $f_i(j) (i \in \omega)$, combines them with share $h(j) = S'_j + \sum_{i \in \omega \setminus \{r\}} (f_i(j))$ (here S'_j is j's current shares), and sends the resulting share to the requesting server r in encrypted format.
4. Server r decrypts these shares and interpolates them to renew S'_r using the secret reconstruction algorithm described in Section 3.

Take Fig. 4(a) as an example. Assume k is 3. When server node 1 receives a certificate updating request from a regular node or from itself, it could produce a partial certificate itself, while it sends two tickets attached to the *CertUpdateReq* message to node 20 and no ticket to node 14. These two tickets would be split into two separate tickets with one ticket being sent to node 9 and one going to node 21. Eventually, the 2 tickets reach server nodes 16 and 22. There are many other options to split the tickets. Note that the ticket is used within the server group, and it is transparent for the non-group members. Any secure routing protocol could be used to find a path from the requesting node to the server group before it sends out the *CertUpdateReq* message. The *CertUpdateReq* message m' should be signed by the original requester i. It includes $\{ID_i, SEQ_i\}$ together with its hashed signature $\{[h(m')]^{K_i^{-1}}\}$. Similar to the procedure of processing *JoinServeReq*, intermediate verification is required. The intermediate nodes on the path relay the *CertUpdateReq* message until it reaches the server d, where a ticket is generated and processed within the server group. The algorithm is shown below.

Certificate updating:

1. The receiving server d on the server group produces $k + \Delta$ tickets attached to requester i's request packet.
2. Each server j that receives a ticket produces a partial certificate for requester i. $Cert_{j\to i} = (K_i)^{S_j * l_j(0)} \bmod p$ and sends it back to server d.
3. Server j decreases ticket by 1, and splits it if necessary, then forwards the request together with the ticket.
4. Server d combines k partial certificates into one certificate $Cert_i = \prod_{j=1}^{k} Cert_{j\to i} = \prod_{j=1}^{k} K_i^{S_j * l_j(0)} \bmod p = (K_i)^{\sum_{j=1}^{k} S_j * l_j(0)} \bmod p = K_i^{K_{ca}^{-1}}$, and sends it back to the requesting node i. Note: server d could send $k + \Delta$ partial certificates back to requesting node i and let the certificate be combined at i instead of at server d.

4.7. Handling certificate expiration and revocation

A certificate will expire after a predetermined period of time. A node with an invalid certificate is prevented from participating in any network activity. In SEKM, nodes need to update the certificates before expiration. It is possible that a node could recover its expired certificate from the server group based on certain criteria. In this paper, for simplicity, a node with an expired certificate needs some off-line or in-person reconfiguration.

A node's certificate could be revoked by the server group within its validity period for several reasons. A server node could be compromised, and thus initiate inconsistent shares during the share updating/renewing phase. A node could refuse to issue certificates or issue wrong partial certificates for other nodes. A non-server node could misbehave in relaying the join request/reply messages for maintaining the server group; or in the phase of certificate service, routing information dissemination or data transmission. In the occurrence of any misbehavior or malicious attacks, an accusation with the signature of the initiator should be sent to the server group. Once the server receives the accusation, it checks the validity of the packet first; if verified, it marks the certificate state of the accused node as *suspect*. There is a counter and timer associated with it. The counter could decrease after a certain amount of time. Once the counter accumulates to a threshold value υ within the predefined time period ρ, a collaboration of k servers can revoke the accused node's certificate. The revoked certificate is put onto the *CRL*. The *CRL* is broadcast to the entire network periodically. Some information associated with the accuser must be stored in the server's database to prevent abuse of accusations. A node with a revoked certificate needs reconfiguration before reentering the network.

4.8. Summary

In summary, a server group is formed securely and stays connected. The certificate updating request is processed by the server group in a ticket-based approach. The system secret held by each server is refreshed periodically in a fair and efficient way. New joining servers with outdated shares could be renewed. Node's misbehavior is monitored and

could be accused by other network nodes. A certificate can be revoked by the server group. Nodes with expired or revoked certificates need off-line reconfiguration.

5. Performance evaluation

In this section, we analyze the performance of the SEKM scheme. The simulation was implemented in Matlab. The simulation was conducted in a 100×100 2-D free-space by randomly allocating a given number of nodes ranging from 40 to 100 nodes. We assume every node has the fixed transmission range $r = 25$. A unit disk graph is randomly generated, where the node connections depend on node distances. Two nodes are neighbors if their distance is within each other's transmission range.

We implemented 1024-bit RSA cryptographic key pairs. The system secret is distributed to all servers based on a randomly generated polynomial. The coefficients of the polynomial are 512 bits long. The structure of certificates is based on Section 4(A) where all fields, such as ID, time stamp and flags are concatenated, hashed using MD5, and then signed by the system secret key or shares. The witnesses of the polynomials are generated using a public generator $g = 2$ with the module n (1024 bits). The broadcast of witnesses is for the purpose of share verification. Partial certificates are generated and combined according to the algorithm described in Section 4(F). We used a Math Toolbox included in Matlab for handling large numbers.

In the simulation, we analyzed the average distance from a node to the server group, the average size of the forwarding nodes, the average delay for certificate verification in the group formation phase, and the convergence time for the parallel sharing updating, partial certificate generation/combination among server nodes. We also conducted the analysis for the impact of the key length on the convergence time in the parallel share updating phase. In the current simulation, we ignored the communication cost and network delay. We conducted four sets of experiments according to the percentage of server nodes p, where $p = 20\%$, 30%, 40%, and 50%. Another four sets of experiments are based on the percentage of server threshold tp, where $tp = 50\%$, 60%, 80%, and 100%. For instance, if the total number of nodes is 100, $p = 20\%$, and $tp = 60\%$, then there are 20 server nodes, and the threshold is 12. Four sets of key sizes were evaluated, which are 256, 512, 1024, and 2048 bits. The simulation results are shown from Figs. 5–7.

Fig. 5(a) shows the network parameters of the randomly generated connected graph for the case when p is 30%. The average distance (in hops) between a pair of nodes (labeled as Overall) is about 3.2, which is close to the average number of hops between server nodes within the server group (labeled as Within group). It also shows that the average distance from a node to the server group (labeled as To group) is almost equal to 1 hop. This is an obvious advantage of the SEKM scheme. It proves that both accessibility and delay are improved by requesting the server group rather than contacting each server individually. Fig. 5(b) shows that the average distance from a node to the server group is close to 1 hop, unrelated to the server rate p. Fig. 6(a) shows the average number of forwarding nodes, which connect the server nodes and form the server group, with all server nodes. The size of the forwarding nodes shrinks as the server rate p increases while the total number of nodes in the network is more than 50. But as we can see in the following experiment, the higher the server rate p, the longer the computation delay.

Fig. 6(b) shows the average computation time for the certificate verification during the server group formation phase. As we know in the SEKM scheme, any server node must

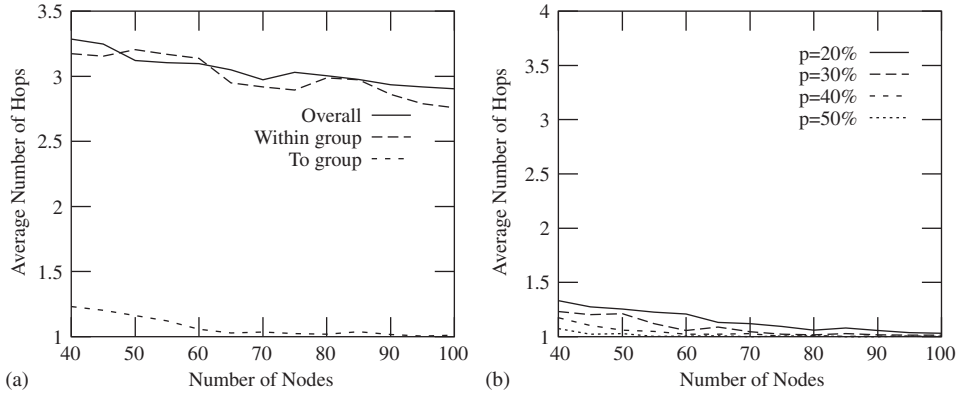


Fig. 5. (a) Average distance for a connected graph with $p = 0.3$. (b) Average distance to the server group.

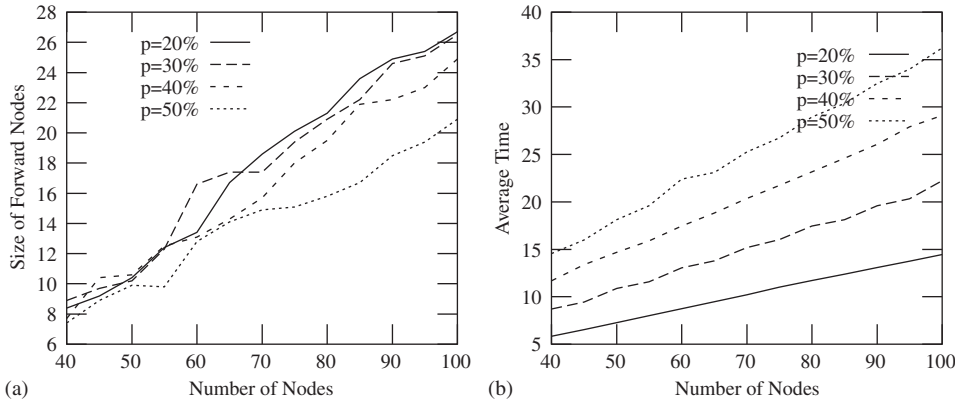


Fig. 6. (a) Average size of forwarding nodes. (b) Average delay of server group formation.

carry a valid certificate before joining the server group and participating in network activity thereafter. This obviously raises the computation load since every server node needs to apply the verification process for every other server node who is willing to join the group. From Fig. 6(b) we can see that the average delay increases almost linearly in accordance with the number of network nodes. It also shows that the higher the server rate p, the higher the delay. For instance, when the total number of nodes is 100, the delay is about 14 s for 20 servers ($p = 20\%$), 22 s for 30 servers ($p = 30\%$), 29 s for 40 servers ($p = 40\%$), and 36 s for 50 servers ($p = 50\%$).

Fig. 7(a) shows the average convergence time for the parallel share updating phase when the server rate p is 30%. Here, the delay is the sum of all time components, which includes the generation of random polynomials, evaluation of sub-shares, encryption of the polynomial coefficients, and combination of all sub-shares. Since only the threshold number of servers participate in all the computations, the larger the threshold, the longer

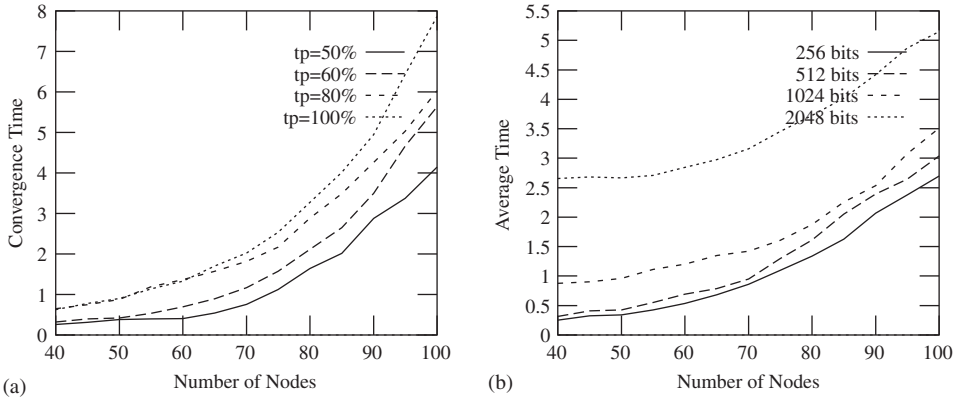


Fig. 7. (a) Average convergence time of share updating with $p = 0.3$. (b) Computation time of different key lengths with $p = 0.3$, $tp = 0.6$.

the computation time. As an example, when the total number of nodes is 100, among them 30 are servers. If the threshold is 15 ($tp = 50\%$), the average convergence delay is 4 s, about 6 s for thresholds of 18 and 24 ($tp = 60\%$, 80%), and about 8 s when the threshold is 30 ($tp = 100\%$).

Fig. 7(b) shows the impact of the selection of key size on the computation time when $p = 30\%$, $tp = 60\%$. By intuition, we can imagine that the longer the key size, the higher the security strength, and the more computation it requires. From Fig. 7(b) we can see that the difference of computation time is not quite significant when the key size increases from 256 bits to 1024 bits. However, the computation time almost doubles when the key size increases to 2048 bits. Obviously, it is a trade off between security strength and computation delay. In practice, the selection of key lengths of 512 bits or 1024 bits is appropriate for most circumstances.

6. Conclusion

Security is an important issue for mobile ad hoc networks. For security we mainly consider the following attributes: availability, confidentiality, integrity, authentication, authorization and non-repudiation. Several security mechanisms and protocols have been designed and proposed for mobile ad hoc networks. Key management is the central aspect of the security of mobile ad hoc networks, and it is still a weak point. In this paper we propose a key management scheme, SEKM, which creates a PKI structure for this type of network in mobile ad hoc networks. In SEKM, server nodes form a mesh-based server group. SEKM is based on the secret sharing scheme, where the system secret is distributed to a group of server nodes. The server group creates a view of a CA. The advantage of SEKM is that it is easier for a node to request service from a well maintained group rather than from multiple "independent" service providers which may be spread in a large area. It is much easier for servers to coordinate within the group rather than with the entire network during the secret share updating phase.

A detailed SEKM framework and operational phases are described in this paper. In SEKM, the server group provides a certificate update service for all nodes including the servers themselves. A ticket scheme is introduced for efficient certificate service. In addition, an efficient server group periodical updating scheme is proposed. Simulations show that both accessibility and efficiency for certificate services and share updating are achieved. The security of SEKM is ensured by the entire framework of threshold cryptographic primitives. In our future work, we will extend SEKM to multiple server groups in large networks including partitioned networks.

Acknowledgements

This work was supported in part by NSF Grants CCR 0329741, CNS 0422762, CNS 0434533, ANI 0073736, EIA 0130806, and by a federal earmark project on Secure Telecommunication Networks. The preliminary version of this paper appeared in the Proceedings of the 19th IEEE IPDPS, The First International Workshop on Security in Systems and Networks (SSN'2005) (Wu et al., 2005).

References

Burnett S, Paine S. RSA security's official guide to cryptography. RSA Press; 2001.

Capkun S, Buttyan L, Hubaux J. Self-organized public-key management for mobile ad hoc networks. IEEE Trans Mobile Comput 2003;2(1).

Desmedt Y, Jajodiay S. Redistribution secret shares to a new access structures and its application. University of Wisconsin-Milwaukee; 1997.

Felman P. A practical scheme for non-interactive verifiable secret sharing. Proceedings of the 27th IEEE symposium on the foundations of computer science. 1987. p. 427–37.

Heinzelman W, Chandrakasan A, Balakrishnan H. Energy-efficient communication protocol for wireless microsensor networks. Proceedings of the 33rd Hawaii international conference on system sciences, vol. 8. January 2000.

Herzberg A, Jarecki S, Krawczyk H, Yung M. Proactive secret sharing or: how to cope with perpetual leakage. Proceedings of Crypto'95, vol. 5. 1995. p. 339–52.

Hu Y, Johnson D, Perrig A. SEAD: secure efficient distance vector routing in mobile wireless ad-hoc networks. Proceedings of the 4th IEEE workshop on mobile computing systems and applications (WMCSA'02). 2002. p. 3–13.

Ilyas M. The handbook of ad hoc wireless networks. Boca Raton, FL: CRC Press; 2003.

Lee S, Su W, Gerla M. On-demand multicast routing protocol in multihop wireless mobile networks. ACM/Baltzer Mobile Networks and Applications, a special issue on Multipoint Commun Wireless Mobile Networks 2002;7:441–53.

Luo W, Fang Y. A survey of wireless security in mobile ad hoc networks: challenges and available solutions. In: Ad hoc wireless networking. Dordrecht: Kluwer Academic Publishers; 2003. p. 319–64.

Luo H, Lu S. URSA: ubiquitous and robust access control for mobile ad-hoc networks. IEEE/ACM Trans Networking 2004;12(6):1049–63.

Papadimitratos P, Haas Z. Secure routing for mobile ad hoc networks. Proceedings of the SCS communication networks and distributed systems modeling and simulation conference (CNDS 2002). 2002.

Perrig A, Canetti R, Tygar J, Song D. The TESLA broadcast authentication protocol. Internet Draft, July 2000.

Salomaa A. Public-key cryptography. Berlin: Springer; 1996.

Shamir A. How to share a secret. Commun ACM 1979;22(11):612–3.

Shoup V. Practical threshold signatures. Proceedings of Eurocrypt 2000. 2000. p. 207–20.

Stadler M. Publicly verifiable secret sharing. Proceedings of Eurocrypt'96. 1996. p. 190–9.

Tanenbaum AS. Computer networks. PH PTR; 2003.

Wong T, Wang C, Wing J. Verifiable secret redistribution for threshold sharing schemes. Technical Report CMU-CS-02-114-R, School of Computer Science, Carnegie Mellon University; September 2002.

Wu B, Wu J, Fernandez E, Magliveras S. Secure and efficient key management in mobile ad hoc wireless networks. Proceedings of the 19th IEEE international parallel and distributed symposium (IPDPS 2005). The first international workshop on security in systems and networks (SSN'2005). 2005. p. 288.

Yi S, Kravets R. Composite key management for ad hoc networks. Proceedings of the 1st annual international conference on mobile and ubiquitous systems: networking and services (MobiQuitous'04). 2004. p. 52–61.

Yi S, Naldurg P, Kravets R. Security-aware ad-hoc routing for wireless networks. Report No.UIUCDCS-R-2002-2290, UIUC, 2002.

Zapata M. Secure ad hoc on-demand distance vector (SAODV). Internet draft, draft-guerrero-manet-saodv-01.txt, August 2002.

Zhou L, Haas Z. Securing ad hoc networks. IEEE Network Magazine 1999;13(6):24–30.