Secure and Efficient Key Management in Mobile Ad Hoc Networks

Bing Wu, Jie Wu, Eduardo B. Fernandez
Dept. of Computer Science and Engineering
Florida Atlantic University
Boca Raton, Florida 33431
bwu@fau.edu, {jie, ed}@cse.fau.edu

Spyros Magliveras
Dept. of Mathematics
Florida Atlantic University
Boca Raton, Florida 33431
spyros@fau.edu

Abstract*— In mobile ad hoc networks, due to unreliable wireless media, host mobility and lack of infrastructure, providing secure communications is a big challenge in this unique network environment. Usually cryptography techniques are used for secure communications in wired and wireless networks. The asymmetric cryptography is widely used because of its versatileness (authentication, integrity, and confidentiality) and simplicity for key distribution. However, this approach relies on a centralized framework of** **public key infrastructure** ***(PKI). The symmetric approach has computation efficiency, yet it suffers from potential attacks on key agreement or key distribution. In fact, any cryptographic means is ineffective if the key management is weak. Key management is a central aspect for security in mobile ad hoc networks. In mobile ad hoc networks, the computational load and complexity for key management is strongly subject to restriction of the node's available resources and the dynamic nature of network topology. In this paper, we propose a secure and efficient key management framework (SEKM) for mobile ad hoc networks. SEKM builds PKI by applying a **secret sharing** ***scheme and an underlying multicast server group. In SEKM, the server group creates a view of the*** **certification 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.***

I. INTRODUCTION

Mobile ad hoc networks are special type of wireless networks 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 setup as needed, security is a 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 [10][12].

Cryptography is an important and powerful tool for security services, namely authentication, confidentiality, integrity, and non-repudiation. It converts readable data (*plaintext*) into meaningless data (*ciphertext*). Cryptography has two dominant flavors, namely *symmetric-key* (secret-key) and *asymmetric-key* (public-key) approach. In symmetric-key cryptography, the same key is used to encrypt and decrypt the information, while in the asymmetric-key approach, different keys are used to convert and recover the information. Although the asymmetric cryptography approach possesses versatileness (authentication, integrity, and confidentiality) and simplicity for key distribution, symmetric-key algorithms are generally more computation-efficient than the public-key approach. There is a variety of symmetric or asymmetric algorithms available, such as DES, AES, IDEA, RSA, and EIGamal [1][2][11]. *Threshold cryptography* [3] 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*. Secret splitting, reconstruction, and verification is quickly reviewed in Section 3. These cryptography tools 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. Key management deals with key generation, storage, distribution, updating, revocation, and certificate service, in accordance with security policies. Key management primitives and a trust model are presented in Section 3. The outline of key management is described below. First, secrecy of key itself must be assured in the local host system. Second, secure network communications involve key distribution procedure between communication parties, in which the key may be transmitted through insecure channels. Key confidentiality, integrity, and ownership must be enforced in the whole procedure. Third, a framework of trust relationships needs to be built for authentication of key ownership. While some frameworks are based on a centralized *Trusted Third Party* (TTP), others could be fully distributed. For example, a *Certificate Authority* is the TTP in PKI, *Key Distribution Center* (KDC) is the TTP in the symmetric system, meanwhile in PGP, no such a trusted entity is assumed.

Fourth, the key could be expired or have been revoked within its valid period.

We introduce here a secure and efficient key management scheme (SEKM). In SEKM, the system public key is distributed to the whole network. Like some schemes described in Section 2, in SEKM, the trust of the central authority (CA) is distributed to a *subset* of nodes (not *all* nodes), which could be nodes with normal or better equipment. 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 the 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 key management and trust model in mobile ad hoc networks. Details of the SEKM scheme are described in Section 4. In Section 5, we conclude the paper and discuss possible future work. Throughout the paper, we use terms node and host interchangeably.

II. Related Work

Recently, security has become a hot research topic in mobile ad hoc networks. Several secure routing protocols have being proposed in the literature. For example, SRP [22], SEAD [20], and SAODV [19] 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 [21], delivery and authentication of the first element in hash chain requires the 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, however we emphasize key management and ignore the rest of these topics here, we will address those topics in the future work.

Zhou and Hass [7] propose a secure key management scheme by employing (t, n) *threshold cryptography*. The system can tolerate $t - 1$ compromised servers. However, this scheme doesn't describe how a node can contact t servers securely and efficiently in case the servers are scattered in the whole area. A *share refreshing* scheme is proposed to counter *mobile adversaries*. However, efficient and secure distributions of secret shares are not addressed.

Luo, Kong, and Zerfos [9] propose a localized key management scheme called URSA. In this scheme *all* nodes are servers. The advantage of this scheme is efficiency and secrecy of local communication as well as system availability; on the other hand, it reduces the system security especially when nodes are not well protected. One problem is that in case the threshold k is much larger than the network degree d, nodes will have to keep moving to get certificates updated. The second critical issue is the convergence in the share updating phase. The third critical issue is that too much off-line configuration is required before accessing the networks.

Yi, Naldurg, and Kravets [8] propose a scheme called MOCA key management. In their approach, certificate service is distributed to Mobile Certificate Authority (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 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, Buttyan, and Hubaux [17] propose a fully distributed scheme. It has the advantage of configuration flexibility. However, it lacks of any trusted security anchor in the trust structure. Lots of certificates need to be generated. Every node should collect and maintain up-to-date a 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 ones in different components. *Certificate conflicting* is just another example of a potential problem in this scheme. A scheme with improved certificate repository maintenance could be introduced.

Recently, Yi and Kravets [18] propose a composite trust model. In their scheme they combine the central trust model and the fully distributed trust model. 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, assignment of *confidence value* properly is a challenge and problematic.

In summary, the schemes proposed in [7][8][9] are based on the secret sharing technique. Zhou focuses on the share updating procedure, Yi's scheme emphasizes efficient communications among MOCA nodes. Luo's approach addresses the problem of share updating and certificate service in a localized environment. Capkun discusses the problem of key repository maintenance and certificate chaining in a fully distributed way.

III. Key Management in Ad Hoc Networks

Key management is a basic part of any secure communication. Most secure communication protocols rely on the substantial secure, robust, and efficient key management system. Key management primitive and trust model are described below.

A. Key management primitive

Key is a piece of input information for cryptography algorithms. First, if the key is disposed, the encrypted information would be disclosed. The secrecy of the symmetric key and 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 is at high risk and suffers from potential attacks. In the traditional *digital envelop* approach, a session key is generated at one side and is encrypted by the public-key algorithm, then it is delivered and recovered at other end. In the *Diffie-Hellman* (DH) scheme, communication parties of both sides exchange some public information and generate a session key on both end. Several enhanced DH schemes have been invented to counter the *man-in-middle* attack. Many complicated key exchange or distribution protocols and frameworks have been designed and built. However, in mobile ad hoc networks the computation load and complexity of key agreement protocol is strongly restricted by node's available resource, dynamic network topology, or network synchronization difficulty.

Third, key integrity and ownership should be protected from advanced key attacks. *digital signature*, *message digest* and *hashed message authentication code* (HMAC) are techniques used for the data authentication or integrity purpose. Similarly, public key is protected by public-key *certificate*, in which a trusted entity called *certification authority* (CA) in PKI, vouches the binding of the public key with owner's identity. In systems where lacking *trusted third party* (TTP), public-key certificate is vouched by peer nodes, in a distributed manner, such as *pretty good privacy* (PGP). Obviously, certificate can not prove whether an entity is "good" or "bad", but the proven of ownership of key. Mainly it is for key authentication purpose.

Forth, key could be compromised or disposed after certain period of usage. Since key should no longer be useable after its disclosure, thus some mechanism is required to enforce this rule. In PKI, this can be done implicitly or explicitly. Certificate contains the *lifetime* of validity. It is not useful after expiry. But in some case, the private key could be disclosed during the valid period, in which case, CA needs to revoke a certificate explicitly and notify the network by posting it onto *certificate revocation list* (CRL) to prevent its usage.

B. 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. So certain trust framework must present to verify the key ownership. For PKI in the public key cryptosystem, there are two dominate trust models, namely, *centralized* trust model and the *web-of-trust* trust model. For network scalability

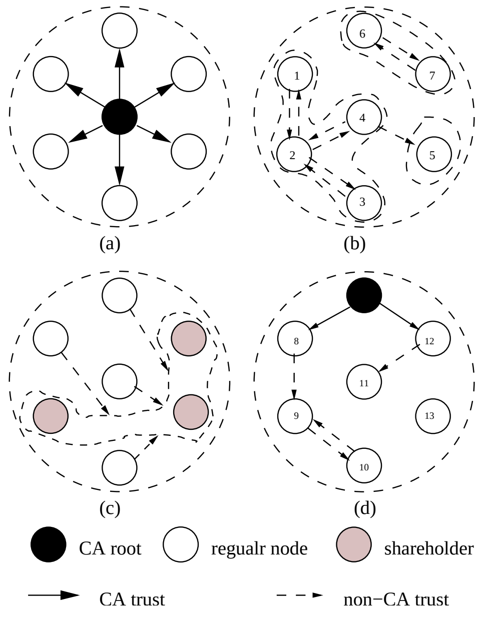


Fig. 1. Different trust models

the centralized trust mode could be in a hierarchical trust structure instead of a single CA entity. Multiple CA roots could be necessary for a large network, like Internet. There are two major variations proposed in ad hoc network, which we named *CA-view* trust model and *hybrid* trust model. The hybrid mode is to glue the centralized and the distributed trust system together[18]. See Figures 1 $(a)-(d)$ for different trust models.

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

Obviously, in mobile ad hoc networks, framework of key management which built on a fully centralized mode is not feasible not only because of the difficulty to maintain such a globally trusted entity but also the central entity could become a hot spot of attacking, thus network suffers from the security bottleneck. Meanwhile a completely distributed model may not be acceptable because of 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 *secret sharing* scheme. In SEKM, the system public key is distributed to whole network, while the system private key is split to all server nodes. For efficiency of secret share updating and certificate updating service, all servers form a multicast server group. The server group creates a view of CA in PKI for the mobile ad hoc networks.

C. *Secret sharing*

In mobile ad hoc network environment, a single CA node could be a security bottleneck if it is not well protected. Multiple replica 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, meanwhile it could be much easier for attackers to locate a target. So, an elegant secret sharing scheme is widely proposed in mobile ad hoc networks with different implementations. To better understand this scheme, a short brief is introduced here. The CA's private key K_{ca}^{-1}, which is 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. Some algorithms have been invented to enhance basic secret sharing scheme. For example, providing a way for shareholder to verify the validity of a share received, periodically shares updating, share recovery and partial certificate and so on. Some of those basic algorithms are shown below, which are referenced later in the SEKM key management scheme.

Secret splitting algorithm: [3] 1) System *dealer* pick up a random $k-1$ degree polynomial $f(x) = (\alpha_0 + \alpha_1 x + ... + \alpha_{k-1}x^{k-1})\ mod\ p$, or $f(x) = \sum_{i=1}^{k} \alpha_{i-1}(x)^{i-1}\ mod\ p$ in short, where p is a large prime number. The secret $K_{ca}^{-1} = \alpha_0\ mod\ p$. 2) Each secret share S_i is evaluated $S(i) = f(i) = \sum_{i=1}^{k} \alpha_{i-1}(x)^{i-1}\ mod\ p$. For simplicity, i is assumed to be integer.

Secret reconstruction algorithm: [3] 1) A *combiner* collect any set of k shares $R = \{(1, S_1), (2, S_2), ..., (k, S_k)\}$. 2) Function $f(x)$ is uniquely defined by set R. $f(x) = \sum_{i=1}^{k} S_i * L_i(x)$, where $L_i(x) = \frac{\prod_{j \neq i}(x-x_j)}{\prod_{j \neq i}(x_i - x_j)}$ is the *Lagrange basic function*. 3) Recover secret $K_{ca}^{-1} = f(0) = \sum_{j \in R \setminus \{i\}} S_i * l_i(0)$, where $l_i(0) = \prod_{j \in R \setminus \{i\}} \frac{j}{j-i}$ (assume $j = x_j$).

Share verification algorithm: [5] 1) The trusted *dealer* broadcasts $\{g^{K_{ca}^{-1}}\ mod\, p, g^{\alpha_1}\ mod\, p, g^{\alpha_2}\ mod\, p, ..., g^{\alpha_{k-1}}\ mod\, p\}$, which are commitments to system secret K_{ca}^{-1} and witnesses for coefficients $\{\alpha_1, ..., \alpha_{k-1}\}$ respectively. 2) Shareholder i with share S_i checking $g^{S_i}\ mod\, p = g^{K_{ca}^{-1}} * \prod_{i=1}^{k-1}(g^{\alpha_i})^i\ mod\, p$ hold or not.

The proofs of all algorithms are are shown in [3][4][5][6]. Some other secret sharing algorithms like share updating, and partial certificate are implemented in SEKM with proper modification and are given in Section 4 [7][13][14][15]. In summary, the above secret sharing algorithms make it a feasible technique to reduce trust and adapt to the distributed and unreliable environment of mobile ad hoc networks. This is also the main reason that in SEKM we adopt this technique in our key management scheme.

IV. SECURE AND EFFICIENT KEY MANAGEMENT (SEKM) SCHEME

A. *Notations and assumptions*

Some notations used in SEKM are listed 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 pre-configuration. The format of certificate is similar to X.509 structure with two extra attributes defined as *server flag* and *share version*. The flag of server is set to 1 and non-server is set to 0. The share version is set to 1 for server and 0 for non-server node. Version is increased by 1 after every share updating. Each server has its secret share stored in an encrypted format like in password-based or KEK scheme. Each server also has a copy of the encrypted share verification parameters $\{g^{K_{ca}^{-1}}\ mod\, p, g^{\alpha_1}\ mod\, p, g^{\alpha_2}\ mod\, p, ..., g^{\alpha_{k-1}}\ mod\, p\}$ stored.

Table 1 some notations.

t_s:	time stamp
n_s:	nounce, 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

Structure outline of certificate is:

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

B. *Overview of SEKM*

In 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 holding a system private key share and are capable to produce partial certificate. Quorum of $k(1 < k \leq m)$ servers can produce a valid certificate. It is quite straightforward to connect all servers and form a special group rather than to search each one of them separately and frequently. It is communication efficient, bandwidth saving, and easy for management. From a node point of view it is easy to locate the server "block" rather than each "point". From the server point of view it is easy to coordinate within the group

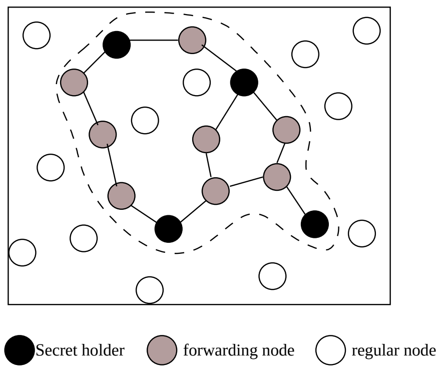


Fig. 2. Server group substructure snapshot in SEKM

rather than the entire network. We name this special group as multicast server group, or *server group* in short, though it is quite different from the traditional *source-receiver* multicast group. This server group is consisted of *server* nodes and *forwarding* nodes. The forwarding nodes within the group are regular nodes. Framework of SEKM consists of several phases namely server group formation phases; group maintenance phases; share updating phases; certificate renew/revocation phase; and handling new server nodes phase. Substructure of server group in essence creates a view of CA for certificate services and efficient share updating. For simplicity we state that server group produces the certificate without explicitly excluding the non-server forwarding nodes.

C. Secure server group formation and maintenance

In the server group formation phase, the SEKM scheme is similar to existing *ODMRP* (On-Demand Multicast Routing Protocol)[23]. ODMRP is an on-demand protocol, where a source-rooted or receiver-rooted forwarding group is formed based on periodically Join-Data and Join-Table messages and a mesh structure is maintained to forward multicast data rather than a tree structure. 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 to be members of the group in their lifetime, subset of non-server nodes could be forwarding nodes in certain period, thus become part of the group. The structure of the server group is a mesh with soft-state maintenance scheme. In general the mesh structure is more stable than the traditional tree structure. Although the tree-base structure is more communication efficient, it is easy to break in high dynamic situation, thereafter incurs excessive control traffic for link recovery. Maintaining the connection of 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 the network is a connected graph and one server group is maintained. In our future work we will consider multiple server groups scenario.

C.1 Group creation

The server group formation procedure consists of the 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. Only the server nodes can initiate the *JoinServeReq* packet, which is enforced by the server *flag* attribute in certificate. By doing this we can prevent malicious nodes from flooding the join request packet. *JoinServeReq* packet contains message m which includes $\{ID_i, SEQ_i, TTL\}$ together with its hashed signature $\{[h(ID_i, SEQ_i)]^{K_i^{-1}}|(TTL)^{K_i^{-1}}\}$, here symbol $|$ denotes concatenation. Node i could attach its certificate $\{Cert_i\}$ also 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 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 information contained in the message and route *backward learning*. Server certificate could also be stored in this table. Nodes that receive valid *JoinServerReq* will rebroadcast *JoinServerReq* packet if TTL is > 0. Compromised node could modify 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, *JoinServerReply* packet is also protected by replier's signature. The server could delay for a while before it sends out reply message so that a better path could be selected based on certain metrics, or enforce multiple pathes by sending reply message to multiple upstream neighbors.

When a node receives *JoinServerReply* it checks the validity of the packet first. After verification node could update its routing table based on *forwarding learning*. If the next hop field matches its own ID it will mark itself as forwarding nodes and forwarding the reply based on the routing table. Please note that the server node could be a forwarding node as well if it is on the shortest path between 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. Detail examples are described below.

C.2 Examples

In the example, nodes $1, 2, 16$ and 22 are servers. Figure 3 (a) shows the *JoinServeReq* initiated by server node 1 and gives a snapshot on dissemination of the request and reply message. When a node receives a request packet, it checks the validity of packet first before taking any further actions. It discards duplicate, non-authenticated, or illegally altered packet. In the example we assume the validity of all packets in processing are verified. After neighbor nodes 14, 18 and 20 receive the request they rebroadcast it. This process continues.

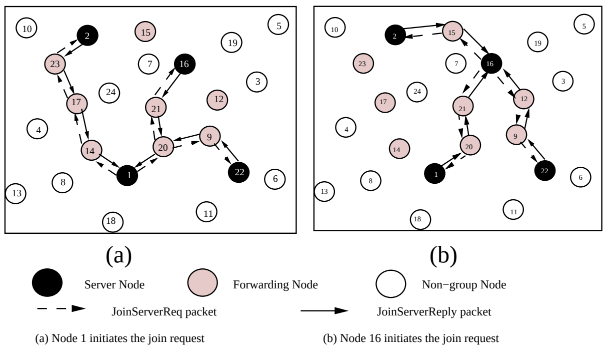
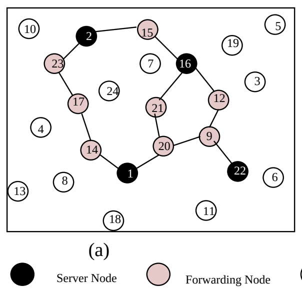


Fig. 3. Server group setup illustration



Server 16's table

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

(b)

Fig. 4. Server group mesh and table snapshot

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 receive the request from node 12. Server 16 could send reply to both node 21 and 12 in order to enforce multiple path. 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 serve node, and 2) it is on the selected path between server 1 and server 16, thus 3) it sets its forwarding *flag* and update its forwarding table. The same process happens on node 12 and node 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 if changed. After certain amount of time, replies from all servers arrive at node 1. Node 1 has the knowledge of all reachable servers. Figure 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. Figure 4(a) shows the server group mesh and Figure 4(b) shows a snapshot of table created by node 16. Every server node maintains a table and the table is referenced for subsequent certificate service phase and shares updating phase.

C.3 ***Group maintenance***

The server group structure should be maintained in 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 possible multiple pathes between pair of servers. Thus if one link is broken the alternative link could be utilized instead of launching the costly procedure for breakage recovery. 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 the dynamic network topology and the possible link breakage. Since this soft state scheme is quite expensive for a large network, the frequency for refreshing should be scheduled carefully according to node mobility.

D. ***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, it is still at risk in case the node is captured and compromised, thus the secret share is disposed. At meantime, once the *mobile attack* compromises enough number of shares the system secret is disclosed. In order to counter these types of attack, a periodically share updating scheme is proposed by some research paper with different implementations. In SEKM, updating the shares held by servers is quite simple. The idea is that only threshold k servers within server group initiated the share update phase. We name these servers as *active server* for convenience. Active servers generate new shares and send them to corresponding server in the group. Obviously, active servers consume more energy than non-active ones. In order to avoid some servers keep working as active servers, this phase is operated in rounds and teams of servers are "selected" as active servers alternatively. Similar to scheme in [16], 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:

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

where r is the current round, and SG is the set of server nodes have not ever initiated update phase 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 *active server*. Each round there are about k active servers. The algorithm in the share update phase is shown in next page.

E. ***Handling new servers***

In SEKM, it is flexible that new servers can join the network meanwhile 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 the possible *share inconsistency*. As we know that server group updates shares periodically. A new joined nodes could carry an outdated share from off-line configuration. In order to handle this situation the new node r need to contact at least k servers to "catch

Share Updating:

1) Each active server i randomly selects a $k-1$ degree polynomial $g_i(x) = (\beta_{i,1}x + \beta_{i,2}x^2 ... + \beta_{i,k-1}x^{k-1})\ mod\ p$, or $g_i(x) = \sum_{d=1}^{k-1} \beta_{(i,d)}(x)^d\ mod\ p$ 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 server group.
3) Each active server i computes share for server j, with $S_{i\to j} = g_i(j)\ mod\ p$, which is encrypted with $j's$ public key K_j, then sent to corresponding server $j(1 \le j \le k)$ in the form of $\{[S_{i\to j}]^{K_j}\}$.
4) Each server will receive about k new shares. It decrypts each new share, check validity and combine k news shares with its old share to 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.

up" with the latest server group with *renewed* share. As we describe above, a new node sends the *joinServerReq* message at first time of entering the network. The server group check 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 the following.

Handling new server:

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

F. *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 issuing. Each node (including server) 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 certificate based on the server group structure in SEKM. Once a server node receives a *CertUpdateReq* and verification of the request, in stead of sending the request to all server group nodes, it attaches a ticket and just send 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 as a counter. The ticket could be split at intermediate nodes. For a small server group broadcasting certificate request within the group is good enough. But for a large server group with $m >> k$, broadcast request to all servers cause significantly processing and bandwidth expenditure.

For example in Figure 4(a). Assume k is 3. When server node 1 receives a certificate updating request from a regular node or originated from itself, it could produce a partial certificate itself, meanwhile it sends two tickets attached to *CertUpdateReq* message to node 20 and no ticket to node 14. These two tickets would be split to two separate tickets with one ticket sending 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, 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 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 described as below,

Certificate updating:

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

G. *Handling certificate expiration and revocation*

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

A node's certificate could be revoked by server group within its valid period for certain reasons. A server node could be compromised, thus initiate inconsistent shares during the share updating/renewing phase, refuse to issue or issue wrong partial certificate for other nodes. A non-server node could misbehave in relaying the join request, join 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 initiator should be sent to the server group. Once server receives the accusation, it checks the validity of the packet first, if verified it marks certificate state of the accused node as *suspect*. There is a counter and timer associate with it. The counter could decrease after certain amount of time. Once the counter accumulates to a threshold value υ within the predefined time period ρ, collaboration of k servers can revoke the accused certificate. The revoked certificate is put onto the *CRL*. The *CRL* is broadcast to entire network periodically. Some information associate with the accuser must be stored in the server's database to prevent the abuse of accusation. A node with revoked certificate need off-line or in person reconfiguration before reenter the network.

*H. **Summary***

In summary, server group is formed securely and maintains connected. The certificate updating request is processed by 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 server with outdated share could be renewed. Node's misbehaving is monitored and could be accused by network nodes. A certificate can be revoked by server group. Nodes with expired or revoked certificate need off-line reconfiguration.

V. 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. Certain security mechanisms and protocols have been designed and proposed for ad hoc wireless network. Key management is a 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 networks 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 CA. The advantage is that in SEKM 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 whole 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 certificate update service for all nodes including the servers themselves. A ticket scheme is introduced for efficient certificate service. Meanwhile, an efficient server group periodical updating scheme is proposed. In our future work we will evaluate the performance of SEKM by simulation, and we will extend SEKM to multiple server groups in large networks and in partitioned networks.

References

[1] A. Salomaa. Public-Key Cryptography. *Springer*, 1996.
[2] A. S Tanenbaum. Computer Networks. *PH PTR*, chapter 8, 2003.
[3] A. Shamir. How to Share a Secret. *Communication of the ACM*, 1979.
[4] A. Herzberg, S. Jarecki, H. Krawczyk and M. Yung. Proactive secret sharing or: How to cope with Perpetual leakage. *CCrypto'95, proceedings*, Springer, 1995.
[5] P. Felman. A practical Scheme for Non-interactive Verifiable Secret. Sharing. *Proceedings of the 28th Annual Symposium on the Foundations of computer science.*, 1987.
[6] M. Stadler. Publicly Verifiable Secret Sharing. Switzerland.
[7] Zhou, L. and Z. Haas. Securing Ad Hoc Networks. IEEE Network Magazine, Vol. 13, 1999.
[8] S. Yi, P. Naldurg and R. Kravets Security-Aware Ad-hoc Routing for Wireless Networks. Report No.UIUCDCS-R-2002-2290 , UIUC, 2002.
[9] H. Luo and S. Lu. URSA: Ubiquitous and Robust Access Control for Mobile Ad-Hoc Networks. UCLA, 2004.
[10] W. Luo, and Y. Fang. A Survey of wireless Security in Mobile Ad Hoc Networks:Challenges and Available Solutions. Kluwer Academic Publishers pp-319-364, 2003.
[11] S. Burnett and S. Paine. RSA security's official Guide to Cryptography. RSA press, 2001.
[12] I. momhamod wireless networks. CRC press, 2003.
[13] Y. Desmedt and S. Jajodiay Redistribution secret shares to a new access structures and its application. University of Wisconsin-milwaukee, 1997.
[14] T. M. Wong, C. Wang, and J. M. Wing. Verifiable Secret Redistribution for Threshold Sharing Schemes. Carnegie Mellon University, 2002.
[15] V. Shoup. Practical Threshold Signatures. Eurocrypt 2000.
[16] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocol forWireless Microsensor Networks. Hawaii International Coference on System ScienceMaui, 2000.
[17] S. Capkun, L. Buttyan, and J. Hubaux. Self-Organized Publick-key Management for Mobile Ad Hoc Networks. IEEE Transactions on Mobile ComputingVol.2 NO.1 2003.
[18] S. Yi and R. Kravets. Composite Key Management for Ad Hoc Networks. First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04).
[19] M. G. Zapata. Secure Ad Hoc On-Demand Distance Vector (SAODV). Internet draft,draft-guerrero-manet-saodv-01.txt ,Aug., 2002.
[20] Y. Hu, D. B. Johnson and A. Perrig. SEAD: Secure Efficient Distance Vector Routing in Mobile Wireless Ad-Hoc Networks Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '02), 2002.
[21] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. The TESLA Broadcast Authentication Protocol. Internet Draft, Internet Engineering Task Force, July 2000.
[22] P. Papadimitratos and Z.J. Haas. Secure Routing for Mobile Ad Hoc Networks. Proceedings of the SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002),2002.
[23] S. Lee and W. Su and M. Gerla. On-Demand Multicast Routing Protocol in Multihop Wireless Mobile Networks. ACM/Baltzer Mobile Networks and Applications, special issue on Multipoint Communication in Wireless Mobile Networks, 2000.