Locality Driven Key Management Architecture for Mobile Ad-hoc Networks *

Gang Xu and Liviu Iftode
Department of Computer Science, Rutgers University
{*gxu,iftode*}*@cs.rutgers.edu*

Abstract

A fundamental issue of securing mobile ad-hoc networks is to ensure mobile nodes can authenticate each other. Because of its strength and efficiency, public key and digital signature is an ideal building block to construct the authentication service. Although this is already mature in the Internet applications, providing public key based authentication is still very challenging in mobile ad-hoc networks because the entire network is world-accessible via wireless channel, the environment is highly volatile and infrastructureless and there is lack of trust in the system. In this paper, we propose a locality driven key management architecture that achieves robust key authentication and facilitates timely and efficient establishment of distributed trust. The architecture reflects our application oriented view of MANET and is based on threshold cryptography to achieve high fault tolerance against network partition and malicious nodes. On top of it, we design distributed trust protocols to help set up trust relations on-the-fly. To verify the concept and the design, we implement the prototype and simulate our solution in a variety of scenarios.

1 Introduction

In the past few years, more and more attention has been drawn to the security of mobile ad hoc network (MANET) [12, 17]. Due to its glorious success in securing Internet computing, the Internet de facto standard public key cryptography including encryption and digital signature becomes the natural choice as a fundamental building block to secure MANET. However, since MANET is significantly different from the Internet, a salient issue is how to adapt the technology to the new environment.

As we know, successful application of public key cryptography relies on the ubiquitous capability of verifying the binding between a public key and the owner principal. In the Internet, the mainstream solution is to have a third-party centrally trusted entity, called Certificate Authority (CA), vouch for the authenticity of the binding by issuing digital certificates, which in essence is a statement of the binding digitally signed by the CA. In practice, CAs and digital certificates are organized and maintained by Public Key Infrastructure (PKI) [8, 1].

It is questionable yet if PKI can be implemented in MANET because PKI requires well-protected CAs and constant connectivity between users and CAs. However, MANET is composed of a group of mobile devices communicating with each other through wireless link without a backbone or infrastructure. In such an environment, all devices are exposed to hacking to the same extent and no one can be assumed to be significantly more secure than the others. Moreover, devices roam around, run out of power or just stop functioning, which lead to volatile connectivity among them and CAs. Research proposals have been seen in [20, 19, 13] etc to address the two issues by distributing the CA's functionality across a set of network nodes and use threshold signature [18] to achieve tolerance up to the threshold number of faulty nodes. These methods are suitable for small MANET with a single CA. This is partially due to the inherent high communication cost. A more fundamental reason is that it is difficult if not infeasible for a CA to get familiar with all the other principals in large MANET while the trustworthiness of certificates a CA signs mainly depends on how much it knows about the principals.

Some researchers take another approach based on the concept of "web of trust" first appearing in PGP [22]. In these methods each principal is its own CA [10, 6] and keeps a certificate directory. To authenticate a certificate signed by another principal, a principal has to find a certificate path between them. Although these methods avoid the problem of maintaining a key infrastructure at high cost, they are faced with difficulty of finding such a path without incurring a lot of broadcasting costs or forcing each principal to save a large number of certificates in its local directory.

We believe to solve the key management issue in MANET we first must have a different view of it. Opposite to the traditional routing driven view of MANET as a monolithic network, we envision it from application angle

*This work has been supported in part by the NSF ITR grant ANI-0121416

0-7803-8815-1/04/$20.00 ©2004IEEE

as a group of interacting networks. This is based on our observation that MANET is more task or application oriented in that a MANET is usually formed to fulfill a task. Different MANETs may need to talk to each other to get help for another task. For instance, in a highway, a few cars going to a common destination establish a MANET to share directions and they may talk to other cars (in other MANET) to get information about traffic situation [11]. The application view discloses the locality of MANET. And we argue locality helps build key management because each MANET now are composed of principals for the same task, which should have close interaction with and distance between each other, and this further makes it possible for trust to be established dynamically.

We propose a new key management architecture driven by the application oriented view of MANET and the concept of locality of trust. In the architecture, certificate authorities are established only within a neighborhood using threshold cryptography and different certificate authorities maintain trust relationships, called trust chains for cross-CA authentication. Compared to the previous methods, our architecture shows a number of advantages which make it suitable for MANET. First and foremost, locality makes certificates more trustworthy in that in a local community a CA has better chance to interact with other principals. Moreover, the communication overhead between principals with their CA is reduced because of shorter local distance of message delivery. Thirdly, as with the other key management systems using threshold cryptography, the architecture inherits fault-tolerance and high availability directly. Finally, it provides support for authenticating foreign certificates in a low-cost and timely fashion by letting each CA maintain a global trust table to maintain trust chains among CAs without forcing each individual principal to keep a huge certificate directory like in other PGP-like approaches [22]. Compared to other systems like [10], the solution provides an answer with full certainty (not just a probability one) but at lower communication cost by eliminating the need of broadcasting trust changes to even irrelevant principals every time.

The rest of the paper is organized as follows: Sect 2 discusses the related works. The architecture is introduced conceptually in Sect 3 followed by detailed descriptions in Sect 4. Then the correctness and costs of the our solution are evaluated in Sect 5. Finally, we conclude the paper in Sect 7.

2 Related Works

[20] is one of the first efforts addressing the key management issue in MANET. The authors proposed a conceptual model of distributed public key infrastructure, where a group of servers collectively act as a certificate authority. To achieve this, the service private(signing) key is broken into pieces, each of which is kept by one server, and threshold cryptography [18] is used to ensure that certificate services will not be subverted unless over a quorum of servers are not available and incorrect. The solution improves both availability and security of the certificate authority in that the system can continue to function as long as at least threshold number of servers are still available and functional. On the other hand, when the network consists of a large number of nodes, it may incur significant communication and computation overheads since every request needs to be distributed to and handled by all participating servers. The idea was followed up and implemented in COCA [21] whereas the implementation is targeted for infrastructure-based networks such as the Internet.

MOCA [19] follows the same direction by building a distributed certificate service with the help of threshold cryptography. It improves security by discriminatively picking more secure nodes as CA candidates and reduces communication overhead by caching routes to MOCAs and using unicast instead of flooding when sufficient cached routes exist. As with COCA, MOCA inherits high communication costs from threshold cryptography. Caching alleviates the problem to some extent when the network stays static such that the cached routes are valid for a relatively long period. However, in more volatile MANETs topology of which is changing rapidly, this optimization will be insufficient.

[13] takes a step further by letting every node hold a share of the certificate authority secret key and any K nodes be able to recover the key. The security depends on the system-wide parameter K. Since now each node is a certificate server and compromising any K of them will disclose the private signing key, it actually endangers security to have a small K relative to the total number of nodes. However if K is too big, it degrades to the basic form of threshold CA as [20].

Pathak et al proposed in [15] a voting based scheme for both public key authentication and group membership control. In this method, the decision of trust is made collectively by a group of n principals via voting. The system achieves high fault tolerance when it satisfies Byzantine condition. Compared to the above threshold based CA solutions, the method does not require a shared trusted principal (the dealer) and therefore does not have any single point of failure. However, the group does not own a single signing key. Consequently each individual principal has to know all the public keys of the n voters and perform n signature verifications to authenticate one public key.

Another direction to the key management in MANET is to extend the "web of trust" concept of PGP [22]. The basic idea is to make each node be its own certificate authority. [10] is one of them. As a self-organized key management system, it eliminates the need to have central certificate directories in PGP for certificate distribution. Instead,

each node picks and maintains a set of certificates according to some special selection algorithms. To authenticate a public key, the node *u* merges its own directory with the directory of the key owner *v* and tries to find a path from *u* to *v*. The selection algorithms guarantee high probability of finding such a path with relatively small-sized local directory.

[6] proposed a group based method, where nodes are organized into groups identified by the group public key. Each group has a leader who owns the group private key and is responsible for certifying memberships by creating certificates to member nodes. Key authentication is at group level in a way similar to other PGP-like protocols but not fault-tolerant in that a single compromised team leader can subvert the membership authentication. Since the method is targeted for scalable authorization, it does not support individual node authentication.

[5] and [7] solve the issue in certain special cases. In [5], a strong key can be derived from some prior context, a shared weak secret among all nodes. In [7], a secure side channel is assumed to exist to help initial key exchange between two nodes.

Our approach is inspired and influenced by the works above. What distinguishes it from them is our unique application driven view of MANETs, which makes it possible to construct a large scaled MANET from smaller local ones. Compared to works in [21], [19] and [13] which also use threshold signature to generate certificates, our approach ties a CA only to the local community it resides in. Consequently, the CA has better chance to interact with other principals within the same community and tell the authenticity of their public keys. To deal with cross CA authentication we build trust chains among communities, which shares the same idea as [22], [6] and [10]. However, in our approach only CA servers need to get involved to build the chains, which greatly reduces the size of certificate directory. Furthermore, trust chains are established using the same threshold method as in local CAs and thereby is not only much more difficult to subvert but able to provide accurate answers to certificate verification requests.

3 Key Management Architecture

Our key management architecture is driven by the application oriented view of MANET and the concept of locality of trust. Actually the concept of locality has already been exploited explicitly or implicitly to help build security. For instance in [7] two network nodes set up a secure channel via location-limited channels. [13] also enhances the trustworthiness of a certificate using locality in that a certificate is generated by one-hop neighbors in most cases. We adopt a different approach. Our architecture is composed of a group of Certificate authorities (CAs), each of which provides public key authentication service to its own community. Among them, there may exist trust relationships. CAs rely on the trust relationships with others to authenticate "foreign" certificates issued by other CAs. The architecture is illustrated in Figure 1. We will explain in details how CAs and the trust relationships among them are established and maintained at conceptual level in this section and discuss the protocols in details in the next section.

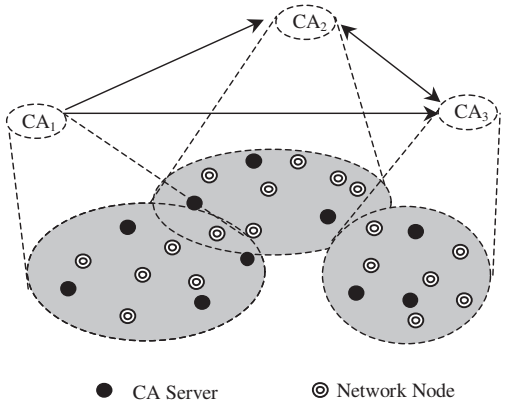

Figure 1. Conceptual Architecture

3.1 Certificate Authority

Adjacent nodes jointly establish and maintain a certificate authority for key authentication. We say two nodes are adjacent if they have short routes to each other. We call these nodes CA servers to differentiate them from the other nodes in the neighborhood. The CA is constructed using standard threshold cryptography. To be self-contained we include the high level description of the threshold mechanism. More information about threshold digital signature can be found in [18, 14].

1. *CA Bootstrap.* We assume each node has the capability of generating public/private key pair, producing and verifying digital signature. And we further assume there exists a special node d in each neighborhood, which is trusted by all nodes. In threshold cryptography, d is also called dealer and its own public key PK_d is well-known to all nodes. The CA is initialized by the dealer d generating a CA key public private key pair denoted as (PK, SK) and polling the community for N nodes who would like to serve as the servers constituting the CA. The criteria that d picks up the N nodes are system dependent and are beyond the scope of our discussion. i.e. d may select servers with most computing resources or better network connectivities etc. Assuming a commonly trusted dealer can be risky in general since this introduces a single point of failure. However, this can be mitigated from two aspects. First, it is not

unusual that such a dealer does exist, i.e. the chair of a workshop or the provider of some MANET services. Second, the trust is only needed at the bootstrap phase and can be revoked after CAs are fully setup. More importantly, bootstrap can also be accomplished without the help of the dealer as discussed in [14]. However to favor performance and simplicity it is not discussed here.

When N servers are chosen, the d generates the partial shares of SK: $SK_1, SK_2...SK_N$ for each server all signed with d's private key SK_d with the threshold set to K. That is, any set of K servers can generate a valid CA signature by combining partial signature from each of them but any fewer than K servers can't. We require $K > \frac{S}{2}$ such that honest servers are the majority. Due to the locality nature of CA, the CA servers should have good connectivity to each other. We assume at any time at least K honest servers are available and for any message delivered at least K honest servers receive it. Then d distributes the partial shares to the other servers, destroys the CA private key SK and broadcasts the composition of the CA to the community.

2. *CA Certificate Generation.* The CA provides both certificate generation and verification services to the nodes in the community. Any node x in the community can request a certificate from the CA, which put in the simplest way, is the public key of the node denoted as PK_x signed with SK of the CA. The procedure starts with x submitting its PK_x to one of the N servers, for instance, S_p. Since S_p acts as the proxy of x we call it *proxy server* in this paper. Then S_p calls other servers to sign PK_x using their partial keys. Each server independently decides if PK_x should be trusted as the public key of x. If not, it signs a predefined denial message instead of PK_x. The partial signatures are gathered by S_p for full signature generation. As long as K servers concur to trust the PK_x, the valid certificate is generated. On the other hand, verification of a certificate is straightforward since the signature is literally the same as that signed with SK and thus can be directly verified using CA's public key PK.

   Our approach does not impose any restriction on how a CA server evaluates the trustworthiness of a principal's public key. Instead, the trustworthiness of a public key is fully determined by policies and rules, which can vary from CA to CA or even server to server within the same CA. CA servers are free to choose any policies at their own discretion. For instance, a CA server may be convinced of the authenticity of a node's public key just because the network address in the certificate request matches the source address of the request sender while a more prudent server may do more by checking if there are any conflicting requests for the same address. However, whatever policy CA servers choose, our architecture guarantees that no single or even up to $K-1$ malicious servers can subvert the authentication.

3. *CA Certificate Revocation.* Basically, the revocation process is similar to standard PKI with an exception that signature is generated in a threshold manner. The CA keeps a copy of all the revocations it has signed in its certificate revocation list (CRL). When a node y later wants to verify the validity of a certificate, it may send the request to the CA. Similar to processing certificate generation request, each CA server check its local CRL. If y is found in its CRL, it will partially sign a predefined denial message. Finally combining the partial signatures the CA can generate an signed reply in line with the decisions of at least K servers. When the CA revokes a certificate, it may optionally notify new revocations to the community. Since the community is local and the revocation is relatively rare, the communication cost is not prohibitive.

3.2 Trust Chain

So far, CAs manages key authentication for nodes in their local neighborhoods. When two nodes from different neighborhoods need to authenticate each other, they need to have a trust chain. In this section, we discuss the details of how a trust chain is established and maintained and how certificate can be authenticated using it.

3.2.1 Trust Chain Definition

In the context of key management, we have a narrow definition of trust between two principals: A principal A trusts another principal B if and only if (1) A can authenticate B and (2) A believes in the authenticity of any valid certificate signed by B. Opposite to this are the relation distrust and unfamiliar. We say A distrusts B if only if A can authenticate B BUT does not believe in the authenticity of any valid certificate signed by B. Unfamiliar covers all the rest, which means A has no idea about B, either its identity or its behavior. We assume trust is transitive while both distrust and unfamiliar are not. If A trusts B on its own, i.e. based on direct interactions with B, we call this direct trust, denoted as $\rightarrow$ to differentiate it from the trust gained by transitivity. Opposite to direct trust is indirect or recommendation trust. We say A indirectly trusts B if there exists a trust chain from principal P_0 to P_n, denoted as $P_0 \Rightarrow P_n$, defined recursively as follows:

(1) $P_{n-1} \rightarrow P_n$ and
(2) $P_0 \Rightarrow P_{n-1}$

Neither $\rightarrow$ nor $\Rightarrow$ is symmetric, i.e $A \rightarrow B$ does not imply $B \rightarrow A$. Formally, the trust chain can be represented as a directed graph $G = (V, E)$ where V is the set of vertices representing all CAs and E the set of edges representing the relations between two CAs. Based on the type of the relation, we call it trust, distrust or unfamiliar edge. Let V_1 be a CA C_1 and V_2 be another CA C_2, the trust chain from C_1 to C_2 can be represented as a directed path from V_1 to V_2 which is composed of only trust edges. If we take a CA as a root, all the chains it has compose a trust tree.

3.2.2 Dynamic Trust Chain Maintenance

When the system is bootstrapped, only direct trust exists. Each CA needs to learn from other CAs it directly trusts on to build trust chains. Furthermore, when changes of trust take place somewhere, a CA must be able to update its chains to reflect the new trust relationships on-the-fly. We define a protocol enabling each CA to build locally the trust tree. The idea is inspired by routing protocols such as OSPF [2] in which each router only keeps local information but computes global routing table by exchanging information only with its neighbors. Similarly, each CA keeps record of its directly trusted CAs, and establishes indirect trust on other CAs by exchanging records with them. The details are below:

Data Structures Each vertex V_i maintains two tables: I_i and relation table R_i where

$I_i = \{V_j | V_j \rightarrow V_i\}$ and
$R_i = \{(V_j, V_g, c) | V_j, V_g \in V$ and c is an integer$\}$

In I_i table, each CA keeps track of the truster CAs which trust itself. When a CA becomes trusted by another CA, i.e. receiving a certificate from the CA, it updates its I_i table to include the certificate issuer. The table R_i describes the trust relations a CA has with other CAs. Conceptually,V_g serves the role similar to gateways in routing: it means the reason the relation between V_i and V_j exists is due to the fact V_g has the relation with V_j, i.e., $V_g \Rightarrow V_j$, and $V_i \rightarrow V_g$. c is the weight of the relation and $-MAX \leq c \leq MAX$, where MAX is a positive integer. By assigning c to different values, R table can capture all kinds of relations the CA has with others, for instance, positive for trust, negative for distrust and 0 for unfamiliar. The bigger $|c|$, the more trustworthy the relation.

Trust Propagation Whenever there is a change of inter-CA trust relations including both establishment of new direct trust and revocation of existing trust, all related CAs need to be notified. In details, for a CA C_i represented by vertex V_i,

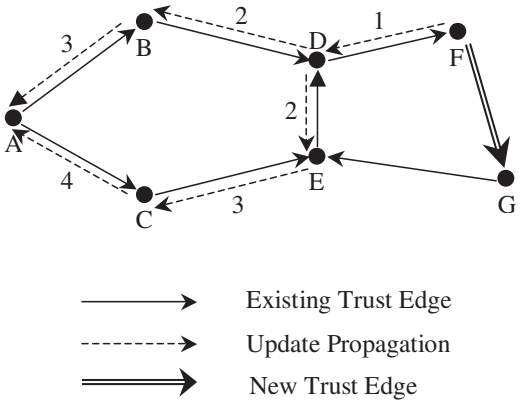

Figure 2. Trust Chain Update

1. *Relation Establishment and Revocation*
   This happens when C_i decides to trust C_j. V_i assigns an initial weight $c(c > 0)$ to the relation and adds the entry (V_j, V_i, c) to R_i. Then it sends an notification message [ESTABLISH:(V_j, V_i, c)] to V_j and waits for replies. When V_j receives the message, it replies with its own current trust table R_j signed with the CA key. When V_i receives the reply, it checks its R_i to find out if it needs to updated because of R_j. The R_i table will be updated if and only if there exist l,m and n such that for (V_l, V_m, c_i) in R_i and (V_l, V_n, c_j) in R_j, and the following holds:

   $$|c_i| < (|c_j| - w)$$

   $w(w \geq 0)$ is the loss of trust transitivity. That is, the trustworthiness of a relation decreases as it becomes more and more indirect. When $c_i = 0$, the condition represents changes of relation from unfamiliar to familiar(trust or distrust). Otherwise, it reflects the fact that there are different levels of trust and a CA always wants to establish the more trusted relation. In both cases, (V_l, V_m, c_i) will be replaced with $(V_l, V_j, -(|c_j| - w))$ if $c_j < 0$ or $(V_l, V_j, c_j - w)$ otherwise, which results in a new table R_i'. Then, the CA sends an update message [UPDATE:R_i'] to each $V_i \in I_i$. Similarly when C_i decides to distrust C_j, it adds an entry (V_j, V_i, c) to R_i and directly sends an update to each $V_i \in I_i$. Sending notification to V_j is an option but not required since distrusting V_j implies no expectation on V_j to follow the protocol any more.

2. *Relation Update*
   The scenario is triggered by the message

[UPDATE:R_l] from V_l. Upon receiving the message, a CA C_k will first check if this message is an out-dated one. This is done via assigning version number to table R and we will explain in details in next section. If not, it just repeats the same procedure as the above to update its own R_k. Finally if a change is made and a new table R'_k is generated, C_k passes [UPDATE:R'_k] to each V_k in its I_k set.

An example is shown in Figure 2 depicting how a trust chain is maintained by updating each CA's local trust table in a lockstep manner. In this example, A, B, C, D, E, F and G are CAs. When F begins to trust G, it triggers chain update along the reverse path of the trust chain step by step. For instance, D gets the update message first and then propagates to both B and E, which further notify A and C.

A CA verifies if the message comes from a directly trusted CA before taking further actions at each step by running public key authentication. The only exception is when a CA receives ESTABLISH message. This is because for one thing the receiver does not need to trust the sender (recall that trust is asymmetric) and for the other thing it may not have the sender's public key.

3.2.3 Cross CA Authentication

Trust chain provides a mechanism for cross CA authentication. When x wants to verify if PK_y belongs to another node y, it verifies the validity of y's certificate. If y is signed by a CA x trusts, e.g. CA_i, the key authentication is successful. However, if y just roams from another neighborhood and only has one certificate signed by CA_j, x may still be able to authenticate y if there is a trust chain from CA_i to CA_j. When CA_i finds an entry like $(j, l, w)w > 0$ in its R_i, it can either reply with a message of "successful authentication" or continue sending verification request to C_l, which recursively repeats the same procedure until a response signed by C_j is returned. The former case trades off assurance for low cost in that the result is based on previous records and does not take into account most recent revocation. By contrast, in the latter case, the result is directly based on the certificate issuer's latest decision but incurs more communication and computing costs. Another way to enhance assurance is to let each CA also notify all CAs in its I table when a CA revokes a certificate and each CA receiving the revocation simply saves it to its local CRL and continue forwarding the message in the same way. As a result, each CA has the CRL of all other CAs it trusts and can answer the verification request on behalf of them.

On the other hand, if y comes into a new community and wants to authenticates node z, it may choose to either phone home or trust the new local CA and rely on it to provide certificate services. The former option stops working when y loses network connectivity to its original community, which takes place more often than not in MANET. The latter solution does not have this problem but requires new trust to be established, which entirely relies on y's own judgment. A more tricky issue is when y is a CA server of its original community. Since the CA uses threshold signature, losing one server does not prevent a CA from functioning. However, if more than $N - K$ servers leave the community, the CA is broken. We have not addressed this problem in the current approach. To tackle this problem, we plan to require each CA server to find an inheritor before leaving and securely transfer the partial secret it keeps to the inheritor. Then the inheritor notifies other CA servers and generates a new CA certificate describing the new composition.

Trust chain is only used for key authentication in this paper due to our narrow definition of trust. But its use is not limited to that. By adding more semantics to the trust definition, trust chain makes it possible to establish distributed trust. For instance, when trust is based on nodes' behavior, the CAs actually become monitors and any node can verify if another node is decent if there is a trust chain between their monitors.

4 Protocols

It is straightforward to implement the trust chain protocol defined in Sect 3 when each vertex is a single network node. However, in our trust chain a vertex represents a local CA, which actually consists of a group of CA servers. Communication protocols are needed to ensure that a group of servers can behave like a single server. Typical solution to this problem in fixed wired networks is picking up a server as a master and let it represent the team. However this will not work in our system because no one can be trusted absolutely. Furthermore the improvement of security in CA lies in its threshold fashion such that no single (or in more general less than quorum) server can compromise the whole system. This only makes sense if each server makes the decision of trusting or distrusting other CA's on its own but master-slave kind of solution invalidates the assumption. To solve this problem, we define two protocols: CA Table Update protocol (CTU) and CA Head Election protocol (CHE), both of which work in an autonomous fashion. That is, each of the servers keeps a copy of all the data including R, and I tables and makes trust decisions independently based on its local data signed with its partial key, while combining these decisions together leads to a uniform decision bearing the CA signature.

In the rest of the section, we will explain in details how the two protocols work. Assume we have a CA $C_i = \{C_{i1}, C_{i2}, ...C_{iN}\}$ with N servers ,K as the threshold, and the local copies of R_i, and I_i of each CA server $C_{ik}(1 \leq k \leq N)$ are denoted as R_{ik}, and I_{ik}, respectively. Especially, A server C_{ik} will not update its local copy of R_{ik} un-

less it receives a new table $R_i{}'$ signed by its CA. This guarantees that all updates to R_i table are certified by at least K CA servers.

4.1 CA Table Update Protocol

Based on the conceptual protocol above, a CA updates its table when it establishes a new trust relation with another CA or just receives an UPDATE message from a CA it directly trusts. Let's discuss them one by one. To deal with replay attack or out-dated data, we let each C_{ik} of C_i keeps a version number of its local copy of R_{ik} denoted as v_{ik}.

1. *Establish a new trust relation*

   (a) *When the proxy CA server, denoted as C_{ip}, generates a new signed trust decision, it makes a copy of its R_{ip} and work on the copy to generate a new table R'_{ip} according to the new trust. Then it multicasts to all other CA servers a table update request [UPDATE:R'_{ip}] with version $v'_{ip} = v_{ip} + 1$ and waits for replies.*

   (b) *Upon receiving the new trust decision, a server C_{ik} verifies if it is signed with the CA key and should be trusted. Then it does the same as C_{ip} to create a new table R'_{ik} and compares R'_{ik} including the version with R'_{ip}. If they are matched, C_{ik} signs the new table with its partial key and returns it to C_{ip}.*

   (c) *When C_{ip} has at least K correct and unanimous responses, R'_i with the version $= v'_i$ it generates the full signature for it. Then it overwrites its own copy of the table R_{ip} with R'_i and multicasts the signed new table to other CA servers. If its I_{ip} is not empty, it also sends the newly signed table R'_i to all CA's in I_{ip}*

   (d) *When C_{ik} receives the new table, it verifies the signature and overwrites its own R_{ik} with the new one.*

2. *Respond to UPDATE message*
   The protocol works in the same way as the previous one except the first two steps:

   (a) *When a proxy C_{ip} receives an update message [UPDATE:R_l] from another directly trusted CA l, it multicasts the message to other CA servers followed by a table update request [UPDATE:R'_{ip}] as well as [UPDATE:R_l].*

   (b) *When a server C_{ik} receives the update messages it verifies if it trusts l directly. If yes, it checks R_l against its local R_{ik} in the same way described in the conceptual protocol above. If it needs to update its R_{ik}, it generates a new table R'_{ik}, compares it with R'_{ip} and signs it if matched in the same way as above.*

In both cases, the message signed with the CA key also serves as a confirmation: a server will not generate more partial signature if it is still waiting for another confirmation. Since in MANET there is no guarantee of message delivery, the server will time out if no confirmation is received. The protocol works even if there are faulty or compromised servers as long as they are less than $N - K$. To get a valid signature, a proxy server, no matter it is honest or dishonest, needs to collect K correctly partially signed and unanimous R. Since we assume at least there are K honest servers which cache the correct R table locally, the protocol guarantees that new table is generated correctly. On the other hand, there are at most $(N - K) < \frac{N}{2} < K$ servers which can have incorrect R or be compromised, it is impossible for a malicious user to generate a wrong signed table R.

4.2 CA Head Election Protocol

When dealing with multiple update requests at the same time, the CA Table Update (CTU) protocol may generate different table R_i with the same version even though the protocol requests each server to generate one partial signature a time. This happens only when compromised CA servers cooperate to violate the protocol. For instance, if both C_{ip1} receives [UPDATE:R_l] and C_{ip2} receives [UPDATE:R_m] at the same time and both send the table update request to other CA servers, there is no way to control the sequence of these requests being received. It is possible that $\frac{K}{2}$ honest servers get [UPDATE:R_l] and the other half get [UPDATE:R_m]. In general the two different UPDATEs result in different R'_i but if all servers only sign one table a time, the two different tables will be signed in sequence and thus with different version number. However, compromised servers can simply sign both requests such that both C_{ip1} and C_{ip2} now may have enough partial signatures to construct their R'_i with the same version number. This will cause confusion to other CA's when they need to update their trust table since version number is used to make sure only the latest table from the trusted CA's will be considered. To solve the problem we have to restrict that only one request is allowed to be sent at a time. To simplify the protocol, we require in one CA only one server, called Head, is permitted to send request. To avoid trusting any single server naively, the server is elected by the group and rotated periodically and/or on demand. There is no means of preventing a Head from vio-

lating the protocol. However, this can be easily detected by honest servers which receive multiple requests at the same time. The protocol is simple and works as follows:

1. *Each server is assigned an ID known to all other servers. Everyone in the CA keeps track of the current head ID, called HeadID and uses the same algorithm, called GetNextHead to get the ID of the new Head. It is easy to find such an algorithm, i.e, if all ID's are within [0..$N-1$], $(HeadID+1) mod N$ works. When some server C_{ip} calls an election, it multicasts the request to all other servers.*

2. *When a server C_{ik} receives the request, it calls GetNextHead to generate the ID and compares it with the request. It signs the request for the new Head with its partial key if they are matched and sends it back to C_{ik}.*

3. *Similar to table update, C_{ip} generates the fully signed notice for the new head, multicasts it to all others and they will update their head accordingly.*

There is no need to restrict a CA to only generate one partial signature for the new head at a time in this protocol, because even if multiple requests arrive at the same time, a CA server will only signs the one which matches its only calculation, which remains the same since *GetNextHead* always returns the same value when current $HeadID$ is unchanged.

4.3 Message Delivery Fault Tolerance

Messages can get lost at network layer, i.e. routing failure and application layer, i.e. a CA server dropping requests. The former problem has been extensively studied in MANET routing algorithms [17, 16] and is beyond the scope of this paper. We simply assume for each message sent at least K honest servers in a CA will receive it. We would like to tackle the latter case. As we know, a single compromised server can not forge a valid signature and the only harm it may do is Denial-of-Service(DoS) attack by dropping it silently. However, this is detectable if it takes place inside a CA where for each request a reply is expected by the requester and dropping a request will finally result in discovery of the deviation from the protocol. The only threat is to inter-CA messages including ESTABLISH and UPDATE. In this case, both sender and receiver can be compromised.

There are two options for this. First, since lost of messages only results in other CA's having out-of-date trust relations, we let each CA run a mandatory periodic update notifications to everyone in its I table by picking up both sender and the receiver randomly. In this solution, the risk is bounded and finally an update notification can be received by an honest server in other CA's. Second, since out of $N-K+1$ servers there is at least one honest server, we can guarantee success of message delivery if there are at least $N-K+1$ senders and receivers. To do this, each CA picks up $N-K+1$ servers as senders and every time when a CA sends message to another, let each server in the sender CA to determine if it is one of the $N-K+1$. If it is, it sends the message to the receiver CA by picking up randomly $N-K+1$ servers of the CA. To make it more efficient, each server is assigned a priority and delay a different time before sending the message using a delay timer. If the server receives a confirmation before the timer expires, it simply cancels the timer. For instance, when sending ESTABLISH message to another CA, a server set up the timer and when it receives UPDATE message from the CA, it cancels the timer. We take both approaches in our solution without implementing the timer.

5 Evaluation

We evaluate our locality driven key management architecture in a hybrid method of both prototype implementation and simulation. First we implement a prototype based on openssl [3] libcrypto library on Linux platform in C to evaluate the real computation cost of the threshold scheme. Second, we use NS-2 [4] to simulate the protocols to evaluate its effectiveness and communication overhead. The limitations and potential optimizations of our implementation are also discussed.

5.1 Prototype Implementation

The prototype implements the basic signature generation function of CA's. It consists of 3000 lines of C code and is compiled on Linux 2.4 for both Pentium (Dell Latitude CPi Laptop with Pentium II 366 MHz processor and 256M SDRAM) and ARM(HP/Compaq iPAQ H3700 series with 206 MHz Intel StrongARM SA-1110 32-bit RISC processor and 64M SDRAM) architectures. Computation costs in terms of CPU time in different scenarios are evaluated on both architectures and communication delay is calculated in the 802.11b network composed of several iPAQs. The results are shown in Table 1 and 2. The calculation is based on RSA 1024 bit public/private key.

From table 1 and 2, the overall cost is higher on the low-end PDAs than on more powerful and resourceful full-fledged computers. The computation cost from the CA server's perspective is independent of how the CA is composed because the computation load remains the same. Furthermore, the impact of increasing K on the proxy is nearly indiscernible. As observed by J. Kong et al. in [13], this is because the overhead of combining more partial signatures,

N	K	Proxy (ms)	CA Server (ms)
3	2	13.95	25.81
5	3	13.96	28.25
5	4	15.89	13.85

Table 1. *Costs of Computation on Dell Latitude CPi Laptop(Pentium II 366M Hz, 256M SDRAM)*

N	K	Proxy (ms)	CA Server (ms)	Total Delay (ms)
3	2	396.70	333.98	717.87
5	3	421.85	203.29	959.88
5	4	427.60	212.93	996.25

Table 2. *Costs of Computation on HP/Compaq iPAQ H3700 (Intel StrongARM 206M Hz, 64M SDRAM)*

when compared to that of key generation, can be ignored. However, the total delay becomes bigger with the number of servers in the CA and the threshold increasing. This is actually due to the increased network communication costs and the waiting time to collect all partial signatures.

5.2 Simulation

The simulation targets for non-computational perspectives of our architecture including communication cost and effectiveness in different scenarios of MANET. It leverages NS-2 simulator and mainly consists of 1500 lines of C++ code and a few hundred lines of TCL code.

5.2.1 Measurements

We evaluates the localization driven key management architecture from effectiveness and communication cost aspects. A certificate service is effective if it can react to changes of trust in the system in time. In our inter-CA trust chain, a change of trust will be propagated along each trust chain and CA's on the chain will update their trust tables according to the new change. We define the *vulnerable window* of a CA as the time between an event of trust change happens and the CA finishes updating its R table for the event, denoted by V. In a vulnerable window, a CA makes decision on trusting or distrusting other CA's based on its previous records, rather than the most recent events. The smaller V the more effective the system is. It is especially significant for a CA to deal with certificate revocation because small V ensures low false positive rate resulting from granting trust to CA's which should have been distrusted if it had received the most recent events.

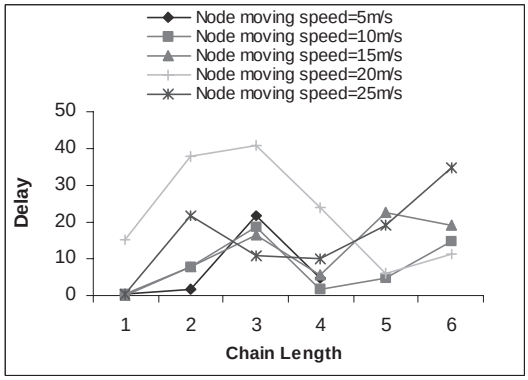

Figure 3. Vulnerable Window(V) vs Node Speed

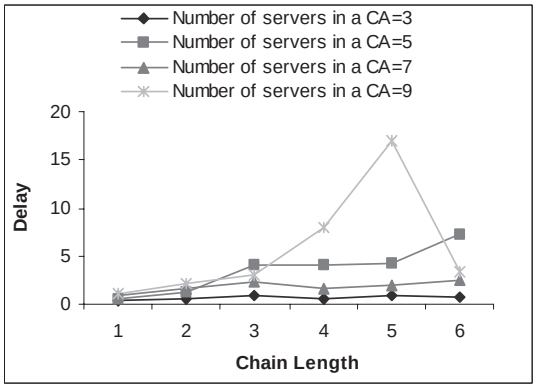

Figure 4. Vulnerable Window(V) vs number of servers(N) of CA

Another important perspective of the system performance is communication cost. The threshold cryptography enhances the security at the cost of messages since generation of a signature gets at least the threshold K number of nodes involved. This is even aggravated by the fact that no single node can be absolutely trusted and thus to ensure correct message delivery redundancy is brought in. We calculate the communication cost by the number of messages triggered by a single event. However, this number varies based on different logic topology of the trust relationships. To avoid being further complicated by these runtime variants, we only calculate the message cost when there is a single trust chain, denoted by M because that of multiple chains is upper-bounded by (number of the chains) $\times M$.

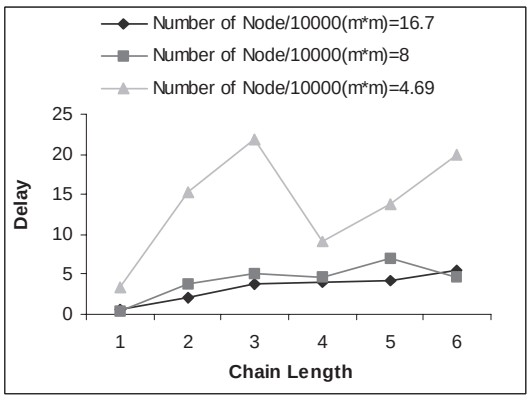

Figure 5. Vulnerable Window(V) vs Network Density

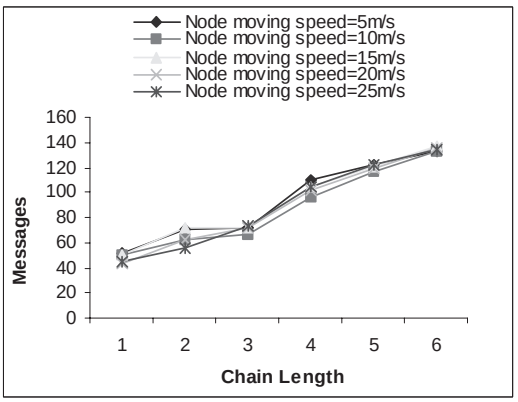

Figure 7. Message Cost(M) vs Node Speed

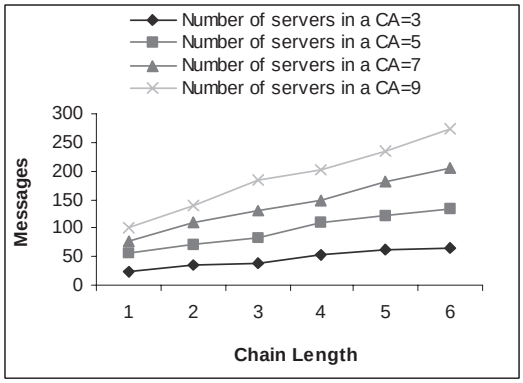

Figure 6. Message Cost(M) vs number of servers(N) of CA

5.2.2 Scenarios, Parameters and Results

We modify Carnegie Mellon University *setdest* [9] utility to generate random-way point mobility models with different node moving speed. The speed varies from 5, 10, 15, 20, 25m/s etc. The simulations are run with 150-300 nodes spread in areas ranging from 300x300, 500x500 and 800x800 m^2. The results are shown in Figure 3, 4, 5, 7 and 6. In the current simulation implementation, we simply use multiple unicasts to simulate a multicast. Therefore, from the aspect of underlying communication cost, the simulation results can be considered as the upper bound.

As shown in Figure 3, the vulnerable window V of the system varies with the nodes' moving speed but there is no evidence of a clear relation between them. The same observation holds for V vs CA composition relation as demonstrated in Fig 4. After analyzing the trace logs of the simulations, we find that routing between CA's dominates the trust propagation delay along the chain, which is relatively random compared to CA internal routing where usually routes are shorter. Fig 5 further supports this by showing V increases as density of the network, represented by the number of nodes per 10000 square meters(m^2), decreases in that sparser the network, farther two nodes are from each other and therefore more difficultly routes can be found.

Unlike the vulnerable window size V, message cost is in direct proportion to the number of servers(N) and threshold K of CA as expected as shown in Figure 6. Figure 7 shows that either network density or volatility does not impact the message cost significantly. This is because we do not rely on retransimission to fail over routing problem or network partition due to the high cost. Instead, we intend to use routine and event based updates to repair broken chains, in which if an update notification fails to reach other CA's, it is simply dropped and the system waits for either next routine or event-triggered update.

5.3 Limitations

The main weakness of our architecture is the commonly trusted dealer in each community. Schemes discussed in [14] allows partial shares of secret to be distributed to each CA servers without requiring the help of a commonly trusted dealer. We have not implemented them in our current prototype to conduct more experiments. In addition, the cost incurred by control messages of our architecture is evaluated in terms of the number of messages. To better understand the network overhead of the protocol, routing cost needs to be taken into account.

Certain optimizations can also be incorporated into the current implementation. The most important one is to implement the delay timer as mentioned in Section 4. Besides, one can use real multicast to send ESTABLISH and UPDATE messages instead of using multiple unicasts. With

these optimizations in place, the overhead of our architecture can be further reduced.

6. Acknowledgment

Special thanks go to Jiejun Kong from UCLA for his help on threshold cryptography programming.

7. Conclusions and Future Works

In this paper, we present a locality driven key management architecture for MANET. The design is motivated by our application oriented view of MANET and targeted for high fault tolerance, in-time services and efficiency. We implement a prototype and run simulations for different scenarios and the results support our design goals.

Our next step is to apply the architecture to a real application, such as high way traffic view. Moreover, we are also investigating more sophisticated trust propagation protocols which can be plugged in to our architecture.

References

[1] Entrust Certificate Service. http://www.entrust.com.

[2] Open Shortest Path First (OSPF). http://www.ietf.org/rfc/rfc1247.txt.

[3] OpenSSL Project. http://www.openssl.org.

[4] The Network Simulator - NS2. http://www.isi.edu/nsnam/ns.

[5] N. Asokan and P. Ginzboorg. Key agreement in ad-hoc networks. In *Nordsec'99 Workshop*, 1999.

[6] T. Aura and S. Mäki. Towards a survivable security architecture for ad-hoc networks. In *Security Protocols, 9th International Workshop, Cambridge, UK, April 2001*, volume 2467 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin. Springer-Verlag Berlin Heidelberg 2002.

[7] D. Balfanz, D. Smetters, P. Stewart, and H. Wong. Talking to strangers: Authentication in adhoc wireless networks. In *the Symposium on Network and Distributed Systems Security (NDSS '02), San Diego, California*, 2002.

[8] S. Berkovits, S. Chokhani, J. Furlong, J. Geiter, and J. Guild. Public key infrastructure study final report. *MITRE report*, 1994.

[9] J. Broch, D. A. Maltz, D. Johnson, Y. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Mobile Computing and Networking*, 1998.

[10] S. Capkun, L. Buttyan, and J. P. Hubaux. Self-organized public-key management for mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, Jan-Mar 2003.

[11] S. Dashtinezhad, T. Nadeem, C.Liao, and L. Iftode. Trafficview: A scalable traffic monitoring system. In *the proceedings of IEEE International Conference on Mobile Data Management*, 2004.

[12] Y.-C. Hu, A. Perrig, and D. B. Johnson. Ariadne: a secure on-demand routing protocol for ad hoc networks. In *the Proceedings of the 8th annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, 2002.

[13] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing robust and ubiquitous security support for mobile ad-hoc networks. In *the Proceedings of the 9th IEEE International Conference on Network Protocols (ICNP'01)*, 2001.

[14] M. Malkin, T. Wu, and D. Boneh. Experimenting with shared generation of rsa keys. *In Proceedings of the Internet Society's 1999 Symposium on Network and Distributed System Security (SNDSS)*, 1999.

[15] V. Pathak and L. Iftode. Byzantine fault tolerant authentication. *Technical Report, Dept of Computer Science, Rutgers University*, 2003.

[16] C. E. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *ACM SIGCOMM'94 Conf on Communications Architecture, Protocols and Applications*, 1994.

[17] C. E. Perkins, E. Royer, and S. R. Das. Ad hoc on demand Distance Vector(AODV) routing. In *2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*, 1999.

[18] V. Shoup. Practical threshold signatures. *Theory and Application of Cryptographic Techniques*, 2000.

[19] S. Yi and R. Kravets. Moca: Mobile certificate authority for wireless ad hoc networks. In *the Proceedings of the 10th IEEE International Conference on Network Protocols (ICNP'02)*, 2002.

[20] L. Zhou and Z. J. Haas. Securing ad hoc networks. In *IEEE Networks*, volume 13(6). 1999.

[21] L. Zhou, F. Schneider, and R. van Renesse. Coca: A secure distributed on-line certification authority. In *Technical Report of Cornell University*. 2002.

[22] P. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.