KONG, YUAN., M.S. A Distributed Public Key Caching Scheme in Large Wireless Networks. (2010)
Directed by Jing Deng. 25 pp.

When asymmetric cryptography techniques are used in wireless networks, the public keys of the nodes need to be widely available and signed by a Certificate Authority (CA). However, the existence of a single CA in large wireless networks such as mobile ad hoc networks and wireless sensor networks can lead the hotspot problem and become a security weakness. In this work, we propose a distributed technique to cache the public keys on regular nodes. Due to the limited memory size that each node is allowed to dedicate for key caching, only some keys can be cached. In our proposed technique, each node caches the public keys of a mix of local and remote nodes. The local nodes are defined as the nodes within the same neighborhood according to the transmission range, while the remote nodes are the ones outside the range. Access to the public keys of other nodes is possible based on a chain of trust. Multiple copies of public keys from different chains of trusted nodes provide fault tolerance. We explain our technique in detail and investigate its salient features in this work. An interesting observation is the need to balance caching public keys of local nodes and remote nodes. We studied the optimum local/remote public key caching ratios for different networks via investigating the availability of the number of required public key copies. These simulation results showed that by balancing the caching public keys with the optimum ratios, the availability of the required public keys kept increasing and finally became stable. We also did the simulation about studying the number of hops to find the first copies of required public keys. The results showed how local/remote ratios affected the minimum number of hops for reaching the first copies.

A DISTRIBUTED PUBLIC KEY CACHING SCHEME

IN LARGE WIRELESS NETWORKS

by
Yuan Kong

A Thesis submitted to
the Faculty of The Graduate School at
The University of North Carolina at Greensboro
in partial fulfillment
of the requirements for the Degree
Master of Science

Greensboro

2010

Approved By

Committee Chair

To my parents.

APPROVAL PAGE

This thesis has been approved by the following committee of the Faculty of The Graduate School at The University of North Carolina at Greensboro.

Committee Chair _____

Committee Members _____

_____

_____
Date of Acceptance by Committee

_____
Date of Final Oral Examination

ACKNOWLEDGEMENTS

First, I would like to thank my advisor, Jing Deng, for his great help with everything. Jing offered a lot of instructions during the whole process of accomplishing my thesis work. He helped me with reading the papers, running the simulations and writing the materials. He always encourage all the members in the group to think and learn. After each group meeting, I got new idea from he. He showed me different way to approach a research problem and the need to be confident to achieve any goal.

Besides, I would like to thank the rest of my thesis committee: Stephen R. Tate, who asked me good questions and offered me new ideas about extending my work, Fereidoon (Fred) Sadri, who gave a number of useful comments to help me express myself better.

I also want to say thank you to all the group members: Yanfen Song, Xiaocheng Zou, Alexey Bogaevski, Siddhiben Naik and Spoorthy Nimmagadda. Discussing with them helped me learn a lot. I enjoyed being with them in the lab for every meeting and for the daily life.

A special thanks goes to Yuesong Wang, who helped me a lot while I was preparing for the thesis defense. During the days that I could not walk since the sprains of my foot, he offered transportaion and made it possible for me to go to lab everyday.

I would also like to thank: Francine Blanchet-Sadri, for the two courses she tought me and the confidence she gave me, Nancy Green, for the course Software Engeneering, Shan Suthaharan, for the course Computer Networks, Lixin Fu, for the course Analysis and Design of Algorithms.

Last but not the least, I thank: Donna Balser, who is the secretary in the department of Computer Science, Richard Cheek, the system administrator in the department of Computer Science.

TABLE OF CONTENTS

Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

I.1 Security Issues in Large Wireless Networks

I.1.1 Large Wireless Networks

With the wide use of wireless devices, large wireless networks are expected to play an increasingly important role to provide networked information. An Examples of large wireless networks is wireless sensor networks (WSNs). WSNs are composed by a large number of sensors which can communicate with each other and monitor environmental conditions cooperatively, such as sound, light, pressure and temperature. In such networks, each node plays the same role and supports multi-hop routing. Since large number of sensor nodes are densely deployed in WSNs, neighbor nodes may be close to each other [2]. Thus, multihop communication is exploited in this kind of network in order to gain less power consumption.

Sensor nodes are usually distributed randomly in the field to form a wireless sensor network in an ad hoc manner for fulfilling certain task [7]. It is known that the idea of ad hoc networking has around for over 30 years [11]. In this kind of network, usually There are no preexisting infrastructures such as specific servers, while there are possibly some power control nodes. Since no access points or routers exist, each node in the network participates in routing and communicates with each other directly with multi-hop routing. One example of such application is the battlefield surveillance. After being randomly deployed in a battlefield, the sensor nodes need to be self-organized and form an ad hoc wireless network. Each sensor node then collects the condition information around

itself and sends the report observation messages to a central node via its neighbors. With the information collected from different member nodes, the activities in the battlefield can be deteted. This will help the control center to make appropriate and quick decision when attack happens.

With more and more information delivered on these wireless networks, security becomes a critical issue. Many traditional protocols and algorithms will not work with ad hoc networking. Security service such as authentications and key managment are essential for ensuring the normal oprations in hostile environments [7]. Authentication is the basis of secure communication [9]. It can only been realized as verifying something which is associated with an identity. Key management can be defined as a set of techniques and procedures that support the establishment and maintenance of keying relationships between authorized parties [8]. These services should adhere to the a number of security attributes including key availability. The high-availability feature prevents degradation of key managment services and ensure keying material is provided to nodes who expect this material [9].

I.1.2 Asymmetric Key Scheme

The asymmetric cryptography scheme can be used to provide information security. Using the asymmetric key scheme, each node in the network has a pair of keys: public key and private key. The public keys should be known by all nodes in the network; the private key should only be held by the node itself. For information confidentiality, a node uses the public key (of a receiver) to encrypt the message. This encrypted message can only be decrypted by the intended receiver who holds the matching private key. For information authentication, a node uses its own private key to sign the message. In order to authenticate the message, the receiver needs the public key of the sender.

However, the availability of these public keys can be an issue in large wireless networks. Usually, there is a Certificate Authority (CA) that will issue certificates for every node. Each certificate, signed by the CA, contains a public key and the identifier of a node. In

large wireless networks, the existence of such a CA can become a security weakness. For example, since the CA is known to the entire network, the adversary can attack it with all its resources. The traffic toward the CA can be mis-routed with the use of worm-hole attacks [5] or black-hole attacks [3]. Jamming attacks can be launched by the adversary to blackout all wireless communications in the CA's neighborhood.

I.2 Distributed Public Key Caching Scheme

In this work, we investigate a wireless network where every node can serve as a CA, or a Peer CA (PCA). Since the certificate from one PCA can be unreliable (imagine a compromised node serving as a PCA), the certificates from a number of PCAs will confirm that the public key of a certain node is valid. This is similar to threshold cryptography [17] and distributed trust [1].

Since every node in the network serves as a PCA, how the keys are stored becomes an essential issue. On one extreme end, we let each node only store the public key of itself. This results in a really key availability as there is only one public key for any specific node. In any some node are compromised, there is no way for other nodes to realize the existence of these compromises. On the other end, we let each node store the public keys of all the other nodes. In this way, there is no need to worry about the key availability. However, it is ineffecient to do so since we need to consider the source constraints of the nodes. Wireless devices are usually resource-constrained, in terms of computational power, battery energy, and on-board memory space. For example, the prototype sensors in WSNs have about 4K bytes of memory, which needs to support several important tasks: data collection and on-board processing; system parameter storage; and key storage. Meanwhile, a public key is usually as large as 128 bytes. The limited memory space that can be dedicated to key storage gives rise to the following problem: how should the keys be cached and updated in such large wireless networks?

In this work, we propose a distributed key caching scheme. In this scheme, each node caches the public keys of some other nodes in the network. We investigate our public key

caching scheme through the availability of the public key copies and observe the existence of an optimal local/remote ratio in different network scenarios. We also design a key update strategy that will allow nodes to update their public key caches according to the optimum local/remote ratio.

I.3 Related Work

Our work is mostly related to distributed trust establishment and security protection. In this section, we discuss several related work in the following.

In [14] and [15], mobile certificate authorities (MOCAs) were established for heterogeneous mobile ad hoc networks (MANETs) where some nodes are more reliable and resourceful than others. These MOCAs share the responsibility of collectively providing the CA functionality for the network, using threshold cryptography [17]. In the MOCA certification Protocol (MP), a client requiring certificate service broadcasts a Certificate Request (CREQ) message and waits for responses from at least k out of the n MOCAs. With such k responses, the certificate can be fully reconstructed and the certification process succeeds.

Similarly, in [10], a fully distributed trust model for MANET was introduced based on trust graph and threshold cryptography. In their model, users can issue public key certificates and authentication can be performed via certificate chains. To alleviate the adverse effect of malicious nodes in the network, threshold cryptography was used. Thus, a user needs to acquire k partial certificates for authentication.

In [6], a decentralized key management architecture was designed for WSNs. This architecture supports key deployment, key refreshment, and key establishment. Symmetric cryptography was used in the keying protocols in [6].

In the fully self-organized public-key management system presented by [4], all the users in a MANET can generate their own public/private key pairs. The users can also store, distribute, and revoke their public keys by themselves. Each node maintains a certificate repository using two ways: by communicating with its certificate graph neighbors; and

by applying a repository construction algorithm through a chain of trust [1, 12].

In [16], a composite key management scheme was designed to combine the public key infrastructure (PKI) technique and self-organized certificate chaining technique. The PKI usually uses threshold cryptography to adapt the ad hoc networks. The need with scheme is the virtual certificate authority (CA), which is comprised of multiple nodes. The virtual CA approaches require no warm-up period but impose higher maintenance overhead. The self-organized certificate chain fits ad hoc networking but needs a warm-up period. The work combined these two techniques by applying virtual CAs and certificate chain at the same time.

In [13], the key management (SEKM) scheme was introduced for mobile ad hoc networks. The scheme uses an number of server groups while each of them creates a view of the certificate authority (CA). The CAs provide certificate updating service for the nodes. The server nodes form an special group for communication providing the underlying service.

Our work differs from these related work in the sense that we focus on the use of limited memory space to cache the public keys of different nodes. We further design a public key search technique and a cache update algorithm to achieve optimum caching ratio of public keys from local/remote nodes.

I.4 Document Organization

The paper is divided into five chapters followed by the references and the appendix.

- Chapter I introduces the backgroud for our distributed key caching scheme and describes recent related works.
- Chapter II explains the public key caching scheme in detail.
- Chapter III shows the simulations results to evaluate our scheme.
- Chapter IV summarizes our work and discuss future works.
- Appendix A provides the code of the simulations.

CHAPTER II

PUBLIC KEY CACHING SCHEME

We first introduce our notations and variables

- PU_i: public key of node i;
- PR_i: private key of node i;
- $\{pt\}_{PR_i}$: message pt signed by PR_i;
- cm: crypted message;
- $\mathcal{C}_i$: set of nodes whose public keys are cached in node i;
- $\mathcal{N}_i$: set of physical neighbors of node i;
- $\mathcal{R}$: list of nodes that involve in the sequence of KREQ message transmission;
- m: total number of public keys that each node caches;
- ϵ: extensive ($\epsilon = 1$) or simple ($\epsilon = 0$) search;
- γ: ratio of the numbers of keys for local/remote nodes cached by one node;
- TTL: the maximum number of hops a KREQ message may travel.

II.1 Operational Details of the Proposed Scheme

Assume that every node carries the public keys of some other nodes. Such information can be obtained through pre-deployment key caching or through cache update, which will be discussed later.

KREQ Message Format

Source (S)	Destination (Q)	TTL	List of Routers (R)

KREP Message Format

Source (S)	Destination (Q)	$\{PU_Q\}_{PR_i}$	List of Routers (R)

Figure II.1: Message format of the KREQ and the KREP packets.

We first define two control messages that will be sent between nodes for public keys. These messages are called Key REQuest (KREQ) message and Key REPly (KREP) message. The KREQ message will be transmitted from the node requesting the public key of another node. The format of KREQ message is shown in Fig. II.1. In this figure, Source and Destination represent the node requesting the public key and the node's identifier whose public key is being requested; the list of routers ($\mathcal{R}$) represents the nodes who have been passing along the request.

The format of KREP message is similar (see Fig. II.1). The KREP message contains the public key of the destination and this key is signed by the current router's private key.

We explain the operational details of the KREQ and the KREP messages with an example. Suppose node S needs to obtain node Q's public key. It sends a KREQ = $\{S, Q, TTL, \mathcal{R} = \Phi\}$ to itself. Every node receiving a KREQ message should check whether it has cached Q's public key. If it does, it will sign the public key of Q with its own private key, store the result onto a KREP message, and return to the previous sender. Otherwise, it will attach its own ID to the list of routers, $\mathcal{R}$, and forwards it to each of the neighbors whose public keys are cached on its memory. The processing of KREP message is simple: a node receiving a KREP message should first authenticate the message, using the public key of the last sender. Then Q's public key will be signed by this node's private key and sent to the previous node according to $\mathcal{R}$.

There is a parameter ϵ, which controls whether a node caching Q's public key should

still forward the KREQ message. When $\epsilon = 1$, our scheme operates with extensive search. All nodes will forward the KREQ message. When $\epsilon = 0$, our scheme operates with simple search: only those nodes without Q's public key in their cache will forward the KREQ message.

The details of the above operations are presented in Algorithm 1. Nodes receiving KREP messages should proceed according to Algorithm 2.

Algorithm 1 Algorithm to Process KREQ Message

1: Node i receives a KREQ message=$\{S, Q, TTL, \mathcal{R}\}$
2: $d \leftarrow$ last node ID in $\mathcal{R}$
3: $\mathcal{R} \leftarrow \{\mathcal{R};\ i\}$
4: **if** $Q \in \mathcal{C}_i$ **then**
5: Prepare KREP message as $\{S, Q, \{PU_Q\}_{PR_i}, \mathcal{R}\}$
6: Node i sends KREP to node d
7: **if** $\epsilon = 0$ **then**
8: exit
9: **end if**
10: **end if**
11: $TTL \leftarrow TTL - 1$
12: **if** $TTL > 0$ **then**
13: Prepare KREQ message as $\{S, Q, TTL, \mathcal{R}\}$
14: **for** each $j \in \mathcal{C}_i \cap \mathcal{N}_i$ **do**
15: Node i sends KREQ message to node j
16: **end for**
17: **end if**

Algorithm 2 Algorithm to Process KREP Message

1: Node i receives a KREP message=$\{S, Q, cm, \mathcal{R}\}$
2: **if** $i \equiv S$ **then**
3: $d \leftarrow$ first node ID in $\mathcal{R}$
4: Node i computes $PU_Q = \{cm\}_{PU_d}$, where $d \in \mathcal{C}_i$
5: **else**
6: $s \leftarrow$ next node ID in $\mathcal{R}$ after i
7: $d \leftarrow$ previous node ID in $\mathcal{R}$ ahead of i
8: $m \leftarrow \{cm\}_{PU_s}$
9: Prepare KREP message as $\{S, Q, \{m\}_{PR_i}, \mathcal{R}\}$
10: Node i sends KREP message to node d
11: **end if**

II.2 Cache Update

Each node can update its cache when it receives or overhears a public key for another node through secured exchanges (such as the KREP message). We categorize the public keys that are cached on each node into two groups: local and remote, which represent the public keys of the local nodes that are direct neighbors of the node itself and remote nodes, respectively.

We assume that each node will be preloaded with the public keys of m randomly chosen nodes in the network. Therefore, the ratio of local/remote public keys may not be the specified value γ. As more and more KREQ/KREP exchanges take place, nodes can perform a cache update procedure as follows:

If the current local/remote public key ratio in cache is lower than γ and the public key of a local node is received, the public key will be used to replace one of the remote public keys. Similarly, if the current local/remote public key ratio in cache is higher than γ and the public key of a remote node is received, the public key will be used to replace one of the local public keys.[1] Different methods can be used to choose the key to be replaced, such as last-in-first-out, first-in-first-out, and most rarely used. In this work, we choose a random selection method among the keys in the same category.

Algorithm 3 Algorithm to Update Cache

1: Node i receives/overhears the public key of node j, PU_j
2: **if** $j \notin \mathcal{C}_i$ **then**
3: $\quad \gamma_{cur} \leftarrow |\mathcal{C}_i \cup \mathcal{N}_i|/m$
4: $\quad$ **if** $\gamma_{cur} < \gamma$ and $j \in \mathcal{N}_i$ **then**
5: $\quad\quad$ Update cache by replacing a randomly selected key that belongs to remote nodes with PU_j
6: $\quad$ **end if**
7: $\quad$ **if** $\gamma_{cur} > \gamma$ and $j \notin \mathcal{N}_i$ **then**
8: $\quad\quad$ Update cache by replacing a randomly selected key that belongs to local nodes with PU_j
9: $\quad$ **end if**
10: **end if**

[1] Care needs to be taken to avoid an oscillation effect, in which the key cache is frequently updated with the local/remote ratio fluctuating slightly above and below γ. We leave this to our future work.

This cache update algorithm is described in detail in Algorithm 3. We will investigate the effect of such cache updates in Chapter IV.

CHAPTER III

PERFORMANCE EVALUATION

Our simulations were performed in Matlab. We believe that the use of other simulators, such as ns2 or OPNET, are unnecessary because we are focusing on high-level public key caching and cache update process in our evaluation. Unless specified otherwise, these are the simulation setups: $N = 200$ nodes are randomly distributed in a network of size 1000 meters by 1000 meters. The radio transmission range is assumed to be 150 meters. Initially each node carries the public keys of a random set (size m) of the nodes in the network. Then we randomly select some source/destination pairs throughout the network. In each of the source/destination pairs, the source needs to request for the public key of the destination.

In our performance evaluation, we first focused on finding the number of public key copies (K) for the destination node that are available for the source within a limited number of hops. Another performance metric of our investigation was the optimum local/remote key ratio, γ^*. We also studied the minimum number of hops (h) of reaching the first public key copy for the destination node.

III.1 Available Copies

We investigate the availability of the public keys in this section. In this study, we assume that every node has a γ mix of local/remote nodes' public keys. Therefore, we focus on the stable states of the node caches and we assume that key caches have been assigned according to the local/remote ratio, γ. We run simulation for investigating the num-

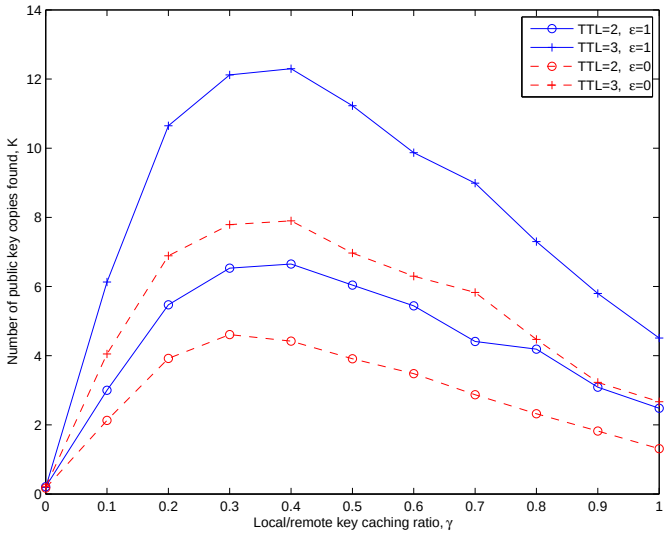

Figure III.1: Comparison of extensive search and simple search when $TTL = 2$ and $TTL = 3$ ($m = 40$).

ber of available required key copies with different parameter setting combinations. The parameters we use are m, TTL and ϵ.

In Fig. III.1, we compare the public key availability of extensive search ($\epsilon = 1$) and simple search ($\epsilon = 0$) with different TTL values. Naturally, the extensive search returned a larger K because the KREQ message in extensive search can reach more nodes than the simple search. The results in Fig. III.1 confirmed such an expectation. Furthermore, as TTL increases, the value of K increases because the source nodes query larger regions.

Another interesting observation from Fig. III.1 is the convex shape of the curves: as γ increases from 0 to 1, K increases with γ at the beginning and eventually decreases as γ increases closer to 1. This can be explained as follows: when γ is close to 0, each node uses its memory to cache mostly the public keys of remote nodes. The secure connectivity in the source node's neighborhood is limited (thus leading to small K), but increasing with γ. On the other hand, when γ is closer to 1, each node uses its memory to cache mostly the public keys of local nodes. The secure connectivity in the source node's neighborhood is good but these nodes are unlikely to have cached the public key of the destination node. Therefore, lowering γ in this region should increase K. Overall, there is an optimum value

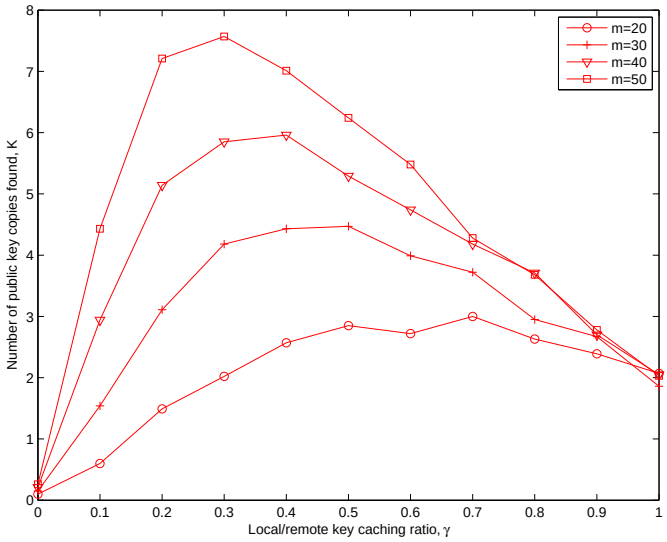

Figure III.2: Comparison of different memory size m ($TTL = 2$, $\epsilon = 1$).

for γ, which will be investigated in Fig. III.4.

Fig. III.2 shows the simulation results with different m when $TTL = 2$ and $\epsilon = 1$. As we can see from the figure, K increases with m. This can be explained by the increased number of public keys cached on each node, increasing the chance of finding the public key of the destination. An interesting observation is the K values at $\gamma = 0$ and $\gamma = 1$ for different memory sizes. They remain approximately the same for different m. The reason is that, when every node is caching only the public keys of local (remote) nodes only, the availability of the key largely depends on whether the destination node is a local node.

As we can see in Fig. III.2, here are also maximum pionts for each curve, which means that the corresponding optimum ratios ensure the highest key availability. It can be noticed that the the optimum ratios are abviously different. The larger m is, the smaller the optimum ratio is. The memory size m is an important factor that will decide what the optimum ratio is for a network. We will explain this when investigating the optimum values of γ in Fig. III.4.

Fig. III.3 presents the simulation results for TTL from 0 to 3. Similar to Fig. III.1, K increases with TTL. The results for $TTL = 0$ were presented for comparison purposes only; the source node should at least send the KREQ message to its 1-hop neighborhood.

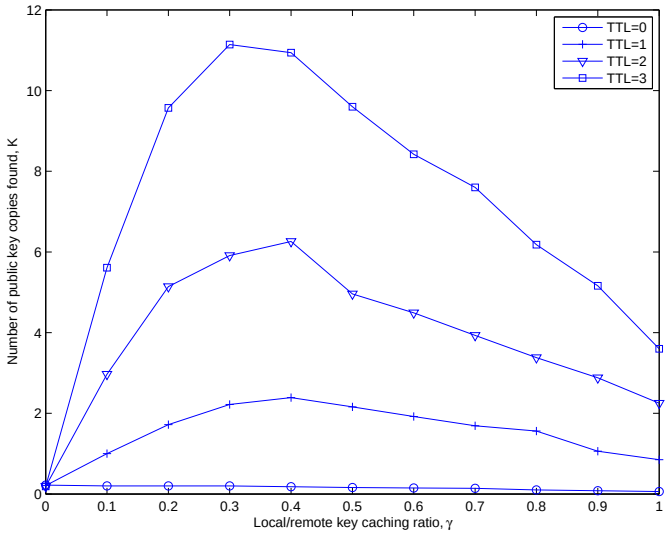

Figure III.3: Comparison of different TTL value TTL ($m = 40$, $\epsilon = 1$).

When $TTL = 0$, it means that the source node will only search the required public key in its own memory. There is only one chance for the source node to obtain the key. Thus the local/remote caching ratio will not affect K much if $TTL = 0$.

As we have seen in the above figures, there are always maximum points on the curves. This means that the local/remote ratio should be optimized to maximize the availability of the public keys for the destination node. Such an optimum ratio, γ^*, is investigated in Fig. III.4.

In Fig. III.4, we present the optimum ratios when we set different numbers for N. For each N, the optimum ratio becomes smaller as the memory size m increases. With the help from this figure, the optimum local/remote caching ratio can be set for a specific network. For example, for a network with $N = 200$ and $m = 50$, the optimum local/remote caching ratio γ^* should be set to 0.27. Similarly, the optimum ratios are presented in Fig. III.4.

Also can be observed in Fig. III.4 and Fig. III.5, the value of γ^* decreases almost linearly with the increase of m. We conjecture that $m\gamma^*$ should be a constant for a network of N nodes. For instance, when $N = 200$, $m\gamma^*$ is about 12-13. The product of $m\gamma^*$ for a network of $N = 400$ is 20-22. These numbers are certainly related to the density of the network, i.e., network region size and wireless transmission range. We leave

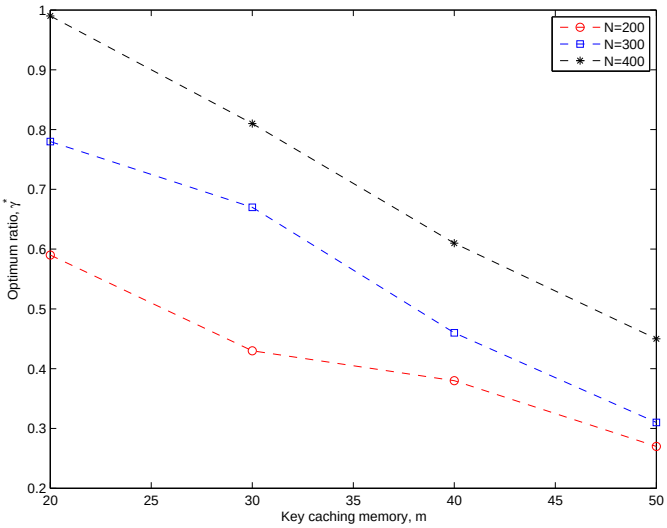

Figure III.4: The optimum ratio γ^* with different N ($TTL = 2$, $\epsilon = 1$).

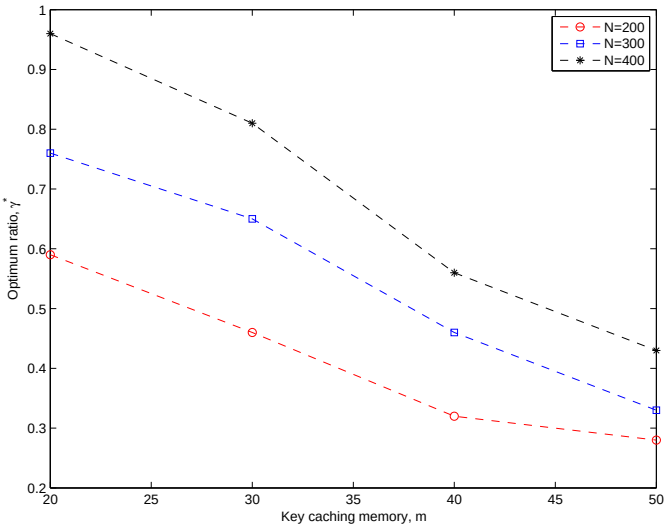

Figure III.5: The optimum ratio γ^* with different N ($TTL = 3$, $\epsilon = 1$).

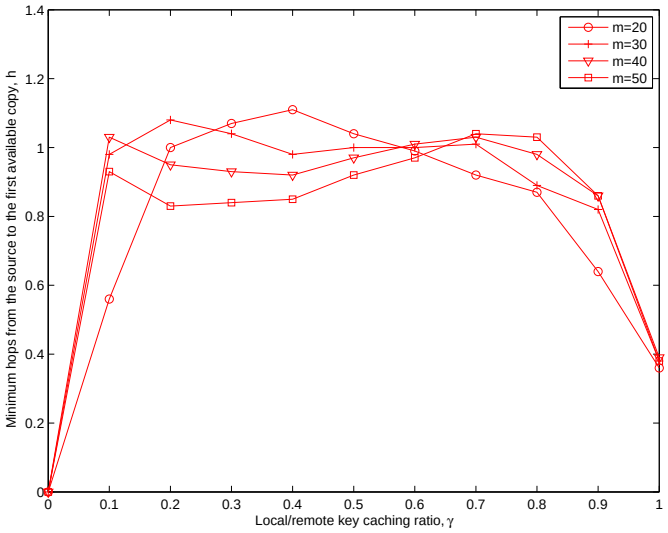

Figure III.6: The minimum number of hops h to find the first public key copy with different memory size m ($TTL = 2$, $\epsilon = 1$).

further investigation as our future work.

III.2 Minimum Hop

From the above simulation results, we know that the local/remote key cahing ratio γ affect the key availability within a certain number of hops. Moreover, an optimum ratio exists for a specific network setting. Will γ work for resulting in different number of hops to find the first copy of required public key? In this section, the minimum number of hops for finding the first key copy is investigated. In this simulation, we also focus on the stable states of the node caches, assuming that key caches have been assigned according to the local/remote ratio, γ.

Fig. III.6 shows the how minimum number of hops changes with the increasing of the key caching ratio γ. We can see that the average minimum hop count values shown in different curves are around 1. This is a good news since it means that the first copy can always be found within 1 hop, which is close to the source. There is no nice curve to show that some optimum ratio can minimize the hop count. This is because all values in the results are so close to each other that no big different can be shown.

III.3 Dynamic Systems

The previous observations tell us that there is an optimum ratio for ensuring the highest key availability. How can we apply this optimum ratio to a simple network with corresponding setting so that the key availability can be improved? In the further observation, we run the simulation as the algorithm in section II.2.

In this section, we investigate the changes of K as a randomly loaded network of nodes dynamically updating their key caches. According to the simulation result shown in Fig. III.4, we chose $\gamma^* = 0.27$ for a network with $N = 200$, $m = 50$, $TTL = 2$, and $\epsilon = 1$.

From the curves in Fig. III.7, we can see a clear upward trend of K as time goes on with more and more KREQ/KREP being exchanged. The cache updates improved the availability of required public key copies. After a certain number of cache updates, the value of K stabilizes. The TTL value had a huge effect on the rate of increase for K. As shown in Fig. III.7, a larger TTL allows K to reach stable (highest) value faster. This is because, with larger TTL, more nodes have the chance to update their public key caches in each KREQ/KREP message exchange. Thus, the local/remote caching ratios of the nodes in the network approached the designed optimum ratio, γ^*, much faster.

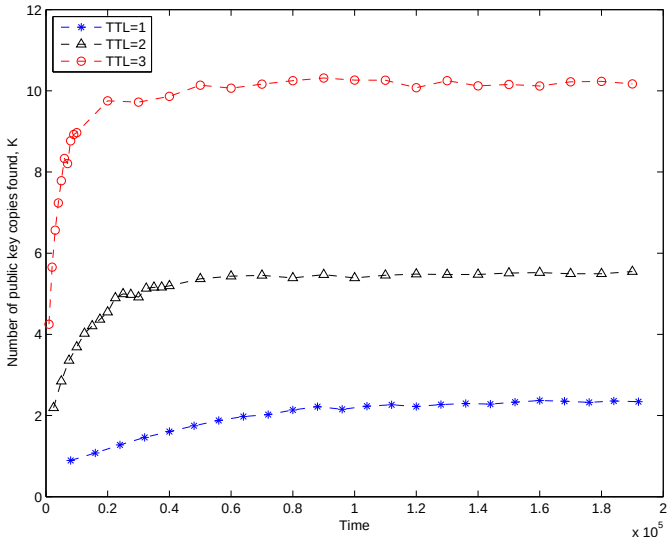

Figure III.7: Cache updates improved the availability of public key copies, K, ($N = 200$, $m = 50$, $\epsilon = 1$).

CHAPTER IV

CONCLUSION

Asymmetric cryptography requires the knowledge of the public key from the other party. These public keys are usually certified by a CA in many network systems. In large wireless networks, such CAs may not exist. In this work, we have proposed to use the nodes to serve as the peer CAs. A node requesting the public key of another node may obtain multiple copies from chain of trust. In order to achieve this goal, we have designed a scheme to allow nodes using their limited memory space to cache some of the public keys that they have securely obtained. We have also designed a public key search technique through secure multi-hop paths.

In the investigation of our scheme, we have observed that there exists an optimum caching ratio for the public keys of local nodes and remote nodes. On the one hand, if a node uses all its memory to cache the public keys of local nodes, the hop count of finding enough copies of the public key for a remote node is significantly large. On the other hand, if a node uses all its memory to cache the public keys of remote nodes, a key request may have to travel multiple hops before finding a public key copy. An optimum ratio of the public keys of local nodes and remote nodes should balance these two requirements, allowing the key request to find enough copies within a reasonably small hop count.

In our future work, we will implement our designed schemes under more realistic network environments. In our work, the nodes are evenly distributed in the network. However, a real network has more complicated topologies with the node joining and

leaving. For example, within a non-uniform network, there are usually several clusters with diffrent dense of the nodes. In this kind of network, it is uncertain that whether we can find a optimum local/remote key caching ratio for each node. Diffrent local/remote caching ratios may be assigned to the nodes in different clusters. Also, we will compare its computation and security performance with other state-of-the-art techniques. With the existence of malicious nodes in the network, public key copies may not agree with each other. Trust levels can be maintained and used to select among these copies.

REFERENCES

[1] A. Abdul-Rahman and S. Hailes. A distributed trust model. In *Proc. of the 1997 Workshop on New Security Paradigms*. ACM, 1997.

[2] L. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38:393–422, March 2002.

[3] M. Al-Shurman, S. Yoo, and S. Park. Black hole attack in mobile ad hoc networks. In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*, pages 96–97, New York, NY, USA, 2004. ACM.

[4] S. Capkun, L. Buttyan, and J. Hubaux. Self-organized public-key management for mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 2:52–64, 2002.

[5] Y. Hu, A. Perrig, and D. B. Johnson. Wormhole attacks in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24:370–380, 2006.

[6] Y. W. Law, R. Corin, S. Etalle, R. Etalle, and P. H. Hartel. A formally verified decentralized key management architecture for wireless sensor networks. In *Personal Wireless Communications (PWC 2003), Sep 2003. Lecture Notes of Computer Science*, pages 27–39. Springer-Verlag, 2003.

[7] D. Liu and P. Ning. *Security for Wireless Sensor Networks*. Advance in Information Security. Springer, 2007.

[8] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of applied cryptography*. CRC Press, 1996.

[9] J. Van Der Merwe, D. Dawoud, and S. McDonald. A survey on peer-to-peer key management for mobile ad hoc networks. *ACM Comput. Surv.*, 39(1):1, 2007.

[10] M. Omar, Y. Challal, and A. Bouabdallah. Reliable and fully distributed trust model for mobile ad hoc networks. *Computers and Security*, 28, May/June 2009.

[11] Y. Pan and Y. Xiao, editors. *Ad Hoc and Sensor Networks*, volume 2 of *Wireless Networks and Mobile Computing*. Nova Science Publishers, 2006.

[12] W. Stallings. The PGP web of trust. *Byte*, 2:161–162, February 1995.

[13] B. Wu, J. Wu, and E. B. Fern. Secure and efficient key management in mobile ad hoc networks. In *Proc. of 19th IEEE International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2005.

[14] S. Yi and R. Kravets. Key management for heterogeneous ad hoc wireless networks. In *10th IEEE International Conference on Network Protocols(ICNP'02)*, 2002.

[15] S. Yi and R. Kravets. Moca : Mobile certificate authority for wireless ad hoc networks. In *2nd Annual PKI Research Workshop Program (PKI 03)*, 2003.

[16] S. Yi and R. Kravets. Composite key management for ad hoc networks. In *Proc. of the 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2004)*, pages 52–61, 2004.

[17] L. Zhou and Z. J. Haas. Securing ad hoc networks. *IEEE Network Magazine*, 13(6), November/December 1999.

APPENDIX A

SIMULATION CODE

A.1 Simulation Setup

A.1.1 Network

$forwarding = 1;$

$TTL = 2;$ % set the value of TTL

$N = 200;$ % number of nodes

$R = 150;$ % tansmition range

$X = 1000 * rand(1, N);$ % randomly distribute the 200 nodes

$Y = 1000 * rand(1, N);$

%$-------$ the neighbor matrix $----------$

$NB = zeros(N);$

$fori = 1 : N$

$forj = 1 : N$

$NB(i, j) = ((X(j) - X(i))^2 + (Y(j) - Y(i))^2) <= R^2;$

end

$NB(i, i) = 0;$

end

A.1.2 Key Distribution

$key = zeros(N, m);$

$for i = 1 : N$

$nc = find(NB(i, :) == 1);$ % find the neighbors for node i

$dc = find(NB(i, :) == 0);$ % find the distant nodes for node i

$if(length(nc) <= m1)$ % assign the neighbor keys

$key(i, 1 : length(nc)) = nc;$

$else$

$index\_nc = randperm(length(nc));$

$nc\_ind = index\_nc(1 : m1);$

$key(i, 1 : m1) = nc(nc\_ind);$

end

$if(length(dc) <= m2)$ % assign the distant keys

$key(i, m1 + 1 : m) = dc;$

$else$

$index\_dc = randperm(length(dc));$

$dc\_ind = index\_dc(1 : m2);$

$key(i, m1 + 1 : m) = dc(dc\_ind);$

end

end

%–convert the key matrix to a N*N matrix

$N\_key = zeros(N);$

$for i1 = 1 : N$

$nz\_key = nonzeros(key(i1, :));$

$N\_key(i1, nz\_key) = 1;$

end

%−− Calculate the trust connection matrix $NB.N\_key$

$TC = zeros(N);$

$TC = NB.*N\_key;$

A.2 Available Copies

$for i4 = 0 : TTL$

$next\_n = [];$

$hits\_sub = 0;$

$for cur\_n = cur\_nodes$

% does cur_n carry public key of dn?

$if\ N\_key\ (cur\_n, dn) == 1$

$hits\_sub = hits\_sub + 1;$

end

% if forwarding or cur_n doesn't carry the public key % of dn

$if((\ N\_key\ (cur\_n, dn) == 0)\ |(forwarding))$

$cur\_nbs = find(TC(cur\_n, :). * (visited == 0));$

$next\_n = [next\_n, cur\_nbs];$

end

end %for $cur\_n = cur\_nodes$

$hits(i4 + 1) = hits(i4 + 1) + hits\_sub;$

$cur\_nodes = unique(next\_n);$

$visited(cur\_nodes) = 1;$ % mark the nodes in R

end %for i4