

國立交通大學

資訊科學與工程研究所

碩士論文

在無線隨意網路中使用完美雜湊族的高效率秘密

更新協定設計



Efficient Share Renewal Protocol Design for
Mobile Ad Hoc Network using Perfect Hash
Families

研究生：許鴻祥

指導教授：葉義雄 教授

中華民國九十五年七月

在無線隨意網路中使用完美雜湊族的高效率秘密更新協定設計

研究生：許鴻祥

指導教授：葉義雄 博士

國立交通大學資訊科學與工程研究所碩士班

摘要：

由於無線隨意行動網路 (Mobile Ad Hoc Network) 的一些先天特性，例如不可靠的無線環境、節點的移動性、不需要任何基地台或移動轉換中心的協助等等，使得提供安全通訊成為一個很大的挑戰。然而，一般用在有線網路上的 PKI 架構也無法直接移植到無線隨意行動網路的環境下，因為一個集中式 CA 是很難建構在無線隨意行動網路中。因此我們必須解決此集中式的現象。

本論文提出一個利用「完美雜湊族」(Perfect Hash Families) 的模式來實現 (n, k) 閾值秘密共享，將私密的金鑰分散給在網路上的每個節點，由一定個數的節點共同做簽章的動作，並且採用預防式秘密共享來更新私密金鑰，以避免長時間擁有相同的金鑰，增加攻擊者攻擊的難度。此外，我們利用完美雜湊族的特性，使得更新金鑰的程序更加有效率。最後，我們會分析此方法與前人所提的各式方法的差異以及效能比較。

關鍵字：隨意行動網路、 (n, k) 閾值秘密共享、預防式秘密共享、完美雜湊族。

Efficient Share Renewal Protocol Design for Mobile Ad Hoc Network using Perfect Hash Families

Student : Hung-Hsiang Hsu

Advisor : Dr. Yi-Shiung Yeh

Institute of Computer Science and Information Engineering National Chiao Tung University

Abstract :

Due to the inherent characteristic, such as unreliable wireless media, host mobility and lack of infrastructure, providing a secure communication platform in a mobile ad hoc network is a big challenge. However, common authentication schemes like PKI, which is used extensively in wired network, are not applicable in the ad hoc network environment because public key infrastructure with a centralized certification authority is rather difficult to deploy here. Thus, the centralized circumstance needs to be solved.

This thesis propose a scheme using the perfect hash families to implement the (n, k) threshold secret sharing. We separate the private keys into several shares and distribute them to every node in the mobile ad hoc network. Only a fixed number of nodes can sign the signature collaboratively. We also use the proactive secret sharing to update the private shares. It can avoid one node holding the same secret share for a long time and can increase the difficulty to being attacked. Moreover, we use the property of a PHF to do the proactive secret sharing, resulting in a more efficient update procedure. Finally, we analyze the performance of this scheme and compare our system with other previously mentioned methods.

Key words: Mobile Ad Hoc Network, (n, k) threshold secret sharing, Proactive Secret Sharing, Perfect Hash Families.

致 謝

這篇論文能夠順利的完成，首先我必須感謝我的指導教授，葉義雄教授，這兩年來教了我許多，不論是在學問上或是待人處事方面都令我獲益良多。接著很感謝的是高銘智學長，在我遇到瓶頸時，給了我很多的寶貴意見以及協助；還有，也謝謝實驗室裡的各位學長姊、同學與學弟妹，小白、阿甘、昇哥、英宗、雅婷、伯昕與 Gobby，大家平日的相互勸勵與學習，讓我成長許多；尤其是雅婷，在期間幫我一起跑程式，讓我因此節省許多寶貴的時間，真是太感謝了。此外，還要感謝在美國的怡君，特地撥時間幫我修改英文寫作，改正了我許多寫作的問題與錯誤。

最後，要謝謝我的父母，由於你們多年來的栽培與支持，才会有今天的我，真的非常非常的感謝你們。

謹將我這篇論文獻給所有關心我與支持我的人，謝謝你們。

許鴻祥

中華民國九十五年七月

Contents

摘要 :	ii
Abstract :	iii
致 謝	iv
Figure List	vii
Table List	viii
Chapter 1 Introduction	1
1.1 Background	1
1.2 Motivation	3
1.3 Related Work	4
1.4 Organization	7
Chapter 2 Related Knowledge	8
2.1 Network Security	8
2.1.1 Requirement	8
2.1.2 Types of Attacks	10
2.2 Fundamental of Cryptography	12
2.2.1 Symmetric Cryptography	13
2.2.2 Asymmetric Cryptography	14
2.3 Public Key Infrastructure	16
2.3.1 PKI overview	18
2.3.2 Certificate and Certification Authority	20
2.4 (n, k) Threshold Secret Sharing	21
2.5 Proactive Secret Sharing	22
2.6 Combinatorial Object	24
2.6.1 Introduction to Perfect Hash Families	25
2.6.2 Propositions of Perfect Hash Families	27
2.6.3 Construction methods of Perfect Hash Families	28
2.7 Fundamentals of Mobile Ad Hoc Network	33
2.7.1 Mobile Ad Hoc Network	34
2.7.2 Characteristics of Mobile Ad Hoc Network	34
2.7.3 Security challenges of Mobile Ad Hoc Network	35
Chapter 3 System Architecture	37
3.1 Concept	37
3.1.1 Construction	37
3.1.2 Partition	41
3.1.3 Conceptual Building Blocks	43
3.2 System Assumption	44
3.2.1 Intrusion model	44

3.2.2	Trusted dealer	45
3.3	Details and Protocols	45
3.3.1	System initialization	45
3.3.2	Certification service.....	47
3.3.3	Updating the Secret Shares.....	49
Chapter 4	Evaluation and Analysis	54
4.1	Evaluation.....	54
4.2	Analysis	56
4.3	Discussion.....	61
Chapter 5	Conclusion.....	63
References:	64
Appendix A	66
Appendix B	70



Figure List

Figure 2.1.1	Passive Threats.....	10
Figure 2.1.2	Active Threats	11
Figure 2.2.1	Symmetric Cryptography	14
Figure 2.2.2	Asymmetric Cryptography	15
Figure 2.3.1	Man-in-the-middle Attack.....	17
Figure 2.3.2	Components and Process of PKI	19
Figure 2.7.1	Mobile Ad Hoc Network.....	34
Figure 3.1.1	Finding the minimal prime power q	38
Figure 3.1.2	Construction Algorithm of PHF.....	38
Figure 3.3.1	Construction Algorithm of the whole Network.....	46
Figure 3.3.2	Request for Certification.....	49
Figure 3.3.3	Certification Service	49
Figure 3.3.4	Algorithm of Changing the Partition Header	50
Figure 3.3.5	Share Refreshing.....	52
Figure 4.2.1	PHF, $q = 3$	57
Figure 4.2.2	PHF, $q = 4$	58
Figure 4.2.3	PHF, $q = 5$	58
Figure 4.2.4	PHF, $q = 3$	59
Figure 4.2.5	PHF, $q = 4$	59
Figure 4.2.6	PHF, $q = 5$	60
Figure 4.2.7	PHF, $q = 7$	60

Table List

Table 2.6.1 PHF(4; 9, 3, 3)	26
Table 2.6.2 Construction Methods	29
Table 3.1.1 A PHF(4; 9, 3, 3)	40
Table 4.1.1 Communication cost	55



Chapter 1 Introduction

1.1 Background

In the recent decades, with the development of science and technology, the internet has clearly become an indispensable part of our lives. A wide variety of applications and services are offered through the internet. For example, through the internet libraries can establish joint borrowing and returning systems, and we can also make flight and hotel reservations for destinations worldwide in advance. By using the internet, it is more convenient and easy to communicate with one another. Through some simple internet applications such as e-mail and instant messenger, we can easily transmit messages or even files to another party of the communication. The internet has helped bridge the barriers of time and distance through enabling communication with virtually no boundaries. In addition, a current trend is the rapid growth and development of wireless network, as evidenced in the increased numbers of wireless related applications. Without any wired help, users can now use their wireless devices, such as PDAs and laptops, to connect to the internet via various wireless transmission protocols including, for example, 802.11b and Bluetooth. Wireless network provides users with greater flexibilities and convenience. However, as an increasing amount of sensitive and important information being transmitted electronically, network security is an increasing concern. Thus, in order to ensure the integrity and confidentiality of transmitted data, people are paying more attention to network security related issues.

As we continue to expand on the conceptual framework for guiding further development in network security, we need to understand the different types of attacks. Attacks can generally be divided into two categories: passive attack and active attack. A passive attack is

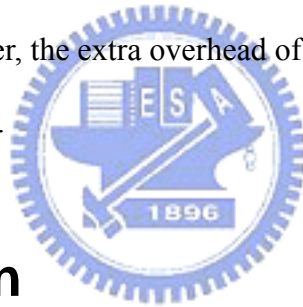
when adversaries simply want to obtain information off the network; it includes, for example, eavesdropping and peep. An active attack, on the other hand, is when adversaries not only read information off the network but also want to modify or write data to the network; it includes, for instance, masquerade, replay attack, and denial of service.

In order to resist aforementioned attacks, a security mechanism should satisfy following requirements: confidentiality, authentication, integrity, non-repudiation, access control, and availability [1][2]. These requirements can be supported by cryptography including both symmetric and asymmetric approaches. In addition to the cryptography technology, a suitable infrastructure also needs to be constructed. Public key infrastructure (PKI) is one of such structures. PKI supports a trusted third party called Certification Authority (CA) [3]. CA has its own key pair (public key / private key). CA uses its private key to sign the digital certificate, and the certificate then includes the user's identity and his corresponding public key. Other people can verify the certificate by using the CA's public key. If the certificate is valid, it can then be concluded that the content of the certificate is indeed valid. Depending on the certificates, users can authenticate the identity and transmit the data safely.

However, deploying security mechanisms is rather difficult due to the inherent properties of ad hoc networks, such as the high dynamics of their topology, limited resources of end systems, or bandwidth constraints and possible asymmetrical communication links. Any centralized design entity of security service is not practical in the ad hoc network because such entities would obviously become the targets of attack. Therefore, it is impossible to implement a centralized CA for managing public keys of the participants, because if the centralized CA is compromised, the attackers would be able to obtain much useful and sensitive information. The attackers can even impersonate a valid CA and consequently issue the certificate to the users. In this case, intruders can steal the transmitting data easily. Furthermore, due to the high mobility of each node, if the node, playing the role of CA, leaves the network, then other users would not be able to find the CA. Therefore, they cannot apply

for a certificate or manage the certificate. In this case, the entire network system may crash. Hence, a distributed solution must be found instead.

One popular method to achieve the distribution condition is to implement the (n, k) threshold secret sharing [4]. In fact, there has been much research focused on using the (n, k) threshold secret sharing method to construct the distributed CA. In this paper, we propose and evaluate a new architecture for securing communication in the mobile ad hoc network. More specifically, we use the perfect hash families (PHF) [5][6] approach to construct the network along with the threshold method. Also, we distribute the CA function and network secret to multiple nodes. Furthermore, we adopt proactive secret sharing [7] to improve network security. By using the characteristics of the PHF, we can make the proactive secret sharing more efficient and also enhance the effectiveness of this approach in the event when some nodes are unavailable. Moreover, the extra overhead of doing secret shares updating can be shared among a group of nodes.



1.2 Motivation

As wireless technology and applications continue to develop and gain popularity, there is a high demand in developing a secure wireless network. Considering some inherent limits in the mobile ad hoc network, the traditional PKI cannot be directly applied to the mobile ad hoc network without any modification. Therefore, our goal is to make some changes to the traditional PKI and ultimately implement it in the mobile ad hoc network.

The most important component in the PKI structure is the CA. In order to ensure availability and high survivability, we implement a decentralized CA by distributing the CA's functionality to many nodes. We use Adi Shamir's (n, k) threshold secret sharing to do it. The CA has a key pair (public key / private key). The secret key is divided into n shares, and these shares are owned by n nodes. Only when k or more than k nodes combine their own secret

shares cooperatively, they can collectively function as the role of CA. The original secret key is, however, not visible or known by any component of the network except at the system bootstrapping phase. Then these k nodes sign the certificate collectively and issue it to the user. The CA's public key is public in the network, so every user can easily verify the validity of a certificate. Users can then use the certificates to authenticate others' identities and communicate securely. We construct the (n, k) threshold secret sharing by using the PHF. Then we use the partition characteristic of the PHF to divide the entire network into several disjoint partitions. We use the partition point of view to do the proactive secret sharing. In this way, we can update the secret shares at a lower communication cost. Additionally, we can ensure a higher rate of success in doing the secret shares updating. Moreover, since our system does not require the synchronization of the network in the update process, we can prevent some attacks of synchronization. Finally, we can also securely transmit the new shares to each participant who needs it.



1.3 Related Work

There has been much research focused on the mobile ad hoc network security. In [8], Zhou and Hass proposed a secure key management scheme. They used (n, k) threshold cryptography to distribute trust among a set of servers. They focused on the security of the shared secret in the presence of possible compromises of secret share holders. The system can tolerate $k-1$ compromised servers. However, they did not provide any specific explanation for how a node can make contact to sufficient servers, especially when the servers are spread across a large area. The authors also proposed to employ proactive schemes to achieve share refreshing to counter mobile adversaries. Yet, their solution assumed the group of servers with rich connectivity. It is not suitable for ad hoc environments. The authors also did not address the issue of how to distribute the update shares to the server nodes efficiently and securely.

In [9], Kong et al. also used threshold secret sharing mechanism to distribute the functions of CA to some nodes. In order to ease the difficulty of contacting server nodes, they employed localized certification schemes. In other words, each entity holds a secret share, and multiple entities in a local neighborhood then jointly provide complete services. This method also can enhance the service efficiency for users. The authors noted that k is the balance point between service availability and intrusion tolerance. However, in their scheme the threshold value k is difficult to set. It is known that if k is too small, the probability of a global secret key being compromised is quite high. On the other hand, if k is big, although we can resist more compromises, it is relatively harder to find k one-hop legitimate neighboring nodes. Another problem is that they also did not address the issue of how to distribute the update shares to the server nodes efficiently and securely. [10] is an extension of [9] because the authors proposed the parallel share updates to prevent from emulating a coalition of k nodes to fake share updates. Yet, this method requires a much higher communication cost due to the fact that each update polynomial function has to be generated by k nodes collaboratively. After the polynomial function is generated, each node that wants to do the update procedure would be required to ask k nodes again to decrypt the polynomial function, then to complete the update. The authors also implemented a localized certification service to enhance service availability for mobile nodes and robustness against DoS attacks. However, this localized certification service operates under the assumption that each node has at least k legitimate neighbors; it surely has some difficulties to ensure that in a mobile ad hoc network environment.

In [11], Bechler et al. proposed and evaluated a clustered architecture for securing communication in mobile ad hoc networks. They divided the network into clusters and used threshold cryptography to implement a decentralized CA. The authors further separated the cluster internal traffic from the network-wide traffic. For cluster internal traffic, they used the symmetric encryption. For network-wide traffic, they used public key cryptography. There are,

however, two major problems with their proposed architecture: (1) with its log-on procedure and (2) with its sharing update. First, they did not specify how to find the warrant nodes, and also the number of warrant nodes is indeterminate. Second, they used proactive secret sharing without any modification. Therefore, the communication overhead is too high for the wireless channel.

In [12], Zhu et al. proposed a novel key management scheme based on the hierarchical structure and secret sharing to distribute cryptography keys and to provide certification services, called the Autonomous Key Management (AKM). AKM is a logical tree, in which all the leaf nodes represent real wireless nodes, while all the branch nodes only exist logically. AKM can achieve flexibility and adaptivity by issuing certificates with different levels of assurance and can handle the mobile ad hoc network (MANET) with a large number of nodes. They further proposed two algorithms, which are based on threshold cryptography and Verifiable Secret Sharing (VSS). These algorithms can resist active attacks targeting certification services. The disadvantage of AKM is that if we want to change the configuration of (n, k) to (n', k') , it would require a significant cost. Under their “join operation,” when one real node wants to join a region, the system would choose a group of k nodes randomly. The authors assumed that each node in that group should know the identity of one another. However, their assumption is not sound, since each node is randomly chosen.

In [13], Wu et al. also adopted the threshold cryptography to distribute the private key share to shareholders. The major difference in this presented model is that the shareholders form a special group, called the server group. In the previous approach the shareholders are all independent; users must communicate with each server node individually. In contrast, here a user only needs to communicate with one member of the server group, then that server node will send the information to other server nodes automatically. The advantage of this method is that it is easier for a node to request service from a well maintained group rather than from multiple “independent” service providers, which may spread across a large area. Furthermore,

the server group does not have to include all shareholders; it takes the soft state maintenance to ensure a number of shareholders. In sum, the benefits include communication-efficiency, bandwidth-saving, and easy management. However, the size of a server group is the determinant of the entire network performance. That is, if the server group is small, it is then relatively easier to respond and manage. Yet, this kind of small server groups may not have the ability to serve a large network. On the other hand, if the group is big, the response rate would, as expected, be slower and thus would have an impact on the entire network performance. Therefore, how to decide on the size of a server group would be key in determining the quality of network performances.

1.4 Organization

This thesis is organized as follows. First, it begins with a brief overview of related knowledge in chapter 2. Specifically, it includes backgrounds on cryptography and security, mobile ad hoc network concepts, principles of secret sharing, and some combinatorial objects. Then chapter 3 focuses on the specific concepts and describes the detailed protocol of our scheme. In chapter 4, we present the analysis of mobile ad hoc network that uses our scheme in order to show its availability and performance. Results and related parameters are also discussed in this chapter. Finally, chapter 5 provides conclusions pertaining to the proposed scheme and results and suggestions for some future research directions based on this thesis.

Chapter 2 Related Knowledge

2.1 Network Security

2.1.1 Requirement

Network security is an increasingly important issue, especially for those security-sensitive applications. In order to ensure the safety of data transmission process in the network, the following attributes need to be considered [1][2]:

Confidentiality

Authentication

Integrity

Non-repudiation

Access control

Availability



Below is a description for each requirement.

(1) Confidentiality

Confidentiality is the protection of transmitted data from passive attacks. It ensures that the data only be accessible by authorized entities. The other aspect of confidentiality is the protection of traffic flow from analysis. Network applications with sensitive information transmission require confidentiality. If such kinds of information are leaked out to the adversaries, it may result in devastating consequences.

(2) Authentication

Authentication is the process that enables a node to ensure the identity of the peer node that it is communicating with. Without authentication, adversaries could masquerade a node and thus obtain unauthorized access to resource and sensitive information.

(3) Integrity

Integrity is the measure of ensuring the correctness and completeness of transmitted information. It guarantees that data are received as sent, with no insertion, modification, reordering, or corruption.

(4) Non-repudiation

Non-repudiation prevents either sender or receiver from denying a transmitted message. Also, the sender and receiver have ways to prove that they actually have sent and received the message respectively. Essentially, non-repudiation is the process that holds senders and receivers accountable for sending or receiving any data. Non-repudiation is also useful for detecting and isolating compromised nodes. When node A receives an erroneous message from node B, non-repudiation allows A to accuse B for sending the message and to convince other nodes that B has, indeed, been compromised.

(5) Access control

Network resources are limited. If unauthorized nodes use the network resources unrestrictedly, it may endanger the entire network. Therefore, access control is the ability to permit or deny the access to network resources. To achieve this control, each entity has to obtain authorizations prior to any use of the resources.

(6) Availability

Availability ensures the survivability of the network service. Many kinds of attacks can result in loss or reduction in the availability of services. Thus, there is a strong need for finding solutions to deal with this situation and to let only authorized nodes get the service easily.

2.1.2 Types of Attacks

Generally speaking, attacks can be classified into two categories: passive attack and active attack. Below is a description of these two types of attacks [1]:

(1) Passive attack

A passive attack is an attack where an unauthorized attacker monitors or listens to the communication between two parties. Figure 2.1.1 shows the two different types of passive attacks.

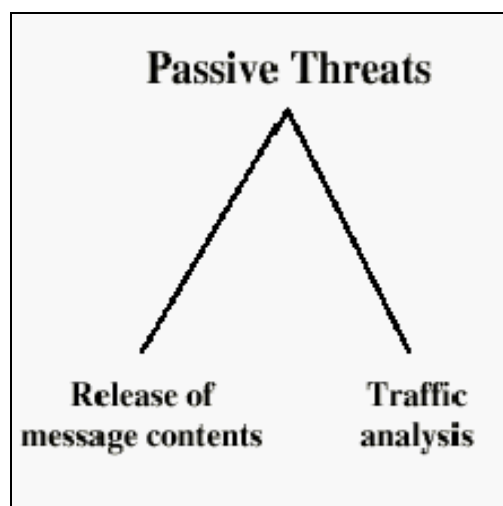
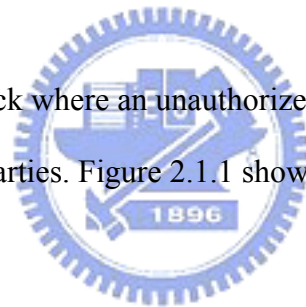


Figure 2.1.1 Passive Threats

One kind of passive attack involves an unauthorized **release of message contents**.

Another type is the **traffic analysis** threat, which is much more subtle. It means that encryption masks the content of the transmitted message, so even if captured by attackers, they would be unable to read the content of the message. Although the attackers cannot see the content, they nevertheless could determine the location of the sender or some other important information, such as the communication model.

Passive attacks are very difficult to detect because they do not involve any alteration of the data or message. Due to the difficulty in detecting passive attacks, the primary defense strategy is in prevention rather than detection.

(2) Active attack

In active attacks, adversaries may modify the data stream or create a fake reply. The attacker may also transmit data to one or both parties, or block the data stream in one or both directions. It is also possible that an attacker could deceive user A into believing he is user B and could deceive user B into believing he is user A. In other words, users A and B do not know that the communication link between them has been compromised. Figure 2.1.2 shows various types of active attacks.

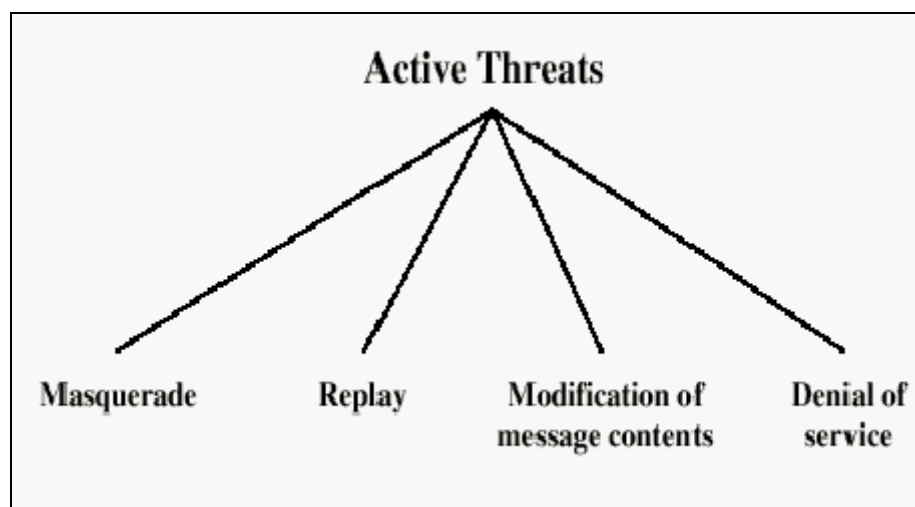


Figure 2.1.2 Active Threats

Masquerade is an attempt to act like or to impersonate someone else or some other system.

Replay attack means that attackers first intercept the valid data and then subsequently retransmit the data to fulfill their illegal intention.

Modification of message contents simply means to produce an unauthorized effect by altering or reordering some portions of the message.

Denial of service (DoS) attacks lead to a loss of services and resources to legitimate users in the system.

In general, it is also quite difficult to prevent active attacks because it would require a complete protection of all communication facilities to do so. Contrary to passive attacks, the primary defense strategy to active attacks is to detect them and recover any disruption or delay caused by them.

2.2 Fundamental of Cryptography

Cryptography is the science of keeping secrets secret. The origin of cryptography traces back millenniums ago. When humans had learned how to communicate with one another, they had no choice but to find methods to keep their private messages secret. Therefore, cryptography is the science of encoding a message such that only the sender and the intended recipients are able to understand it. "Plaintext" refers to a message before encryption, and after the encryption, the corresponding message is known as the "ciphertext."

Cryptography is fundamental to the network security. All of the security requirements discussed in the earlier section of this text can be achieved by relying on cryptography algorithm. Cryptography algorithm could involve some replacement or permutation techniques. It also could be based on some kinds of mathematical function operations. With the participation of some secret information, usually referred to as a "secret key," we use the

cryptography algorithm to transform the plaintext into unreadable ciphertext, which is something unintelligible to anyone other than an authorized recipient.

Generally speaking, depending on how the secret key is used, cryptography can be classified into two categories: symmetric cryptography and asymmetric cryptography. The following is an introduction of each cryptography approach [14][15].

2.2.1 Symmetric Cryptography

Symmetric cryptography uses the same symmetrical key for both encryption and decryption. Without the secret key, it is impossible to recover ciphertext back to its original plaintext. Therefore, the secret key has to be stored secretly. Two popular symmetric cryptography techniques are DES and AES. There are several weaknesses associated with the symmetric cryptography approach: 1. The fact that encryption and decryption parties share a same key, before transmitting any data, the secret key must be securely sent to both parties first. However, how key exchange process can be kept secret is a major problem. 2. Another weakness is related to the difficulties in managing the key, as one secret key is shared by users altogether. 3. Lastly, if a secret key is compromised, then the key must be destroyed and replaced with a new one. The effort to distribute the new secret key to all users would be enormous. Despite its disadvantages, symmetric cryptography is nonetheless an efficient approach to encrypt and decrypt messages.

Figure 2.2.1 illustrates the operation of symmetric cryptography. The sending party and the receiving party use the same secret key and the same encryption model to do the encryption and decryption.

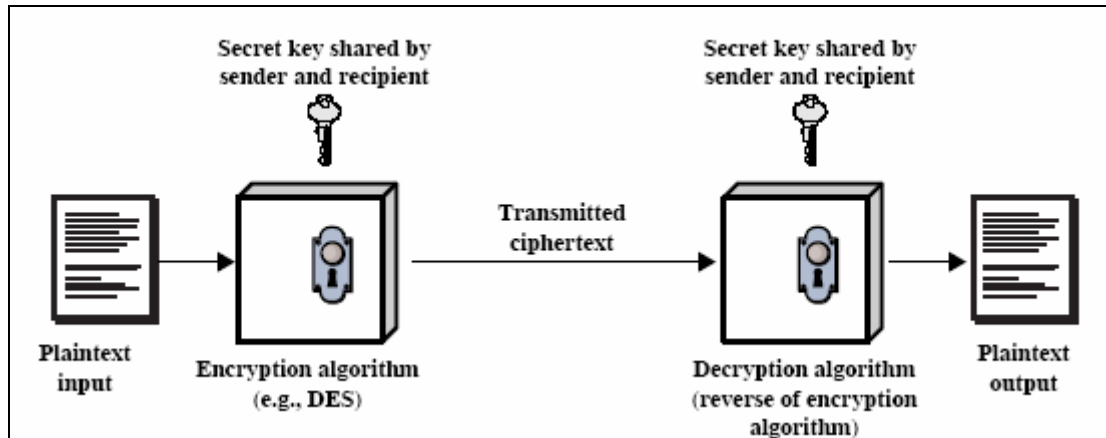


Figure 2.2.1 Symmetric Cryptography

2.2.2 Asymmetric Cryptography

Asymmetric cryptography, also known as public-key cryptography, is perhaps the most important development in the history of cryptography. Introduced by Whitfield Diffie and Martin Hellman in 1976, the concept of public-key cryptography was specifically developed to solve problems related to the secret key distribution and management in symmetric cryptography. Public-key cryptography has, in fact, taken cryptography development to a new direction and network security to a new level. First, the asymmetric algorithms are based on mathematical functions instead of replacement or rearrangement techniques. Then, most importantly, the asymmetric cryptography uses two asymmetric keys. Asymmetric cryptography uses one key for encryption and a different, but relative, key for decryption. The concept of public-key cryptography is built upon the idea that it might be possible to find a cryptosystem where it is computationally infeasible to determine d_K given e_K . If that is the case, then according to the encryption rule, e_K is a public key, which can be published in the world, and d_K is a secret key, which has to be stored secretly. Two common kinds of asymmetric cryptography are RSA and DSA.

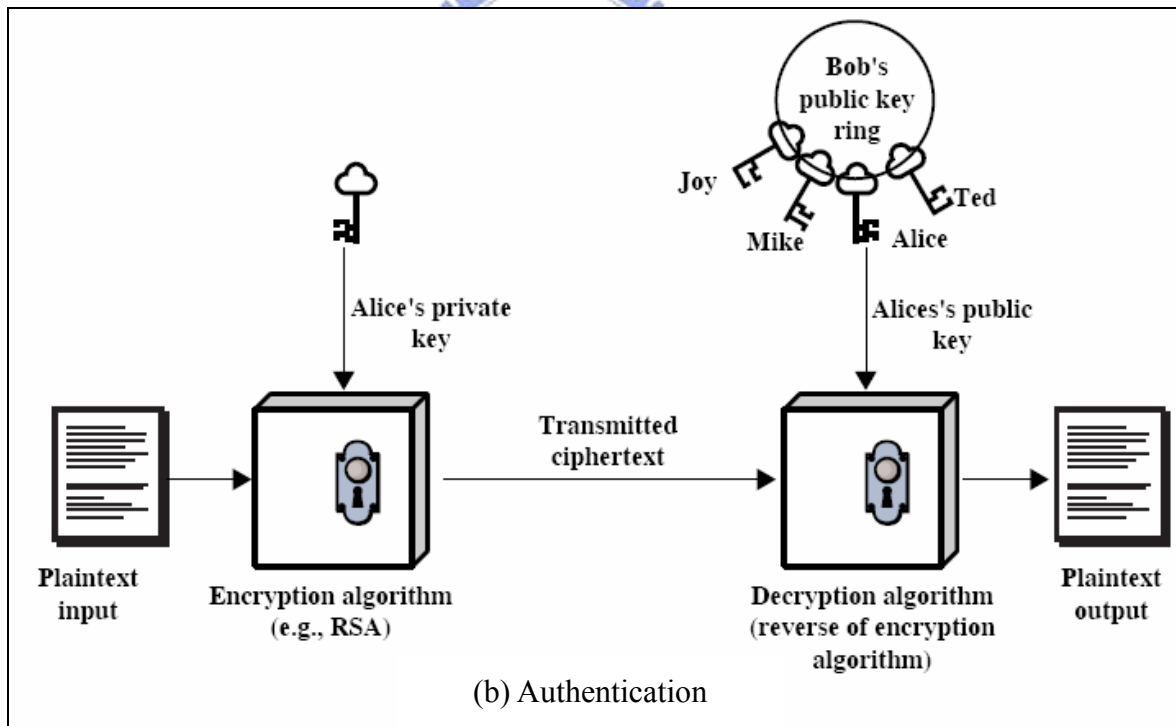
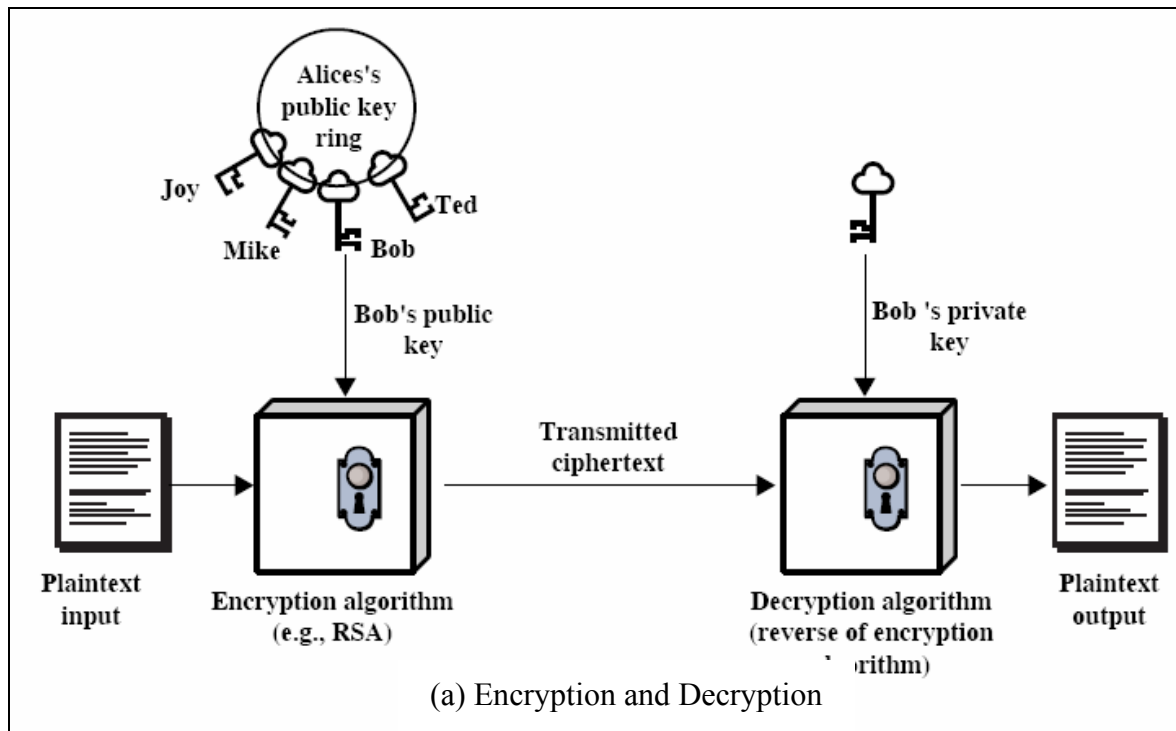


Figure 2.2.2 Asymmetric Cryptography

As Figure 2.2.2(a) shows, when Alice wants to transmit a message to Bob, Alice uses Bob's public key to encrypt the message. Then, instead of the original message, she

transmits the ciphertext to Bob. When Bob receives the ciphertext, he uses his own private key to recover the message. From the example above, it clearly shows that only when we have Bob's private key, then are we capable of recovering the ciphertext back to its original state. Thus, an advantage of the public-key cryptography is that even if other people were to intercept the transmitting ciphertext, they would not be able to recover the message without Bob's private key.

As Figure 2.2.2(b) illustrates, the public-key cryptography can also be used for identity authentication. Alice uses her secret key to encrypt the message that she wants to send to Bob. When Bob receives it, he can use Alice's public key to decrypt the message. Because this message is encrypted by Alice's secret key, only Alice could generate this message. Thus, the entire message itself can also serve as a digital signature. Additionally, the person who sends this encrypted message can be authenticated. This achieves the identity authentication.

Asymmetric cryptography solves the key management and key distribution problems successfully. However, the fact that asymmetric approach involves relatively more complicated algorithms as compared to the symmetric approach, the efficiency in encryption and decryption is then not as good as that of the symmetric approach.

In the current thesis, both symmetric and asymmetric cryptography are used. Specifically, we use the asymmetric method for getting the signatures to verify the identities of the node and of the corresponding public key. Then, when each node has the same symmetric key, we encrypt the transmitting message by using the symmetric algorithm. Therefore, not only can our approach solve the key distribution problem but also achieve high transmission efficiency.

2.3 Public Key Infrastructure

Asymmetrical algorithms, such as RSA and DSA, have undoubtedly revolutionized the

science of cryptography, but these algorithms do not guarantee a carefree crypto life.

Problems with asymmetrical algorithms stem from their practical applications, and these problems can only be avoided by constructing a suitable infrastructure. Problems with asymmetrical algorithms are summarized in the following section [3].

(1) Authenticity of the key: In an asymmetric cryptosystem, how can one tell a public key belongs to whom? In other words, an attacker can easily use a man-in-the-middle attack to cheat both the sender and the receiver, as depicted in Figure 2.3.1. This is a scenario when user A wants to communicate with user B secretly. First, they exchange their public keys with each other. If there is an attacker that tells A that he is B and also tells B that he is A, and in the event that both users A and B were to believe the identity of this “middle-man,” they would then share their public keys with the attacker. Upon receiving A and B’s public keys, the attacker would then send his public key to both users, pretending that this was, in fact, the public key of their communication partners. Then A and B use the attacker’s public key to encrypt the message that they want to send and think that it is secure. What A and B do not realize is that the attacker can, in fact, use his public key to decrypt the message. This is called the man-in-the-middle attack.

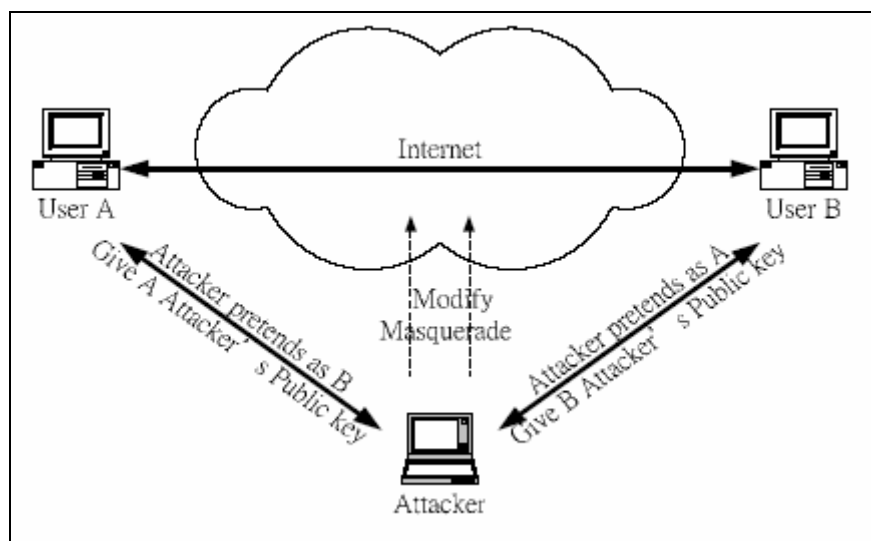


Figure 2.3.1 Man-in-the-middle Attack


(2) Revoking keys: If someone noticed that his private key had been stolen, he

immediately generated a new key pair to replace the old private key. But then how can others know that his old private key has been revoked? In other words, it is not possible to tell from a public key itself whether it has been revoked or not.

(3) Non-repudiation: The purpose of digital signature is to ensure that the individual sending the message is indeed who he claims to be. If one, however, keeps his private key secret, and he simply denies that the key used in the signature was his, no one can challenge him. Since no one can forge his key, the problem then is how to prove that a particular key belongs to whom?

Therefore, a suitable structure must be constructed to address these problems. Such a structure is called a public key infrastructure (PKI).

2.3.1 PKI overview



PKI has more than twenty years of history in development. The concept of the public-key cryptography was first introduced by W. Diffie and M. Hellman in 1976. Two years later, computer scientists proposed the concept of public-key digital certificates. In 1988, the first certificate standard X.509 was developed. In 1993, the first IETF certificate was introduced to the public. In short, the technology of PKI refers to the framework that uses asymmetric approach to generate digital signatures and that provides a highly secure service platform for network transactions. The major goal of PKI is to construct trust relationships among individuals. Establishing trust relationships is fundamental to the implementation of information security. Major components of a PKI are shown in the next illustration, Figure 2.3.2.

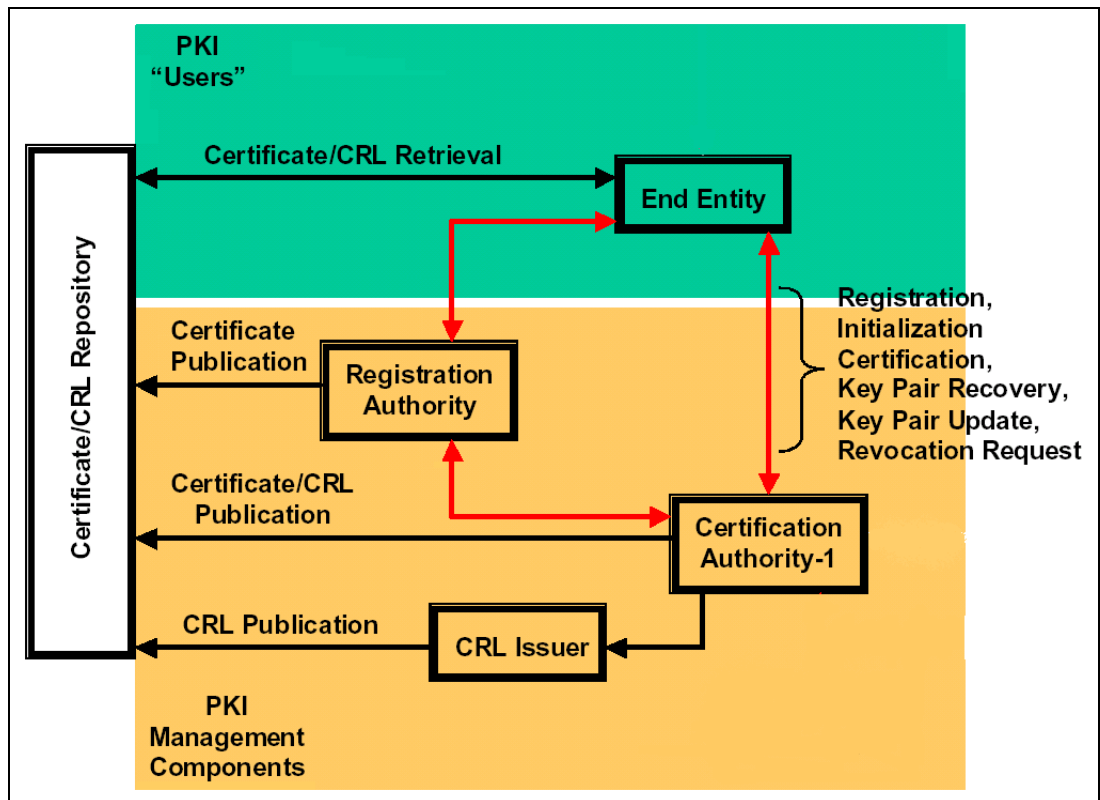


Figure 2.3.2 Components and Process of PKI

End entity is the subject who owns certificates or has owned certifications. For example, a web server or even a mobile phone can process certificates. **Registration authority** (RA) is the administrative center where an end entity can apply for certificates. The RA may be authenticated face to face by the **certification authority** (CA) and may then be trusted to perform face to face authentication for the end entity. The PKI standard does not strictly define the existence of RA. End entity can communicate with the CA directly. However, almost all PKI implementations are provided for an RA and actually do not allow any communication between an end entity and the CA.

The present thesis seeks to implement some roles for the CA. The relationship between the certificate and certificate authority is introduced in the following sub-section.

2.3.2 Certificate and Certification Authority

The certification authority (CA) plays the most important role in a trust center. By definition, a trust center without CA is not possible. The CA is the entity that issues certificates. From the security perspective, CA is a highly crucial component. In particular, the CA's private key has to be stored in a highly secure environment. In order to avoid attacks from the network, the computer running the CA software should, as a rule, not be connected to the internet.

The certificate issued by the CA is the solution to the aforementioned remaining problems of asymmetric algorithms. The content of a certificate may consist of, for instance, issuer, user identity, a corresponding public key, valid period, and issuer's signature. Often, the certificate has signature signed using the CA's private key. As a result, if anyone wants to verify the certificate, he can use CA's public key to do so. He can then trust that the public key inside of the certificate actually belongs to the corresponding user. This solves the authenticity of the key problem. A CA can also revoke a public key very well by signing a revocation list called Certificate Revocation List (CRL). The CRL contains the invalid certificates. One should download the CRL and check a certificate's validity against the CRL before accepting it. Finally, because a CA issues digital certificates to all users, non-repudiation is much easier to guarantee. If one registers with a CA and is officially given a key pair, he can hardly have any dispute over his ownership at a later time.

In summary, there are several characteristics of using a CA:

- (1) Only CA can create, update, and revoke a certificate.
- (2) Every user can verify whether the certificate is issued by the CA.
- (3) Every user can verify the correctness of a certificate.
- (4) Every user can read the content of a certificate to determine its identity and the owner's public key.

2.4 (n, k) Threshold Secret Sharing

Secret sharing means that a secret is divided into many shares, and these shares are distributed to a group of users. As a group, these people collectively share this secret. No one in the group knows or holds the complete secret, and no one can use his own partial share to retrieve the original secret. Only when enough partial shares are combined can we recover the original secret. The (n, k) threshold secret sharing means that we divide a secret into n shares and distribute the shares to n participants. Only when k participants ($k \leq n$) collaboratively combine their own shares, then the secret can be recovered.

(n, k) threshold secret sharing was introduced by Adi Shamir in 1979 [4]. He used a polynomial function to generate a secret into n shares. After that, one can use the Lagrange interpolation method to recover the secret. Below is a step-by-step instruction of this method.

First, suppose that the secret we want to share is S , and there are n participants to share this secret. The identities of participants are denoted as $ID_1 \dots n$. Then there is a dealer, which is trusted by all participants. Specifically, the dealer is responsible to perform the following actions and to distribute the shares to their corresponding participants.

- (1) Give a prime number p , that $p > \max(S, n)$.
- (2) Dealer chooses a polynomial function, $f(x) = S + a_1x + \dots + a_{k-1}x^{k-1}$, that $a_0 = S$ and choose $a_1 \dots a_{k-1}$ from Z_p .
- (3) Compute share secret, $S_i = f(ID_i) \pmod{p}$, for $i = 1 \sim n$.
- (4) Distribute share secret S_i to ID_i .

In order to recover the original secret, Lagrange interpolation must be used. We must have k or more than k , use $f(x)$ to construct $f(0)$, and finally secret S is retrieved.

Lagrange interpolation:

$$f(x) = \sum_{i=1}^k S_i \cdot \prod_{j=1, j \neq i}^k \frac{x - ID_j}{ID_i - ID_j} (x) \pmod{p}$$

2.5 Proactive Secret Sharing

The secret sharing we mentioned above relies on the idea of distribution. We share the secret to many nodes, so if one node is compromised, the attacker still cannot retrieve the secret. The only way that an attacker can obtain the secret is to compromise at least k nodes. If each share is, however, never changed, an attacker may have ample time to compromise k nodes and ultimately retrieve the secret. This probability increases over time. Therefore, some people, such as A. Herzber, proposed proactive secret sharing in 1995[7]. The objective of this method is to decrease the likelihood of the abovementioned situation by using an update function to periodically calculate new share and distribute the new share to each node.

Without disclosing the service private key, proactive secret sharing allows the users to calculate the new shares from the old ones in collaboration. After the update, users remove the old shares and replace them with only the new ones. The fact that different time periods have different update shares and that they are completely independent of the old shares, there is no way to reconstruct the original secret by combining the new and old shares. Furthermore, no one can predict the new share value for each node after each update cycle – that is, it is completely random, not predictable. Thus, if an attacker wants to get the secret, he has to compromise at least k nodes during one point in time. Otherwise, after each update, any information he had previously obtained would clearly become useless. Share refreshing basically relies on the following homomorphic property.

If (s_1, s_2, \dots, s_n) is a (n, k) sharing of secret S and $(s'_1, s'_2, \dots, s'_n)$ is a (n, k) sharing of secret S' , then $(s_1 + s'_1, s_2 + s'_2, \dots, s_n + s'_n)$ is a (n, k) sharing of $S + S'$. If S' is 0, then we get a new (n, k)

sharing of S . Now, let's turn our attention to how it does that.

First, for each node i , we must generate a new polynomial $f^{(i)}(x)$ to correspond with $f(x)$. The constant term of $f^{(i)}(x)$ has to be 0 because we want the secret shared by this polynomial is 0. The method of updating share is shown as follows:

$$f(x) = (S + a_1x + \dots + a_{k-1}x^{k-1}) \bmod p$$

$$f^{(i)}(x) = (b_{i,1}x + \dots + b_{i,k-1}x^{k-1}) \bmod p, b_{i,1}, \dots, b_{i,k-1} \text{ is random number.}$$

$$f'(x) = (f(x) + \sum_{i=1}^k f^{(i)}(x)) = (S + (a_1 + \sum_{i=1}^k b_{i,1})x + \dots + (a_{k-1} + \sum_{i=1}^k b_{i,k-1})x^{k-1}) \bmod p$$

The update share for each node is computed by $f'(ID_j)$, for $j = 1, \dots, k$. Therefore, each participant first generates its own polynomial $f^{(i)}(x)$. Then according to the polynomial, he computes $f^{(i)}(ID_j)$. After that, he sends these results securely to the corresponding node. When a node receives the new share from other participants, he will add to the original share. The result of this addition is the new share.

A verification system must, however, be in place in order to prevent some nodes being compromised. A compromised node may not want to participate in the update process, or it may intentionally send incorrect update shares to other nodes. If other nodes use the incorrect shares to construct their new shares, the secret, which is recovered from the new shares, will then not be consistent with the original one. Hence, verifiable secret sharing is used to prevent this kind of attack. The method is detailed below.

(1) Prior to distributing the secret share to other nodes, the dealer publishes

$g^{a_0}, g^{a_1}, \dots, g^{a_{k-1}}$ that are the witnesses of coefficients of the sharing polynomial.

(2) Each participant then receives its share and verifies it by calculating

$$g^{S_i} = g^S \cdot (g^{a_1})^{ID_i} \cdot \dots \cdot (g^{a_{k-1}})^{ID_i^{k-1}}$$

2.6 Combinatorial Object

We begin this section by first introducing some specific notations and their definitions [16].

Definition 2.6.1 Θ -notation [16]

For a given function $g(n)$, we denoted by $\Theta(g(n))$ the set of functions

$\Theta(g(n)) = \{f(n): \text{there exists positive constants } c_1, c_2 \text{ and } n_0 \text{ such that}$

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\} \quad \square$$

A function $f(n)$ belongs to the set $\Theta(g(n))$ if there exists positive constants c_1 and c_2 such that it can be “sandwiched” between $c_1 g(n)$ and $c_2 g(n)$ for sufficiently large n .

Definition 2.6.2 O -notation [16]

For a given function $g(n)$, we denoted by $O(g(n))$ the set of functions

$O(g(n)) = \{f(n): \text{there exists positive constants } c \text{ and } n_0 \text{ such that}$

$$0 \leq f(n) \leq c g(n) \text{ for all } n \geq n_0\} \quad \square$$

We use O -notation to give an asymptotic upper bound on a function to within a constant factor.

Definition 2.6.3 Ω -notation [16]

For a given function $g(n)$, we denoted by $\Omega(g(n))$ the set of functions

$\Omega(g(n)) = \{f(n): \text{there exists positive constants } c \text{ and } n_0 \text{ such that}$

$$0 \leq c g(n) \leq f(n) \text{ for all } n \geq n_0\} \quad \square$$

We use Ω -notation to provide an asymptotic lower bound on a function to within a constant factor.

2.6.1 Introduction to Perfect Hash Families

Computer scientists have studied perfect hash families (PHF) for more than 15 years. Perfect hash families are basic combinatorial structures, and they have played many important roles in the field of computer science, such as in database management and compiler constructions. Such hash functions should be easily computable, and only a minimal amount of memory would be required. Not until recently that the concept of perfect hash families has been applied to cryptography. For example, it can be seen used in the broadcast encryption schemes, secret sharing, and cover-free families. A perfect hash family can be defined as follows [5]:



Definition 2.6.4 [5]

A perfect hash family, denoted as $\text{PHF}(F; A, B, w)$, should satisfy the following statements:

1. A and B are the finite non-empty set.
2. F is a finite set of hash functions from A to B such that for each $X \subseteq A$ if $|X| = w$, there exists at least one $f \in F$, where $f|_X$ is injective. \square

Note: In many situations, a perfect hash family can also be denoted as $\text{PHF}(F; |A|, |B|, w)$.

According to the above definition, the notation ' $f|_X$ ' is used to denote the restriction to the set X . We say that a function $f : A \rightarrow B$ separates $X \subseteq A$ if f is injective to the set X .

Note: We may also write $f(A, B, w)$ as $f \in F, f: A \rightarrow B, F = \{ f(A, B, w) \mid f \}$.

Let N be the minimum number of functions such that a $\text{PHF}(F; A, B, w)$ would exist.

That is, $N = \min \{ |F| \}$ is the optimal solution [5].

Below is a simple example of a perfect hash family – $\text{PHF}(4; 9, 3, 3)$.

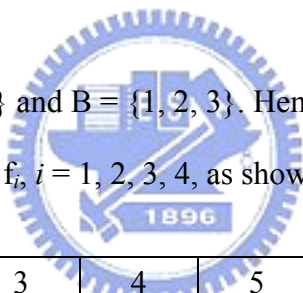
Example 2.6.1

We have a $\text{PHF}(4; 9, 3, 3)$. Consider the matrix:

$$M = \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 \\ 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 \\ 1 & 2 & 3 & 3 & 1 & 2 & 2 & 3 & 1 \\ 1 & 2 & 3 & 2 & 3 & 1 & 3 & 1 & 2 \end{bmatrix}$$

Let $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and $B = \{1, 2, 3\}$. Hence $|A| = 9, |B| = 3$.

Let f be a set of hash functions $f_i, i = 1, 2, 3, 4$, as shown in Table 2.6.1.



X	1	2	3	4	5	6	7	8	9
$f_1(x)$	1	1	1	2	2	2	3	3	3
$f_2(x)$	1	2	3	1	2	3	1	2	3
$f_3(x)$	1	2	3	3	1	2	2	3	1
$f_4(x)$	1	2	3	2	3	1	3	1	2

Table 2.6.1 PHF(4; 9, 3, 3)

From these four functions, we can see that for any subset of $X \subseteq A$ with $|X| = 3$, we have at least one function f_i that separates X . The verification of that F is a $\text{PHF}(4; 9, 3, 3)$ has been shown in [5]. □

In the next section, two propositions of perfect hash families are presented. These two propositions are used in our scheme later.

2.6.2 Propositions of Perfect Hash Families

Two propositions of perfect hash families are discussed here [5][6]. One is the partition characteristic offered by the perfect hash families. The other one is the corresponding matrix.

Definition 2.6.5 w -partition of A

$w \mid |A|$, Π is a partition of A. $P_i \in \Pi$, $\{P_1, P_2, \dots, P_{\frac{|A|}{w}}\}$, $|P_i| = w$.

The order of each subset is w . □

Note: A set $X \subseteq A$ is separated by a partition π of A if the elements of X are in distinct part of π .

Proposition 2.6.6 [5][6]

Suppose that Π is a family of w -partition of A with $|\Pi| = N$. For all sets $X \subseteq A$ with $|X| = w$, X is separated by at least one $\pi \in \Pi$. Then there exists a PHF($N; n, m, w$). Conversely, a PHF($N; n, m, w$) gives rise to such w -partition set Π of A. □

The proof for Proposition 2.6.6 is included in Appendix A. A simple example of Proposition 2.6.6 is given below.

Example 2.6.2

Let $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Consider the PHF(4; 9, 3, 3) we constructed in Example 2.6.1, we can get the following results:

$$\begin{aligned} \pi_1 &= \{\{1, 2, 3\}, \{4, 5, 6\}, \{7, 8, 9\}\}, & \pi_2 &= \{\{1, 4, 7\}, \{2, 5, 8\}, \{3, 6, 9\}\} \\ \pi_3 &= \{\{1, 5, 9\}, \{2, 6, 7\}, \{3, 4, 8\}\}, & \pi_4 &= \{\{1, 6, 8\}, \{2, 4, 9\}, \{3, 5, 7\}\} \end{aligned}$$

Thus, $\Pi = \{\pi_1, \pi_2, \pi_3, \pi_4\}$ is the most desired set of partitions of A .

Conversely, we can find a function family $F = \{f_1, f_2, f_3, f_4\}$, such that $f_i(x)$ is denoted as π_i and for each $x \in A$, labeling the part for each partition π_i according to the given order.

□

Proposition 2.6.7 [5][6]

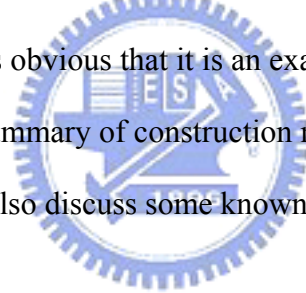
Suppose that there exists a PHF($N; n, m, w$). Then there exists an array M , where size is $N * n$ and which has entries in a set B of size m , such that for any subset X of columns of M with $|X| = w$, there is at least one row of M that separates the subset X of columns of M .

Conversely, such an array gives rise to a PHF($N; n, m, w$).

□

The proof of Proposition 2.6.7 is also included in Appendix A. Referring to the matrix provided in Example 2.6.1, it is obvious that it is an example of Proposition 2.6.7.

Next section provides a summary of construction methods of perfect hash families proposed in other studies. We also discuss some known bounds of $N(n, m, w)$.



2.6.3 Construction methods of Perfect Hash Families

In this section, we are more interested in the behavior of minimum N as a function of n when m and w are fixed. Bounds on N have been studied extensively (For examples, see [5][6][17][18][19]). In particular, in [19], it provides a proof that when m and w are fixed, N is $\Theta(\log n)$. However, this existence is non-constructive. It is also believed that it is difficult to give explicit constructions that are as asymptotically good. Here, we introduce some explicit constructions and point out the bound on N in those constructions. Although these constructions are not as asymptotically good as the one presented in [19], they are quite

reasonable.

There are many kinds of method to construct perfect hash families, such as using combinatorial structures and using algebraic structures. Table 2.6.2 lists the approaches included in the combinatorial and algebra structures.

Combinatorial Structures	Algebra Structures
Design Theory	Special Global Function Field
Error-Correcting Codes	Algebraic Curves
Recursive Constructions	

Table 2.6.2 Construction Methods

In the combinatorial structures, we can use the design theory to construct perfect hash families. There are some set systems, such as the balanced incomplete block design (BIBD) and the separating resolvable block design (SRBD). The detail of this construction was introduced in [17]. According to their inference and proof, in the situation when m and w are fixed, the bound of N is $\Omega(n)$. Although these methods give simple constructions, they are limited in the sense that they cannot be applied to obtaining a PHF with an arbitrary $m \geq w$. In other words, they cannot obtain a PHF in which m is $O(w)$. In addition, in a construction of perfect hash families using Error-Correcting Codes, the bound of N is $O(n)$. The restriction of this method is the same as using the design theory to construct it. It also cannot construct a PHF in which m is $O(w)$. Finally, in [17], the authors proposed two kinds of recursive construction. First, they used an already existing PHF together with a (n, k, λ) - difference matrix to obtain another PHF with larger N and n . In this construction, the bound of N is $O((\log n)^{\log \binom{w}{2} + 1})$. Second, the authors used three already existing PHFs and combine them into a new PHF with larger N and n . The bound value of N in the second method is about the same as the first one, but the second method has a slightly larger constant term.

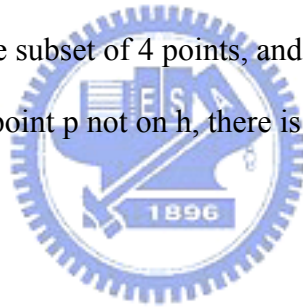
In [18], the authors proposed the method of using algebra structures to construct perfect

hash families. Specifically, they used an algebraic curve to construct a PHF. In this method, the bound of N is $O(\log n)$. Details of this kind of construction are not included in this text since algebra structures are not used in this thesis. (For more related information, see [18].)

In this thesis, the PHF is constructed by the affine plane and resolvable BIBD. Before we introduce the construction mechanism, we first give some definitions for affine plane and resolvable BIBD.

An affine plane is a $PBD(P, B)$ with some specific properties[20]. Before we state the corresponding properties, $PBD(P, B)$ is introduced first. A pairwise balanced design, referred to as the PBD, is an ordered pair (P, B) . P is a finite set of symbols, and B is a collection of subsets of P called blocks, such that each pair of distinct elements of P occurs together in exactly one block of B . The properties of an affine plane are summarized below.

- (1) P contains at least one subset of 4 points, and no 3 of which are collinear.
- (2) Given a line h and a point p not on h , there is exactly one line of B containing p , which is parallel to h .



Example 2.6.3

Affine plane.

$$P = \{1, 2, 3, 4\}$$

$$B = \{ \{1, 2\} \quad \{1, 3\} \quad \{1, 4\}$$

$$\{3, 4\} \quad \{2, 4\} \quad \{2, 3\} \} \quad \square$$

In an affine plane (P, B) , the number of points in each block is called the order of the affine plane.

Definition 2.6.8 *k*-power set of X

P is the power set of X.

A is a *k*-power set of X if $A \subseteq P$ and for each $x \in A$ $|x| = k$.

We have $\binom{|X|}{k}$ *k*-power sets of X. □

Definition 2.6.9 [5]

X is a non-empty set of points, and A is a subset of the *k*-power set of X called blocks.

Let *v*, *k*, λ be positive integers such that $v \geq k \geq 2$. A (v, b, r, k, λ) – balanced incomplete block design (denoted as (v, b, r, k, λ) – BIBD) is a set system (X, A) such that the following properties are satisfied:

1. $|X| = v$,
2. Every point occurs in *r* blocks, and
3. Every pair of points occurs in exactly λ blocks. □



For simplicity, in the following examples, we write blocks in the form abc, rather than {a, b, c}.

Example 2.6.4

$A(10, 15, 6, 4, 2)$ – BIBD.

$X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, and

$A = \{0123, 0145, 0246, 0378, 0579, 0689, 1278, 1369, 1479, 1568, 2359, 2489, 2567, 3458, 3467\}$.

□

Theorem 2.6.10. [5][21]

1. A (v, b, r, k, λ) – BIBD follows from elementary counting that $vr = bk$ and $\lambda(v - 1) = r(k - 1)$. □

The proof of Theorem 2.6.10 is included in Appendix A.

A parallel class in (X, A) is a set of blocks that forms a partition of the point set X . A BIBD is resolvable if A can be partitioned into r parallel classes, and each of which consists of v/k disjoint blocks. Obviously, a BIBD can have a parallel class only if $v \equiv 0 \pmod k$.

Example 2.6.5 A resolvable $(6, 15, 5, 2, 1)$ – BIBD.

Let $X = \{0, 1, 2, 3, 4, 5\}$, and $r = 5$. Hence there are 5 parallel classes, and each consists of 3 blocks.

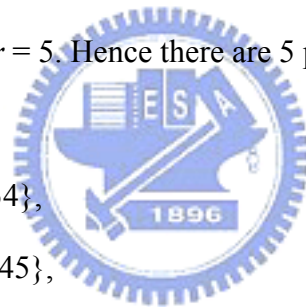
So parallel classes = $\{01, 25, 34\}$,

$\{02, 13, 45\}$,

$\{03, 24, 15\}$,

$\{04, 35, 12\}$,

$\{05, 14, 23\}$ □



It is well-known that an affine plane of order q is an $(q^2, q(q+1), q+1, q, 1)$ – BIBD. It is also a resolvable BIBD. Thus, the following theorem can be derived: For any prime power q , there exists an affine plane of order q . That is, there exists a $(q^2, q(q+1), q+1, q, 1)$ – BIBD.

Theorem 2.6.11. [5][18]

If there exists a resolvable $(v, b, r, k, \lambda) - \text{BIBD}$ with $r > \lambda \binom{w}{2}$, then there exists a $\text{PHF}(r; v, v/k, w)$. □

The above theory is derived and obtained from [5][18]. The proof is stated in Appendix A. Based on this theory and the above description, we then can derive the following corollary.

Corollary 2.6.12 [5]

Let w be an integer such that $w \geq 2$. Suppose q is a prime power and $q+1 > \binom{w}{2}$. Then there exists a $\text{PHF}(q+1; q^2, q, w)$. □

Therefore, we can use an affine plane to construct a PHF.

In this thesis, we construct the perfect hash families according to Corollary 2.6.12. The detail of our construction is described in the next chapter. By observations, we find that formats of the BIBD and the PHF, which are constructed from an affine plane, are determined by only one parameter – prime power q . Thus, we give a special name for these kinds of BIBDs and PHFs – namely, $(q, 1) - \text{BIBD}$ and $(q, w) - \text{PHF}$.

2.7 Fundamentals of Mobile Ad Hoc Network

In the last few years, computer scientists have shown growing interest in studying mobile ad hoc networks as they have tremendous military and commercial potential [22]. Security related issues in mobile ad hoc networks are also an important topic of research. Below is a brief introduction to the mobile ad hoc network [23].

2.7.1 Mobile Ad Hoc Network

Mobile ad hoc network for short is called MANET. It is a non-infrastructure network. MANET is consisted of a group of autonomous mobile nodes. Each node has the function of a router. They are able to communicate with each other without any support of wired infrastructure. As long as they stay in the communication scope of each other, they can talk to each other by using the wireless link. Since the nodes have mobility, the network topology may change rapidly and unpredictably. Also, the network is decentralized; all network activities, which include discovering the topology and delivering messages, must be executed by the nodes themselves. Nowadays, a mobile node can be a notebook, a PDA, or any other kinds of wireless device with mobility. Next figure illustrates a mobile ad hoc network constructed by many kinds of mobile node.

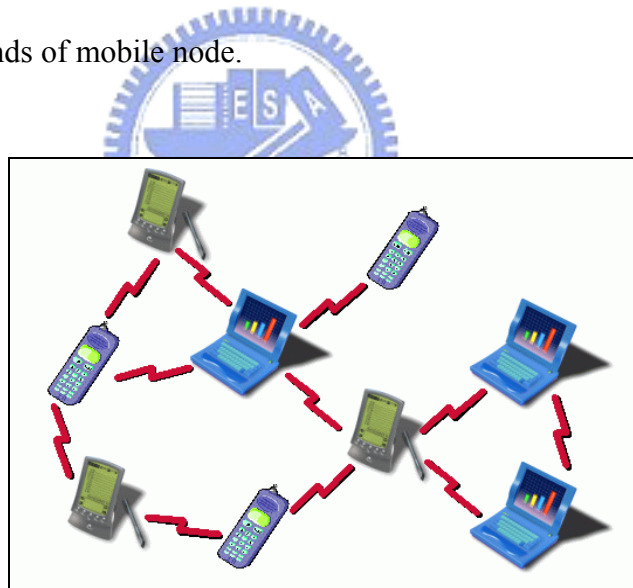


Figure 2.7.1 Mobile Ad Hoc Network

2.7.2 Characteristics of Mobile Ad Hoc Network

According to IETF RFC 2501[24], the characteristics of MANET include several parts, and they are outlined below.

1. Dynamic topologies: Because the nodes in MANET have high mobility, the topology

may change randomly and rapidly at unpredictable times.

2. Bandwidth-constrained, variable capacity links: Compare to the wired network environment, MANET is resource-constrained in bandwidth and link capacity. In addition, the realized throughput of wireless communications is often much less than a radio's maximum transmission rate.

3. Energy-constrained operation: The power supply for some nodes in MANET is batteries or other exhaustible energy. Therefore, the consideration of limited battery power and life is also necessary.

4. Limited physical security: Wireless links are generally prone to more physical security threats than wired links. MANET suffers from, for example, passive eavesdropping, active impersonation, and denial-of-service attacks. As a benefit, though, the decentralized mechanisms in MANET have higher security.

2.7.3 Security challenges of Mobile Ad Hoc

Network

Due to the noticeable characteristics of mobile ad hoc networks, achieving the requirements of security mentioned in the earlier section can be rather challenging. First, compared with wired links, wireless links are generally more prone to link attacks because all the data are transmitted in the air. As a result, it is relatively easier to perform eavesdropping, impersonation, message replay and message distortion. The simplest way to protect the transmitting data is to encrypt it before sending it out. Certainly, in this way, we have to bear the overheads from encryption. Second, since each node has the mobility, it may roam to a dangerous environment. Some nodes may therefore be compromised. Thus, we should not only pay attention to the malicious attacks from the outside, but also, equally important,

watch out for the wrong information from the inside compromised nodes. The trust relationship among nodes may change very often; for example, some nodes may have been detected as compromised nodes, so we have to authenticate the neighbor nodes periodically. Third, due to the high mobility, some roles that are responsible for authentication, such as CA, cannot be the central entities. In order to improve the survivability, the authenticator must use a distribution structure. Fourth, because of a rapid changing topology, a mobile node may only be able to perform effectively and have timely communication with its local neighbors but not with remote entities. For example, routing protocols may fail to establish robust communication over multi-hop paths. Thus, it is imperative to localize the security service. Finally, an ad hoc network may consist of hundreds or even thousands of nodes. Therefore, the scalability and flexibility of security mechanisms are crucial properties.

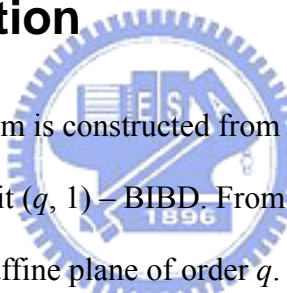


Chapter 3 System Architecture

3.1 Concept

In this thesis, we distribute the CA's functionality to many nodes in the network. In addition to using Shamir's threshold secret sharing scheme, we also use perfect hash families' properties to distribute the trust among a group of nodes. In this chapter, we discuss the concept behind the design of our protocol.

3.1.1 Construction



Initially, our network system is constructed from a special form of BIBD: $(q^2, q(q+1), q+1, q, 1) - \text{BIBD}$, and we call it $(q, 1) - \text{BIBD}$. From section 2.6.3, we know that for any prime power q , there exists an affine plane of order q . Furthermore, an affine plane of order q can construct a $(q, 1) - \text{BIBD}$. Because $q^2 \equiv 0 \pmod{q}$, it is also a resolvable BIBD. Let w be an integer such that $w \geq 2$. Suppose q is a prime power and $q + 1 > \binom{w}{2}$. Then, based on Corollary 2.6.12, there exists a $\text{PHF}(q + 1; q^2, q, w)$, and we call it $(q, w) - \text{PHF}$. When we want to construct the system, we therefore first consider the security parameter w that we want to achieve and the system secret S . After both variables are determined, we then get a minimal random prime power q such that $q + 1 > \binom{w}{2}$. Finally, we use this prime power q to create a $(q, w) - \text{PHF}$. The following two figures show the construction algorithm of this PHF. The program of the construction is appended in Appendix B. Figure 3.1.1 provides the procedure of finding the minimal prime power q that satisfies the requirement $q + 1 > \binom{w}{2}$.

That is $q = \min\{x \mid x \text{ is a prime power and } x+1 > \binom{w}{2}\}$. Figure 3.1.2 then illustrates how the appropriate (q, w) – PHF is generated [20].

```
// Finding the minimal prime power q
// Input: the security parameter w
// Output: prime power q

Finding_q( w )
1.   q ←  $\binom{w}{2}$ 
2.   while q exists
3.       if q is a prime power
4.           then return q
5.       break
6.       else q ← q + 1
```

Figure 3.1.1 Finding the minimal prime power q

```
// Construction algorithm of PHF
// Input: the prime power q
// Output: A (q, w) - PHF

Construct_PHF(q)
1.   Use the prime power q to construct a finite field
2.   Derive an affine plane of order q from the finite field
3.   Map the affine plane to a (q, 1) - BIBD
4.   Construct a (q, w) - PHF from the (q, 1) - BIBD
5.   return (q, w) - PHF
```

Figure 3.1.2 Construction Algorithm of PHF

The meaning of every parameter in the $\text{PHF}(q + 1; q^2, q, w)$ is explained as follows: q^2 means the number of nodes owning the secret shares in our system. We call this kind of nodes as the server nodes. $(q + 1)$ represents the number of secret shares each server node should hold. Besides, it also implies that this PHF has $(q + 1)$ hash functions, and we have to create $(q + 1)$ polynomial functions with degree $(w - 1)$ for secret sharing. Each polynomial function has the same secret S . Also, a set of secret shares can be combined for every w server nodes to reconstruct the system secret S and use S as the secret key to sign a certification collaboratively. In other words, w is the minimum number of server nodes required to retrieve the original S . Finally, q refers to the number of disjoint sets in each partition when we partition q^2 sever nodes. That is, each partition has q disjoint sets. Furthermore, it also implies the number of secret shares that each polynomial function has to generate.

When we divide the system secret S into several secret shares, we have to send these secret shares to the sever nodes. The coefficient of the corresponding hash functions determines which serve node gets which secret share.

Take the $\text{PHF}(4; 9, 3, 3)$ we mentioned above for example. From this PHF, we know that the system has nine sever nodes. We denote them as from Ser_1 to Ser_9 . Then we would generate four polynomial functions with degree 2, and each would individually generate three secret shares. Each sever node would then have four secret shares from different polynomial functions. Any three randomly chosen server nodes could use a set of secret shares to reconstruct the system secret. We distribute the secret shares to the nine server nodes according to the four hash functions, which is summarized below in Table 3.1.1.

X	1	2	3	4	5	6	7	8	9
$F_1(x)$	1	1	1	2	2	2	3	3	3
$F_2(x)$	1	2	3	1	2	3	1	2	3
$F_3(x)$	1	2	3	3	1	2	2	3	1
$F_4(x)$	1	2	3	2	3	1	3	1	2

Table 3.1.1 A PHF(4; 9, 3, 3)

Suppose we denote the four polynomial functions as P_1 , P_2 , P_3 and P_4 . Furthermore, we mark the secret shares generated from P_1 as S_{11} , S_{12} , S_{13} , those from P_2 as S_{21} , S_{22} , S_{23} , and so on and so forth.

Assuming the distribution of secret shares generated from P_1 corresponds to hash function $f_1(x)$, and those generated from P_2 , P_3 and P_4 are based on $f_2(x)$, $f_3(x)$ and $f_4(x)$ respectively to do the distribution. Table 3.1.1 shows that for the hash function $f_1(x)$, Ser_1 , Ser_2 and Ser_3 map to the same coefficient – that is, 1. For this reason, these three server nodes get the same secret share S_{11} from P_1 . Similarly, Ser_4 , Ser_5 and Ser_6 map to the same coefficient of $f_1(x)$, which is 2, so they get the same secret share S_{12} from P_1 . Lastly, the secret share S_{13} from P_1 is sent to Ser_7 , Ser_8 and Ser_9 because these server nodes have the same coefficient. Likewise, the secret shares generated from P_2 are distributed based on $f_2(x)$. Thus, S_{21} is sent to Ser_1 , Ser_4 , and Ser_7 ; S_{22} is sent to Ser_2 , Ser_5 , and Ser_8 ; and S_{23} is sent to Ser_3 , Ser_6 and Ser_9 . As for all other secret shares from the last two polynomial functions, they are distributed to the remaining nine server nodes based on $f_3(x)$ and $f_4(x)$. Finally, the distribution of all secret shares is completed.

In the following section, we describe how w server nodes can recover the system secret. Since the degree of the polynomial function is $w-1$, we know that we can retrieve the constant, or say the system secret, of that function by getting w different secret shares from that polynomial function. Each server node has $q+1$ secret shares, so the question is how to determine which secret share to use for recovering the secret. The answer is that according to

the definition of a PHF, when we randomly select a subset of order w , there must exist at least one hash function that makes this subset injective. We, therefore, take the secret shares that corresponded with this specific hash function to reconstruct the system secret. Since this hash function makes the subset of order w injective, the w server nodes map to w coefficients accordingly. In other words, all w server nodes get different secret shares from the polynomial function that is mapped to this hash function. Therefore, we derive the system secret by using Lagrange interpolation method with these secret shares. Here we also take the above example to illustrate. If we randomly select three server nodes – for example Ser_1 , Ser_5 , and Ser_8 – then we can find at least one hash function that creates one-to-one mapping. That is, they have different coefficients. The above table demonstrates that $f_1(x)$ satisfies this requirement. As a result, these three server nodes can use the secret shares that they got from polynomial function P_1 to recover the system secret. More precisely, they use S_{11} , S_{12} and S_{13} respectively to complete the reconstruction.

In our system, we do not have to ensure that there exists connections among all q^2 server nodes. Specifically, we do not have to promise a full connection among server nodes. Full connection is, in fact, not a reasonable requirement in MANET environment. Instead, we only have to guarantee that each server node has connections with at least the other $(w-1)$ server nodes. In this way, we can be certain that there is enough server nodes to participate in recovering the system secret. So, referring back to the above example, we only have to maintain the requirement that each server node has links with at least two other server nodes. In sum, our system can handle the dynamic characteristics of a MANET well.

3.1.2 Partition

Based on the characteristics that we described in section 2.6.2 and in the last section, we know that each partition generated from $PHF(q + 1; q^2, q, w)$ consists of q disjoint sets. In our

system, we would appoint one server node in each disjoint set as a partition header, or PH for short. From the example PHF(4; 9, 3, 3), we can discover that each hash function can generate a corresponding partition. Take $f_1(x)$ for example, the partition made by $f_1(x)$ is $\{\{1, 2, 3\}, \{4, 5, 6\}, \{7, 8, 9\}\}$. So, among these three disjoint sets, we would appoint three server nodes as partition headers for each disjoint set individually. For instance, we may assign Ser_1 , Ser_5 and Ser_8 to be the partition headers. As for $f_2(x)$, its partition is $\{\{1, 4, 7\}, \{2, 5, 8\}, \{3, 6, 9\}\}$. Similarly, we would also appoint three PHs for these blocks, for example Ser_4 , Ser_2 , and Ser_9 . For all other partitions, we would also do the same. Details on how to appoint the partition headers are described in the latter section. Each server node can be the PH in different blocks simultaneously. When the server nodes do the secret share update procedure, these PHs can make it more efficient; the detail is also presented in the latter section. Furthermore, these PHs also have to maintain connection status with all of the server nodes.

Every PH has the responsibility to maintain the connection status. The way a PH maintains the connection is to periodically broadcast a packet in the block in which it belongs to. At the same time, the PH would also set up a timer. If the PH does not receive the reply packet from server node Ser_i before the timer expires, we would say that Ser_i has left the block. In this case, the PH must find another node that has passed the authentication procedure and holds a valid certification. The PH would then replace the Ser_i by appointing this new node as a server node and deliver the common secret share of this block to him. In the next periodic broadcast packet, the PH would announce the new location of that server node to other server nodes. In order to prevent the adversaries from getting the sensitive information such as location information, all broadcast packets delivered from the PH can be encrypted using the common secret share of that block. This concept is called the intra-block security and is introduced in the following section.

3.1.3 Conceptual Building Blocks

The architectural concept behind our system is that the network-wide security is based on the distributed certification infrastructure in the entire network. By public key cryptography, the distributed certification infrastructure forms a basis for a secure end-to-end communication. Besides, the security of communication within the disjoint set is provided by symmetric encryption. Therefore, two conceptual building blocks can be sketched in our system.

1. Network-wide security: Network-wide security is the security concept that uses public key cryptography to ensure confidentiality, integrity, and authentication. In this kind of network, each node holds a self-generated key pair. The key pair is used for providing end-to-end security between arbitrary nodes. In the ad hoc network, the public keys are distributed by using the certification issued by a trusted CA. Thus, each node needs to apply to the CA to obtain the certification before distributing its public key. Contrary to the traditional PKI in a fixed network, the function of the CA in our system is distributed. The CA's functionality is delivered by a group of server nodes in the network. The system secret, also representing the CA's private key, is distributed by server nodes. This concept has two advantages: First, service availability is enhanced, since certification can be issued even if some server nodes cannot be reached. Second, this kind of infrastructure can be more resistant to attacks. Specifically, it can tolerate some server nodes being compromised without leaking the system secret.

2. Intra-block security: The characteristic of a PHF suggests that server nodes can be divided into many partitions, and each partition consists of many disjoint blocks. Based on this description, we know that the server nodes in the same disjoint block share a property. That is, they all map to the same coefficient of one hash function. It also means that they have a common secret share. Moreover, we know as a fact that except for the ones belong to this

block, no other server nodes own this particular secret share. Therefore, when the server nodes in the same disjoint block want to communicate with one another, they can use their common secret share as a secret key and perform the symmetric cryptography. In this way, we can lighten the computational overhead for securing communication among server nodes. The intra-block communication is primarily used in the process of updating secret shares.

3.2 System Assumption

3.2.1 Intrusion model

In this section, we briefly discuss what kind of intrusion model that our system can resist. In the worst-case scenario, if a network entity has been compromised, the attacker would be able to get all the information from that entity, regardless whether it is public or private information. The intrusion attacker then has the power to tamper, impersonate, or even delete any information he has obtained. However, in order to achieve authentication, we ought to assume that there is always something that cannot be duplicated or impersonated. Otherwise, as long as a network entity was compromised, we have no way to recognize whether he has been intruded or not. More generally, we would not be able to differentiate compromised nodes from normal nodes. In other words, our system needs to set limitations to restrict an attacker's ability to threaten the network. An intruder with infinite power is, in fact, meaningless because security systems would fail then.

Let's consider some realistic intrusion models that we can resist in our system.

1. An intruder cannot compromise or control w or even more server nodes within an update interval. Our assumption of having the ability to resist these kinds of adversaries can be achieved by using the share update technology.

2. The user's identity cannot be copied or forged by intruders. Hence, if a node was compromised, the other nodes could, in theory, discover this situation by verifying the identity.

3.2.2 Trusted dealer

The dealer is required only once when the system is initialized. Once the security parameter w has been decided, our system then needs a trusted dealer to construct the corresponding perfect hash family. Additionally, the trusted dealer has the responsibility to generate the system key pair, the corresponding polynomial functions, and the corresponding secret shares. The system key pair includes the CA's public key and the secret key or called system public key and secret key. The dealer generates the polynomial functions based on the system secret key. Based on the polynomial functions, he then generates the corresponding secret shares. Finally, the trusted dealer would distribute the secret shares to the corresponding server nodes in a secure way. The presence of a trusted dealer is required as currently there is no other known method for efficiently generating such key shares and securely distributing them to the server nodes during the initial period in an asynchronous distributed system [25].

3.3 Details and Protocols

3.3.1 System initialization

The system initialization stage requires an off-line trusted dealer. After setting up the security parameter, this off-line trusted dealer would generate everything needed for the system, including an appropriate balanced incomplete block design and a perfect hash family, system key-pair, corresponding number of polynomial functions, and secret shares. The

system key-pair consists of system public key and the secret key. The system secret key is further divided into many secret shares depending upon the polynomial functions. The system public key is then distributed within the entire network. Finally, using the aforementioned mechanism, the trusted dealer distributes the secret shares to the server nodes securely and completes the system initialization procedure.

Figure 3.3.1 shows the construction algorithm of the entire network. This algorithm relies on the BIBD blocks for distributing secret shares. In fact, it is identical with the distribution method that we discussed earlier.

```

// Construction algorithm of the whole network
// input: PHF( q + 1, q2, q, w ), q(q+1) BIBD blocks, network secret S
// We denoted the BIBD blocks as B1, B2, ..., Bq(q+1).

Construct_Network( PHF( q + 1, q2, q, w ), B1, B2, ..., Bq(q+1), S )
1.   create q2 server nodes denoted as Ser1, ..., Serq2
2.   create q + 1 polynomial functions of secret S denoted as P1, ..., Pq+1
3.
4.   for i = 1 to q+1
5.       do generate q secret shares from Pi denoted as Wi1, ..., Wiq
6.       for j = 1 to q
7.           Si[j] ← Wij;
8.
9.   for i = 1, j = 1 to q, k = 1 to q(q+1)
10      do
11          if k mod (q+1) ≡ 0
12              i ← i + 1
13              j ← 1
14
15          each point in the block Bk gets the secret share Si[j]
16      return Bk

```

Figure 3.3.1 Construction Algorithm of the whole Network

3.3.2 Certification service

Our certification service is based on the (n, k) threshold secret sharing. For convenience, regular nodes in this text refer to all nodes with the only exception of server nodes. When participating in the network, each regular node would hold a self-generated key pair. Before the regular nodes communicate with one another, each of them has to get a certification to verify his identity and his public key. Certification is obtained by asking the server nodes to do the authentication. Because of the threshold mechanism, a regular node is required to pass the authentication from at least w server nodes individually, and he then can obtain the certification.

In [9][10], Kong et al. use the RSA scheme to provide the certificate services. In our thesis, we refer to [12] to employ our certificate services. In [12], Zhu et al. proposed two schemes to provide the certificate services. For the threshold value w , one is assigning certificates based on $2w-1$ nodes, another is based on w nodes. In the first one, they use $2w-1$ nodes to achieve verifiable certificate services and ensure the certificate process can be finished within one round. In the second one, only w nodes sign the certificate cooperatively, but it may take more than one round to assign the certificate.

A regular node only needs to send a request to one of the nearest server nodes. The request packet includes this regular node's identity and its public key for doing the authentication. Due to the characteristic of $(q, 1) - \text{BIBD}$, we know each pair of elements occurs in exactly one blocks. On the other hand, in the $(q, w) - \text{PHF}$, the server nodes in the same block of partition P_i must in the distinct blocks of other partitions. Each server node in $(q, w) - \text{PHF}$ would belong to q blocks. Thus, once the server node receives the request packet, he will choose a block and then forward the original packet to his block members. Additionally, he also provides the identities for all q server nodes. In our system, if $q > 2w-1$, we can satisfy the first scheme in [12]. Therefore, these q server nodes would each do the

authentication to this regular node individually. After the regular node passes the authentication of server nodes successfully, the server nodes would pass their secret shares in the form of partial certifications g^{s_i} to him. In other case, if $q < 2w-1$, we then can use the second scheme in [12]. A partial group of w server nodes would cooperate to assign the regular node a certificate. Although the second may need more than one round, the simulation results in Kong et al. show that more than 96% of certificate services can be finish within two rounds. Thus, it would not take a long time to complete the certificate service.

We know that among these w server nodes, each has a set of secret shares. The question then is how could they know which secret share should be passed. The answer is based on the identities they received from the request packet. From these identities, they can know which specific hash function made these w server nodes one-to-one mapping. As a result, they know which secret shares they should pass. After the regular node collects w partial certificates, he could combine the partial certificates altogether to form a valid certificate. A valid certificate consists of information on the relationship between the regular node and its public key and the expiration time. After that, the regular node can use this certification to ensure the network-wide security. The other participators in the network can use the system public key to verify this certification. If the certification is valid, they then can trust this regular node. It is important to note that throughout the entire certification service procedure, the CA's private key is never disclosed.

The following two figures demonstrate the certification service procedure. Figure 3.3.2 illustrates the process in which a regular node requests a certification from a nearest server node. Figure 3.3.3 shows that when a regular node passes the authentication, the server nodes would each send back its partial certification to him.

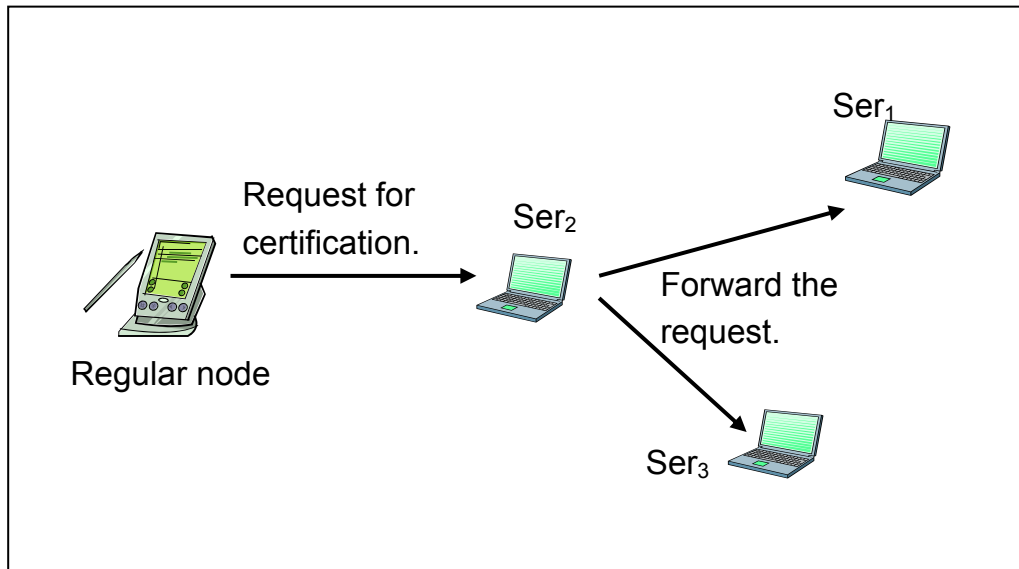


Figure 3.3.2 Request for Certification

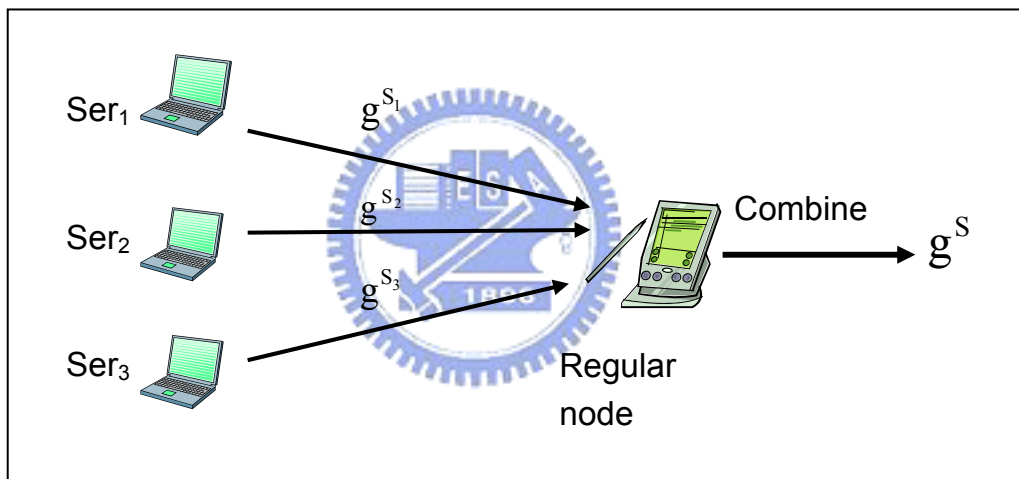


Figure 3.3.3 Certification Service

3.3.3 Updating the Secret Shares

In addition to the threshold secret sharing, our mechanism also involves proactive secret sharing. In order to resist the mobile adversaries, we would periodically update the secret shares in each server node. The proactive secret sharing scheme computes new threshold secret shares for every server node without changing the original secret, and these new secret shares are completely independent of the old ones. No adversary can compute or derive the new secret shares based on the previous ones. The server nodes then use their new secret

shares to sign the partial certifications, but others still use the same public key to verify it. Unlike the traditional proactive secret sharing, in our secret share update procedure, the new partition headers would be selected first. Moreover, from the characteristics of perfect hash families, we know that there is more than one polynomial function. Each polynomial function can obtain a set of secret shares. Thus, each polynomial function can perform the secret share update procedure to refresh its shares.

```

// Change the Partition Headers for PHF( $q + 1; q^2, q, w$ )
// input:  $q$  disjoint set blocks of a partition  $P$ ,  $q(q+1)$  BIBD blocks
// We denoted the disjoint set blocks of partition  $P$  as  $P_1, P_2, \dots, P_q$ .
// We denoted the BIBD blocks as  $B_1, B_2, \dots, B_{q(q+1)}$ .

ChPH( $P_1, P_2, \dots, P_q, B_1, B_2, \dots, B_{q(q+1)}$ )
1.    $c \leftarrow 0$ 
2.   for  $i \leftarrow 1$  to  $q(q+1)$ 
3.       for  $j \leftarrow 1$  to  $q$ 
4.           do if  $B_i \cap P_j \neq \emptyset$ 
5.               then  $c \leftarrow c + 1$ 
6.
7.       if  $c = q$ 
8.           then the new partition header of  $P$  is the points in  $B_i$ 
9.               return  $B_i$ 
10.          break
11.
12.   printf("False! We can not find the new partition headers.\n")
13.   printf("The network should be re-built again.\n")

```

Figure 3.3.4 Algorithm of Changing the Partition Header

Figure 3.3.4 shows the algorithm of changing the partition header of partition P . As we know, each hash function in $(q, w) - \text{PHF}$ can generate a partition for the q^2 server nodes, and this partition is consisted of q disjoint blocks. Each disjoint block has, indeed, q server nodes.

For an arbitrary partition P , if we want to update the secret shares that are distributed based on the coefficient of this hash function, we should choose the partition headers for the corresponding polynomial function. That is, a server node that is a PH from each disjoint block of P should be selected. The number of PHs should be the same as the value of q . This PH selection method is designed based on the relationship between a $(q, 1) - \text{BIBD}$ and a $(q, w) - \text{PHF}$. By observations, we would discover that the existing blocks in the BIBD are the disjoint blocks consisted in all partitions. Furthermore, for partition P , the server nodes in other disjoint blocks, which are derived from the other partitions, are exactly distributed in different blocks of P . Therefore, we appoint the PHs for P by selecting other partitions' disjoint blocks. Thus, if we can successfully choose a disjoint block, the q server nodes in that block would be the PHs of P . PHs should be re-selected for each time the update procedure is performed. This way, every node has the opportunity to be the PH. All server nodes, therefore, share the overhead of being a PH. Furthermore, our system has much more flexibilities to handle the dynamic network environment.

After selecting the PHs for partition P , these PHs would do the secret share update procedure. From section 2.5, we know that secret share refreshing basically relies on the following homomorphic property.

If (s_1, s_2, \dots, s_n) is a (n, k) sharing of secret S and $(s'_1, s'_2, \dots, s'_n)$ is a (n, k) sharing of secret S' , then $(s_1 + s'_1, s_2 + s'_2, \dots, s_n + s'_n)$ is a (n, k) sharing of $S + S'$. If S' is 0, then we get a new (n, k) sharing of S . Thus, each PH would complete the following procedure:

- (1) Each PH_i generates a polynomial function $f^{(i)}(x)$ with secret 0.
- (2) Then, PH_i uses other PHs' identities to generate the new partial secret shares S_{ij} .
- (3) PH_i sends the new partial secret share S_{ij} to corresponding PH_j .
- (4) When PH_j receives all the new partial secret shares from other PHs, he could create

his new secret share by $S'_j = S_j + \sum_{i=1}^q S_{ij}$.

- (5) Finally, after PH_j creates his new secret share, he sends the new shares to the same partition block member securely.

Stage five is similar to the intra-block communication that we introduced in 3.1.3. The PH has the responsibility to securely deliver the new secret share to the server nodes in the same block. Figure 3.3.5 provides a figural illustration of this concept—namely, the share refreshing.

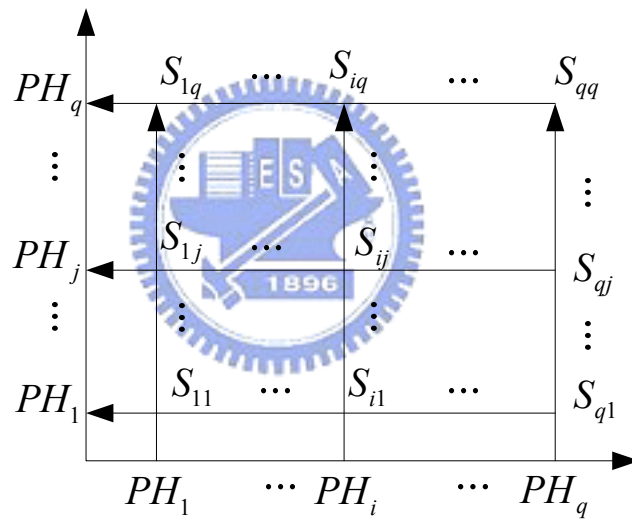


Figure 3.3.5 Share Refreshing

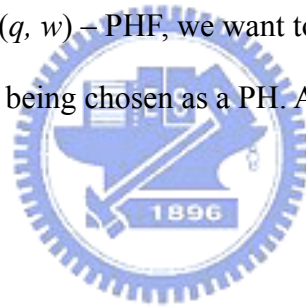
There are many advantages with our secret share update procedure. With $PHF(q + 1; q^2, q, w)$, there are $(q + 1)$ polynomial functions that should be required to do the secret share update, and they all map to the same system secret. Thus, due to the high mobility, we design a system that allows some of the polynomial functions for not taking a part in the secret share update procedure, but the system can remain operational. In fact, as long as there is one polynomial function that can perform the secret share update, our network system can work

well and the server nodes also can refresh one of their secret shares successfully. This concept enhances the effectiveness of proactive secret sharing. Moreover, different polynomial functions can do the update procedure in different time slots. Our system also does not require all q^2 server nodes to do the update altogether; instead, only the PHs of that polynomial function are required to complete the update. In this way, we can avoid synchronal attacks and mitigate the overhead of network traffic. Each time when we do the update, the PHs would be re-selected again. This design increases the difficulty of attacks because it would be hard for attackers to anticipate which server nodes would do the update procedure next. Furthermore, each server node has the chance to be a PH, and we can then distribute the overhead of playing the PH evenly to all server nodes. Therefore, in comparison to the traditional proactive secret sharing, our secret share update method not only improves the effectiveness and efficiency, but also increases the level of security.

There are many reasons for selecting the PHs in the way we proposed. First, it is an easy selection method. Due to the characteristic of $(q, 1) - \text{BIBD}$, for partition P , when we select the PHs from the blocks that are derived from other partitions, we are not required to check whether the PHs we have selected are exactly distributed in each block of P . Because the $(q, 1) - \text{BIBD}$ guarantees that each pair of points occurs in exactly one block, for partition P , each pair of server nodes within the same block would not appear together in any other blocks. On the other hand, for other partitions, the server nodes in the same block would not show up as a couple. That means they would all belong to different blocks. There are q server nodes in one block, and there are also q blocks in a partition. Therefore, each server node in the other partitions' block would belong to the blocks of partition P respectively. The second reason is for the verification. In section 2.5, the verifiable secret sharing was presented. When we select the PHs by the unit of blocks, there exists a property. All the PHs would hold a common secret share. So, the PH can make use of their common secret share to do encryption on their transmitting information.

Chapter 4 Evaluation and Analysis

In the current chapter, evaluation and analyses of our proposed scheme are discussed. We first use the Maple to obtain the finite fields and then use them to construct the affine plane. After that, using the affine plane of order q for input, we write C codes to construct the corresponding $(q, 1)$ – BIBD and (q, w) – PHF. By using the $(q, 1)$ – BIBD and (q, w) – PHF, we complete the following analyses. For the first and second analyses, we also write C programs with the input of PHF blocks to simulate the mobility phenomenon. We test the effect of server nodes losing connection on our system. Finally, based on the relationship between a $(q, 1)$ – BIBD and a (q, w) – PHF, we want to show that there is an equal probability of each server node being chosen as a PH. All of the analysis programs are run on a Pentium IV 2.66GHz laptop.



4.1 Evaluation

Our scheme uses the secret sharing update procedure, which was introduced in section 3.3.3. In this section, we evaluate the communication cost of our proposed mechanism. The original concept of proactive secret sharing was presented in an earlier section of this text, and recalling this concept, n server nodes are required to update the secret shares [7]. Each server node should generate an update polynomial function with secret 0 and then use other server nodes' IDs to calculate the new partial secret shares. After that, the server node should transmit the partial secret shares to other corresponding server nodes. A server node cannot complete the secret share update procedure unless he receives the partial secret shares from all of n server nodes individually and combines them with his old secret share. The

communication cost refers to the network traffic overhead of all server nodes transmitting all secret shares to complete the secret share update.

From the description above, we know that in the original proactive secret sharing method, each server node should generate n partial secret shares and sends them out. Since there are n server nodes and each one generates n partial secret shares, the communication cost is therefore $O(n^2)$.

In our proposed scheme, however, only the partition headers are involved in the secret share update procedure. According to the format of (q, w) – PHF, although there are q^2 server nodes, only q PHs participate in the update procedure. In other words, if there are n server nodes, the number of PHs is exactly \sqrt{n} . As the result, the communication cost of transmitting secret shares among PHs is $O(n)$. Upon receiving the new secret share, the PH has the responsibility to send the new share to members in his block. This kind of communication cost is called the intra-block communication cost. Depending on the network traffic, the PH can decide when to send the new secret share to his members. Thus, the intra-block communication would not have a significant impact on the entire network traffic. Furthermore, we evaluation the entire system communication cost. That is, we consider the case that all the polynomial functions perform the update procedure. In a (q, w) – PHF with n server nodes, there are $\sqrt{n} + 1$ polynomial functions. Therefore, the communication cost of the entire system is $O(n^{3/2})$. Clearly, the communication overhead of our secret share updating procedure in one polynomial function is comparatively much lower than the original one, as shown in Table 4.1.1. Since there usually has bandwidth constraints in the MANET, the result of our designed scheme is evidently much more suitable for the MANET environment.

Evaluation	Traditional Proactive Secret Sharing	PHF-based Secret Share update	The entire PHF-based system
Communication Cost	$O(n^2)$	$O(n)$	$O(n^{3/2})$

Table 4.1.1 Communication cost

4.2 Analysis

Since our proposed protocol design is built from the concept of perfect hash families, in our system, each server node holds more than one secret share. Furthermore, each secret share that a server node holds comes from different polynomial functions. Thus, more than one polynomial function map to the same system secret, and they all are required to do the secret share update. Moreover, in our system, different polynomial functions can do the update procedure in different time slots. Therefore, we can take advantage of having multiple polynomial functions to enhance the effectiveness of our system. In this section, three analyses of our system are presented.

Before we report the result of the first analysis, we first discuss some situations below. Due to the characteristics of involving multiple functions, our system can still remain operational even if some of the polynomial functions fail to successfully complete the secret share update. In fact, as long as there is one polynomial function that can perform the secret share update, our network system can work well and the server nodes also can periodically refresh one of their secret shares. Thus, we discuss in what situation a polynomial function would fail to perform the secret share update procedure. In the beginning phase of the updating procedure, we have to choose the PHs for the partition in corresponding polynomial function. Therefore, we say that a polynomial function cannot execute the update procedure if and only if we cannot successfully select the PHs for it.

So, in the first instance, due to the mobility in the MANET, we want to examine after how many server nodes disappeared, or lost connection would cause one of the polynomial functions not able to do the update procedure. We analyze the (q, w) – PHF by writing some C programs. We check all of the combinations for q^2 server nodes and obtain the probability of a polynomial function failing to update the secret share. We test the case when $q = 3, 4, 5$.

Second, going a step further, we want to know after how many server nodes disappeared, or lost connection would lead to a complete system failure. This analysis is similar to the first one. The major difference between these two is that in the second analysis, we have to make sure that all polynomial functions cannot perform the secret share update procedure. We also test the case when $q = 3, 4, 5$. The following figures show the results of the above analyses.

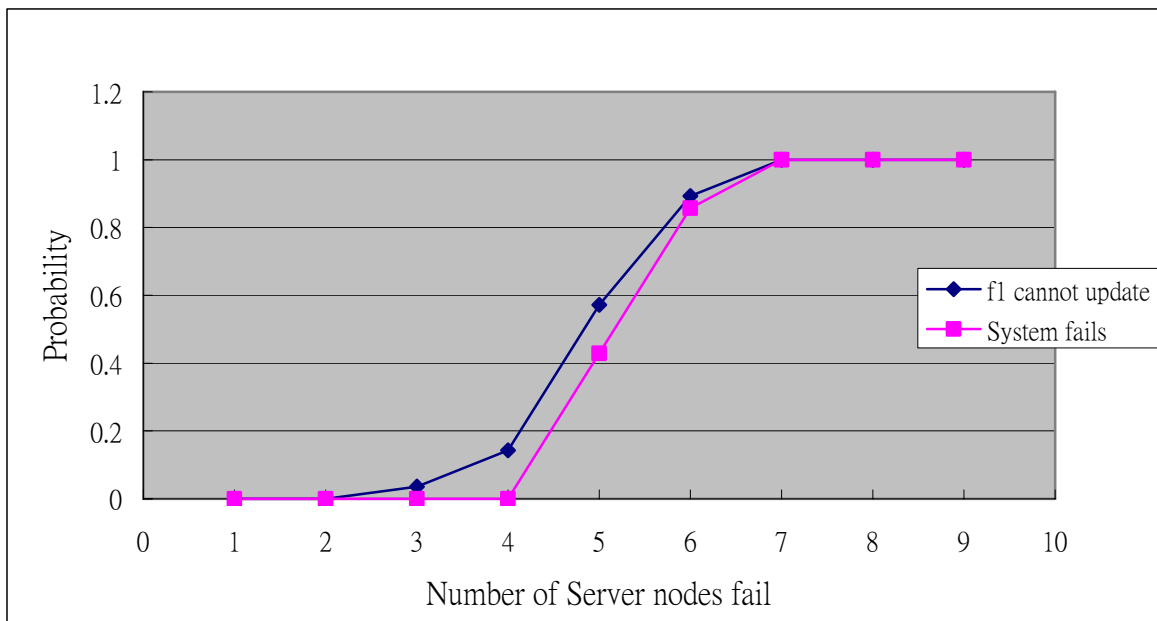


Figure 4.2.1 PHF, $q = 3$

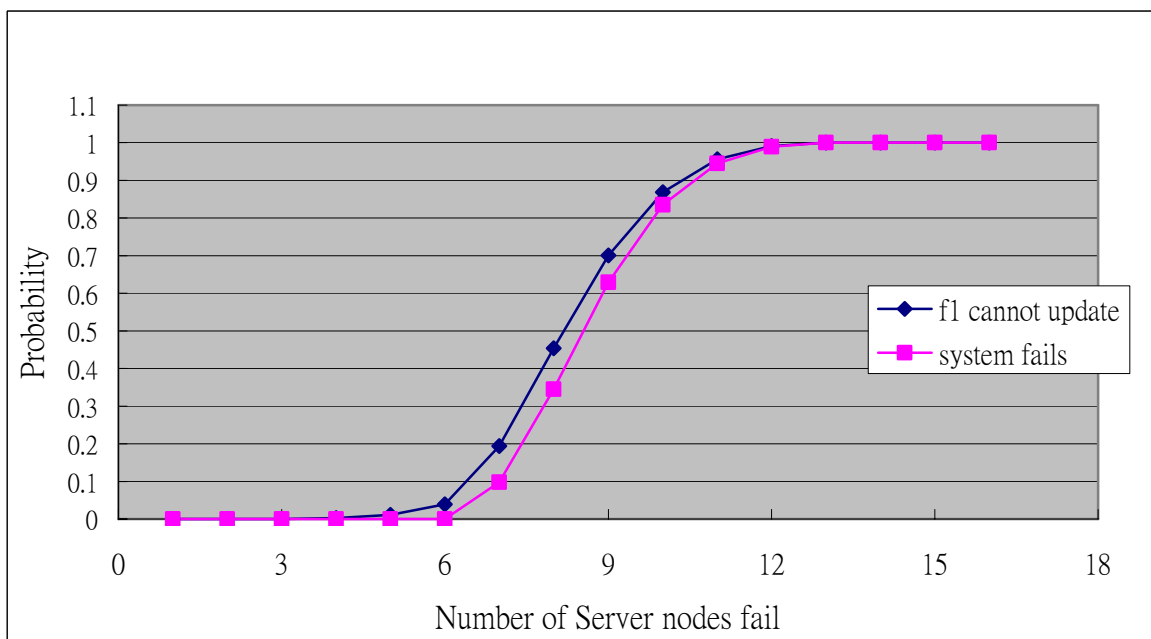


Figure 4.2.2 PHF, $q = 4$

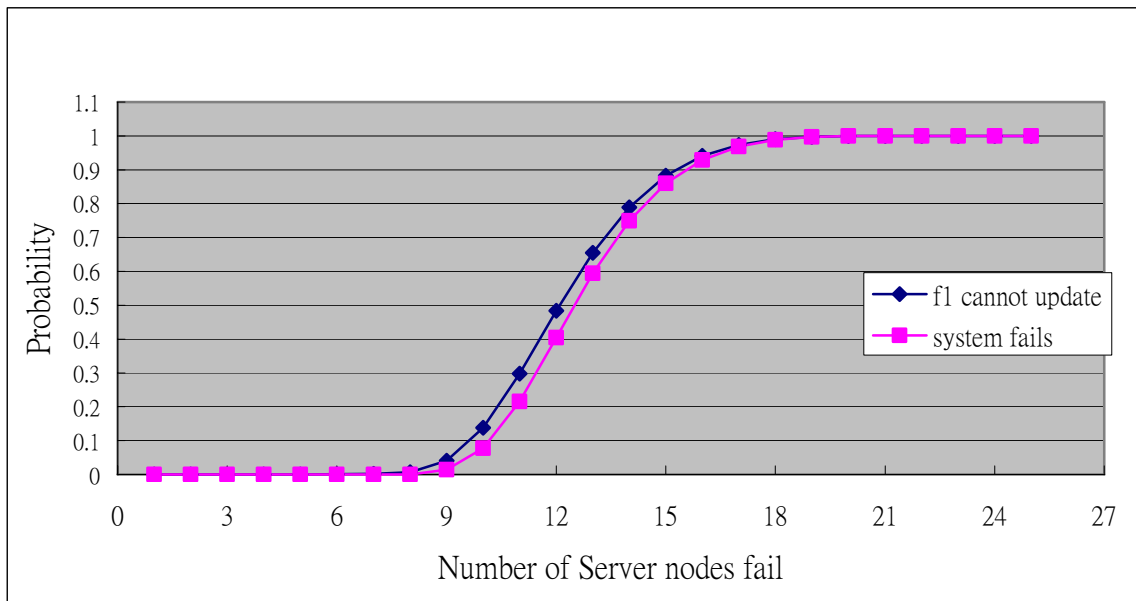


Figure 4.2.3 PHF, $q = 5$

Finally, we analyze the probability of a server node being selected as a PH. For the secret share update procedure, the PH has to generate the new partial shares and distribute the new secret share to his members. Thus, there is an extra overhead associated with being a PH. In our scheme, we want to distribute this extra overhead to all server nodes, instead of just covered by a few specific server nodes. Therefore, we write programs with different parameter T s – meaning the number of secret share update procedure that we have– in order to show the equal probability among server nodes. Results are presented in the following figures.

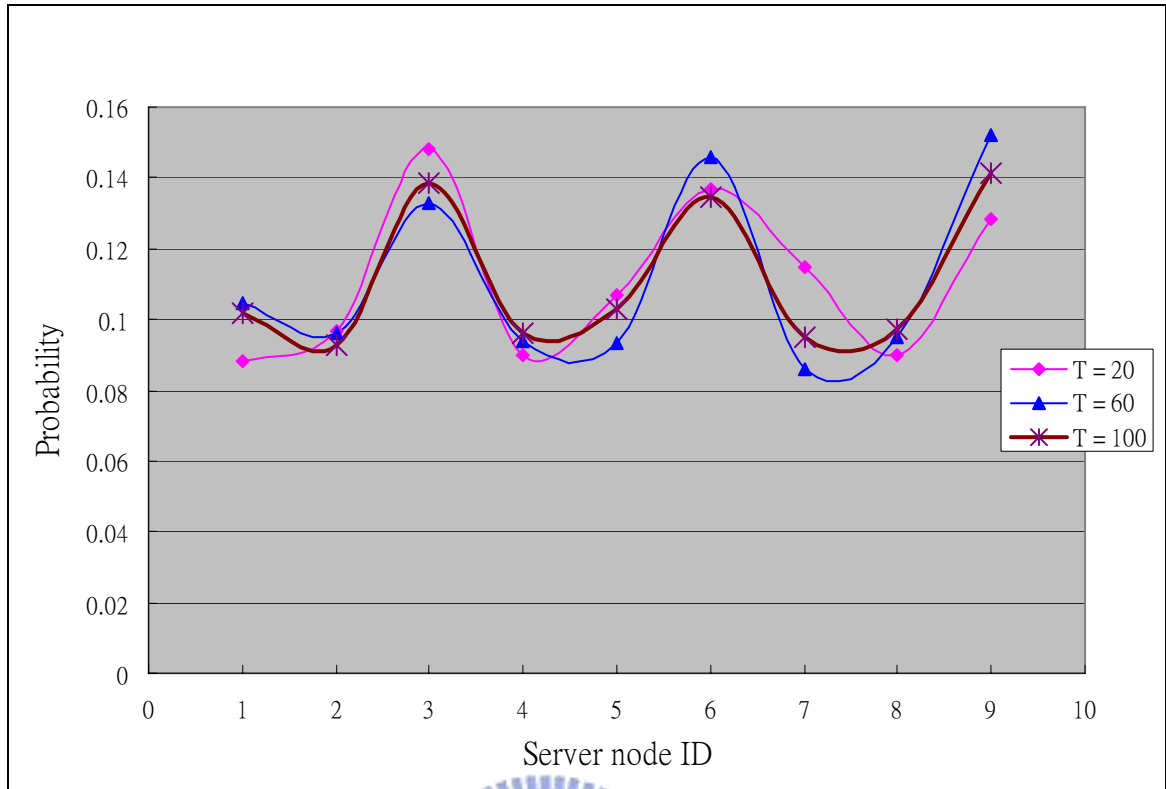


Figure 4.2.4 PHF, $q = 3$

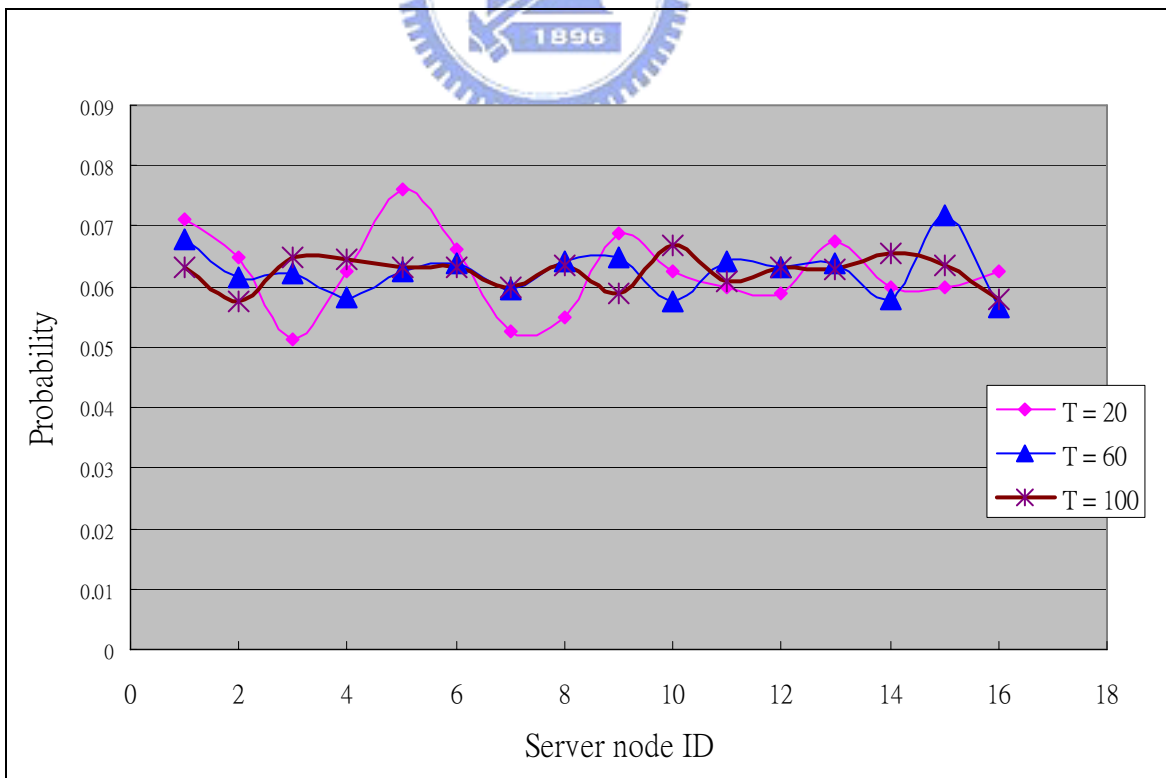


Figure 4.2.5 PHF, $q = 4$

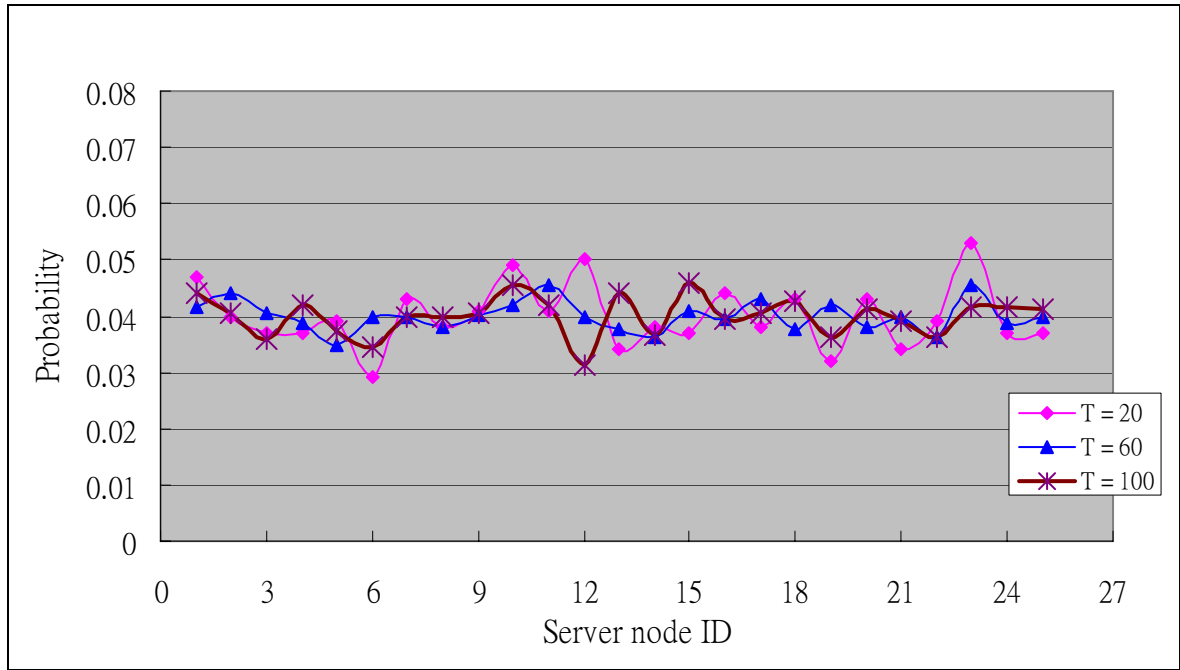


Figure 4.2.6 PHF, $q = 5$

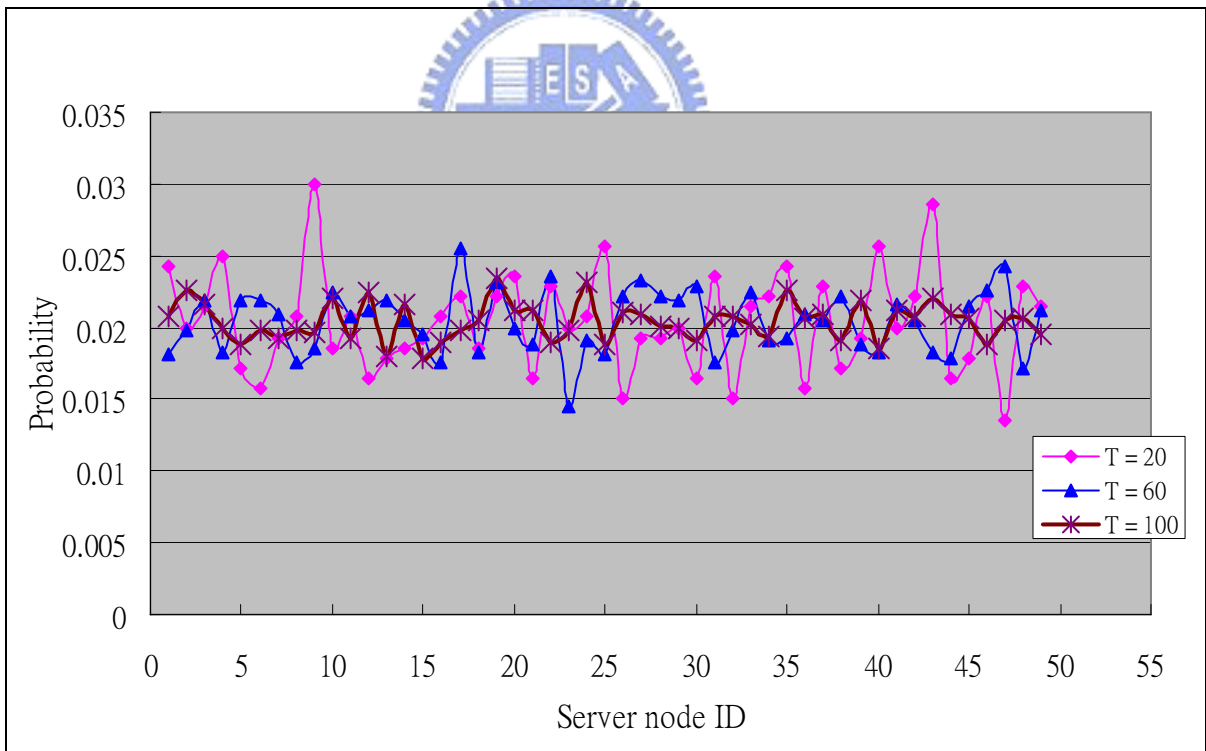


Figure 4.2.7 PHF, $q = 7$

4.3 Discussion

In the previous section, we can see the probability of when a polynomial function and the entire system would fail. Due to the characteristic of $(q, 1) - \text{BIBD}$, we know each pair of elements occurs in exactly one block. On the other hand, in the $(q, w) - \text{PHF}$, the server nodes in the same block of partition P_i must be in the distinct blocks of other partitions. Thus, in order to make a polynomial function f_i fail to choose PHs, the other blocks in other partitions must be destroyed. The minimum number of nodes that needs to destroy all other blocks must equal to q . In this case, the q server nodes must exist in the same block of partition P_i . Furthermore, if we want to destroy the entire system, not only the blocks in other partitions, but also the blocks in P_i that we have to destroy. So the minimum number of server nodes would be $q + (q-1)$. As shown in Figure 4.2.1, Figure 4.2.2 and Figure 4.2.3, we verify that the minimum number of server nodes that could make a polynomial function fail to update is q . The probability in this case is less than 1%. Furthermore, the minimum number of server nodes that would make the entire system fail to update is $q + (q-1)$. The probability in this case is also less than 1%.

In addition to finding the lower bound of server nodes, we also find the upper bound. That means when the server nodes amount to a special number, either one of the polynomial functions or the entire system would crash. In other words, obtaining the upper bound allows us to find out when the server nodes amount to a special number that would result in either one of the polynomial functions or the entire system to crash. We can easily find that the special number is $q*(q-1) + 1$. Therefore, if more than $q*(q-1) + 1$ server nodes fail, our system would fail too. Figure 4.2.1, Figure 4.2.2 and Figure 4.2.3 reveal these results.

In the third analysis, we show that each server node has an equal probability of being

selected as a PH. Looking at the result figures, we can see that the more server nodes that we have in our system, the more even probability of each one being a PH. Furthermore, it also shows that the probability of balancing the overhead to all server nodes increases as the number of secret share update procedure increases. Therefore, in our system, we can evenly balance the overhead that each server node has to contribute.

From the above evaluation and analyses, the following conclusions can be derived: First, the network communication overhead of secret share update in our system is considerably less than the traditional proactive secret sharing method. Second, depending on the properties of PHF, we improve the robustness of our protocol design. Thus, our system handles the dynamic environment reasonably well. Finally, the partition header selection scheme allows the resource consumption be shared by all server nodes.



Chapter 5 Conclusion

Nowadays, since wireless technology and applications have become increasingly popular, there is a high demand in developing a secure wireless network. In order to ensure the integrity and confidentiality of data during transmissions, people are paying more attention to issues related to network security. Considering some inherent limits in the mobile ad hoc network, the traditional PKI cannot be directly applied to the mobile ad hoc network without any modification. The purpose of this thesis is therefore to make some changes to the traditional PKI. We use the properties of PHF to carry out the threshold secret sharing for distributing the trust. Furthermore, we enhance the efficiency of our secret share renewal procedure by choosing only the PHs to do it.

In our proposed protocol design, the effectiveness and efficiency is improved. Analysis results also show that the extra overhead is shared by a group of nodes. Overall, results reveal that our system is more suitable for a mobile ad hoc network. Below are some issues and several suggestions for adjustment to future studies. In our protocol design, a $(q, w) - \text{PHF}$ would generate $q+1$ polynomial functions, and each server node would have $q+1$ secret shares. The bigger the q is, the more polynomial functions and secret shares a server node should hold. Thus, if q is quite big, the server nodes need to increase the storage to store the secret shares. And, it may also increase the complexity in system operation. Each time when we do the update procedure, the PHs would be re-selected again, and it is the additional cost of doing the update procedure. Also, we make the assumption that the server nodes are static when the PH selection process begins. That is, throughout the entire update procedure, the mobility of server nodes is not considered. Taking the mobility at all times into account is a good research topic for future studies. Finally, how often the secret shares need to be updated is also key in balancing between the network loading and the system safety.

References:

- [1] W. Stallings, "Network security essentials: Applications and standards," Prentice Hall, 2000.
- [2] E. Maiwald, "Network security: A beginner's guide," McGraw-Hill, 2001.
- [3] K. Schmeh, "Cryptography and Public Key Infrastructure on the Internet," John Wiley, 2003.
- [4] A. Shamir, "How to Share a Secret," Communications of ACM, vol. 22, no 11, pp. 612 – 613, 1979.
- [5] K. Kyung-Mi, "Perfect Hash Families: Constructions and Applications," a thesis of University of Waterloo, 2003.
- [6] S. R. Blackburn, "Combinatorics and threshold cryptography," in "Combinatorial Designs and their Applications," Chapman and Hall/CRC Research Notes in Mathematics, vol. 403, pp. 49 – 70, 1999.
- [7] A. Herzber, S. Jarecki, H. Krawczyk, M. Yung, "Proactive Secret Sharing Or: How to Cope With Perpetual Leakage," in Advances in Cryptology, Proc. CRYPTO'95, ser. LNCS, vol. 936, pp. 339 – 352, 1995.
- [8] L. Zhou, Z. J. Haas, "Securing Ad Hoc Networks," IEEE Networks, Volume 13, Issue 6, pp. 24 – 30, 1999.
- [9] J. Kong, P. Zerfos, H. Luo, S. Lu, L. Zhang, "Providing Robust and Ubiquitous Security Support for Mobile Ad-Hoc Networks," IEEE ICNP Nov. 2001, pp. 251 – 260, 2001.
- [10] H. Luo, P. Zerfos, J. Kong, S. Lu, L. Zhang, "Self-securing Ad Hoc Wireless Networks," IEEE ISCC'02, pp. 567 – 575, 2002.
- [11] M. Bechler, H.-J. Hof, D. Kraft, F. Pählke, L. Wolf, "A Cluster-Based Security Architecture for Ad Hoc Networks," IEEE INFOCOM, vol. 4, pp. 2393 – 2403, 2004.
- [12] B. Zhu, F. Bao, R. H. Deng, M. S. Kankanhalli, G. Wang, "Efficient and robust key management for large mobile ad hoc network," Computer Networks, Volume 48, Issue 4, pp. 657 – 682, 2005.
- [13] B. Wu, J. Wu, E. B. Fernandez, M. Ilyas, S. Magliveras, "Secure and efficient key management in mobile ad hoc networks," Journal of Network and Computer Applications, SSN'2005, pp.288, 2005.

- [14] H. Delfs, H. Knebl, "Introduction to Cryptography: Principles and Applications," Berlin, 2002.
- [15] W. Stallings, "Cryptography and Network Security: Principles and Practices," 3rd edition, Prentice Hall, 2003.
- [16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, "Introduction to Algorithms," 2nd edition, MIT Press, 2001
- [17] M. Atici, D. R. Stinson, and R. Wei, "Some recursive constructions for perfect hash families," Journal of Combinatorial Designs, pp. 353 – 363, 1995.
- [18] H. Wang, C. Xing, "Explicit constructions of perfect hash families from algebraic curves over finite fields," Journal of Combinatorial Theory, Series A, pp. 112 – 124, 2001.
- [19] K. Mehlhorn, "Data Structures and Algorithms 1: Sorting and Searching," Springer-Verlag, 1984.
- [20] C. C. Lindner, C. A. Rodger, "Design Theory," CRC Press, 1997.
- [21] R. E. Klima, N. Sigmon, E. Stitzinger, "Applications of Abstract Algebra with Maple," CRC Press, 2000
- [22] IETF Mobile Ad Hoc Networks (MANETs) Working Group,
<http://www.ietf.org/html.charters/manet-charter.html>.
- [23] C. Siva Ram Murthy, B. S. Manoj, "Ad Hoc Wireless Networks: Architectures and Protocols," Prentice Hall, 2004.
- [24] S. Corson, J. Macker, RFC 2501, "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations," IETF, 1999.
- [25] C. Cachin, J. A. Poritz, "Secure Intrusion-tolerant Replication on the Internet," IEEE DSN-2002, pp. 167 – 176, 2002.

Appendix A

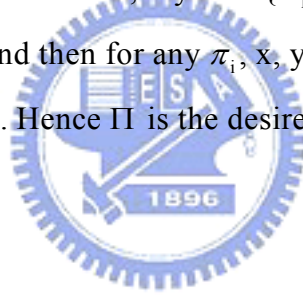
1. The proof of Proposition 2.6.6 [5].

Proof:

Let $\Pi = \{\pi_1, \pi_2, \dots, \pi_n\}$ be a family of w -partition of A . We can construct a collection F of functions, which map the elements from A to B , by labeling the parts of each partition π_i with distinct elements of B , and then defining f_i to map each $x \in A$ to the label of the part of π_i containing x . Then resulting set of functions, say $F = \{f_1, f_2, \dots, f_N\}$, is an (n, m, w) -perfect hash family.

Conversely, suppose that $F = \{f_1, f_2, \dots, f_N\}$ is a $\text{PHF}(N; n, m, w)$. We can construct a set of partitions of A , say $\Pi = \{\pi_1, \pi_2, \dots, \pi_n\}$, by setting π_i to f_i for all $i = 1, 2, \dots, N$. And then for any π_i , $x, y \in A$ in the same part of π_i whenever $f_i(x) = f_i(y)$. Hence Π is the desired set of partition of A .

Q.E.D



2. The proof of Proposition 2.6.7 [5].

Proof:

For given an (n, m, w) -perfect hash family $F = \{f_1, f_2, \dots, f_N\}$, we can produce an array M of size $N * n$ with entries in B as follows: Index the columns of M by the elements $x \in A$, and index the rows of M by the set $\{1, 2, \dots, N\}$. That is, each row of the array corresponds to one of the functions in the family F . Setting the value of the entry (i, x) in M to be $f_i(x)$, the resulting array satisfied the desired conditions.

In the reverse direction, suppose that M is an array of size $N * n$, having entries on B . For $i = 1, 2, \dots, N$ and $x \in A$, we define $f_i(x)$ to be the value of the entry (i, x) of M .

Hence, $f_i(x) = f_i(y)$ for $f_i \in F$ whenever the (i, x) th and (i, y) th entries of M are equal. Then we have a desired set $F = \{f_i : 1 \leq i \leq N\}$, which is a $\text{PHF}(N; n, m, w)$.

Q.E.D

3. The proof of Theorem 2.6.10 [21].

Proof:

To show that equation $vr = bk$ holds, we consider the set

$T = \{(a, B) \mid a \text{ is an object in block } B\}$, and count $|T|$ in two ways.

First, the design has v objects that each appear in r blocks. Hence, $|T| = vr$.

But the design also has b blocks that each contains k objects. Hence, $|T| = bk$.

Thus $vr = bk$.

To show that $(v-1)\lambda = r(k-1)$, we choose an object a_0 in the design.

Then for $U = \{(x, B) \mid x \text{ is an object with } a_0 \text{ in block } B\}$, we count $|U|$ in two ways.

First, there are $(v-1)$ objects in the design that each appear in λ blocks with a_0 , so $|U| = (v-1)\lambda$.

But there are also r blocks in the design that each contains a_0 and $(k-1)$ other objects.

Hence, $|U| = r(k-1)$.

Thus $(v-1)\lambda = r(k-1)$.

Q.E.D

4. The proof of Theorem 2.6.11 [5].

In order to obtain the Theorem 2.6.11, we first give a lemma.

Lemma:

Let (X, A) be a resolvable (v, b, r, k, λ) – BIBD and Π is a set of parallel class. For any subset Y of w points, there exists a parallel class $\pi \in \Pi$ such that the w points in Y occur in w different blocks in π .

Proof:

Let Y be a set of w points of X . Suppose that there exists no parallel class $\pi \in \Pi$ separating Y . Then each parallel class cannot separate some pair of elements in Y . By the definition of a resolvable (v, b, r, k, λ) – BIBD, we note that any pair of points in X occurs in exactly λ blocks. Thus, there are at most λ parallel classes in Π that do not separate a fixed pair of elements. Hence, there are at most $\lambda \binom{w}{2}$ parallel classes in Π that do not separate Y . In a resolvable (v, b, r, k, λ) – BIBD, there are r parallel classes and $r > \lambda \binom{w}{2}$. Thus, there exists at least one parallel class in Π that separates Y .

Q.E.D

Proof of Theorem 2.6.11:

Let $\Pi = \{\pi_i : 1 \leq i \leq r\}$ be a set of parallel class. For any subset Y of w points, by the above lemma, we know that there exists a parallel class $\pi \in \Pi$ such that the w points in Y occur in w different blocks in π . Thus, define a family

$F = \{f_i : A \rightarrow B, \text{ and } 1 \leq i \leq N\}$ as follows:

For any $1 \leq i \leq r$, define $f_i(x) = j$ whenever $x \in A$ is in the j th block in π_i .

Clearly, $j \leq v/k$ and f_i is an $(n, v/k)$ hash function.

Thus the resulting set $F = \{f_i : 1 \leq i \leq r\}$ is a PHF($r; v, v/k, w$) since for any $x, y \in A$, x and y are in the same block of π_i if and only if $f_i(x) = f_i(y)$.

Q.E.D



Appendix B

1. The construction program of 3.1.1.

Based on the prime power q , each program has the different input for construction. We take the program with $q = 4$ for example.

```
//Affine Plane to BIBD
//Affine_to_BIBD_x.c
//Input: Affine plane of order q
//Output: BIBD( $q^2$ ,  $q(q+1)$ ,  $q+1$ ,  $q$ , 1)

// q = 4

#include <stdio.h>
#define q 4

void main()
{
    FILE *stream;

    int ADD[q][q] = {
        {4,1,2,3}, {1,4,3,2}, {2,3,4,1}, {3,2,1,4} };
    int MUL[q][q] = {
        {4,4,4,4}, {4,1,2,3}, {4,2,3,1}, {4,3,1,2} };

    int A[q][q];
    int Transform[q][q];
    int MOLLS[q-1][q][q];
    int BIBD[q+1][q][q];

    int i, j, k, x, y, m, c;
    int index;

    stream = fopen("4-BIBD.txt", "w");

    for(i=1; i<q+1; i++)
```



```

{
    for(j=1; j<q+1; j++)
    {
        A[i-1][j-1] = q+10*j-(i-1);
//      printf("A[%d][%d] = %d\n", i-1, j-1, A[i-1][j-1]);
        if(j == 1)
            fprintf(stream, "A[%d] = %d", i-1, A[i-1][j-1]);
        else fprintf(stream, " %d", A[i-1][j-1]);
    }
    fprintf(stream, "\n");
}

//transfer to MOLS(q)
for(i=1; i<q; i++)
{
//      printf("x * %d + y\n", i);
    fprintf(stream, "x * %d + y\n", i);
    for(x=0; x<q; x++)
    {
//          printf("%d * %d + y\n", x, i);
//          printf("MUL[%d][%d] = %d\n", x, i, MUL[x][i]);
        y = MUL[x][i];

        for(j=0; j<q; j++)
        {
            if(y == q)
            {
                y = 0;
//              printf("9 * %d + y\n", i);
            }

            if(j == 0)
            {
//              printf("ADD[%d] = %d", y, ADD[y][j]);
                fprintf(stream, "%d", ADD[y][j]);
                MOLS[i-1][x][j] = ADD[y][j];
            }
            else
            {

```

```

//          printf(" %d", ADD[y][j]);
           fprintf(stream, " %d", ADD[y][j]);
           MOLS[i-1][x][j] = ADD[y][j];
           }
       }
       fprintf(stream, "\n");
       printf("\n");
   }
}
for(i=0; i<q-1; i++)
{
    for(j=0; j<q; j++)
    {
        for(k=0; k<q; k++)
        {
            if(k == 0)
                printf("MOLS[%d][%d] = %d", i, j, MOLS[i][j][k]);
            else printf("%d", MOLS[i][j][k]);
        }
        printf("\n");
    }
    printf("\n");
}

//Transfer the form
fprintf(stream, "\nTransfer the form.\n");
c = 1;
for(i=1; i<q+1; i++)
{
    for(j=1; j<q+1; j++)
    {
        Transform[i-1][j-1] = c;
//        printf("T[%d][%d] = %d\n", i-1, j-1, Transform[i-1][j-1]);
        if(j == 1)
            fprintf(stream, "T[%d] = %d", i-1, Transform[i-1][j-1]);
        else fprintf(stream, " %d", Transform[i-1][j-1]);

        c++;
    }
}

```




```

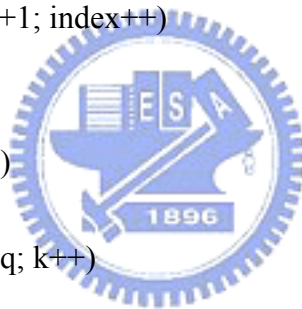
    fprintf(stream, "\n");
}

//Construct the BIBD block - BIBD[q+1][q][q]
for(i=0; i<q; i++)
{
    for(j=0; j<q; j++)
    {
        BIBD[0][i][j] = Transform[i][j];
        BIBD[1][i][j] = Transform[j][i];
    }
}

for(i=0; i<q-1; i++)
{
    m=0;
    for(index=1; index<q+1; index++)
    {
        for(j=0; j<q; j++)
        {
            for(k=0; k<q; k++)
            {
                if(MOLS[i][k][j] == index)
                {
                    printf("%d ", Transform[k][j]);
                    BIBD[i+2][m][j] = Transform[k][j];
                }
            }
        }
        m++;
        printf("\n");
    }
    printf("\n");
}

//Output to the file
for(i=0; i<q+1; i++)
{

```



```

for(j=0; j<q; j++)
{
    for(k=0; k<q; k++)
    {
        if(k == 0 && j == 0)
        {
//            printf("BIBD[%d][%d] = %d", i, j, BIBD[i][j][k]);
            fprintf(stream, "BIBD[%d] = \n{%d", i, BIBD[i][j][k]);
        }
        else if(k == 0)
            fprintf(stream, "{%d", BIBD[i][j][k]);
        else
        {
//            printf(" %d", BIBD[i][j][k]);
            fprintf(stream, ", %d", BIBD[i][j][k]);
        }
    }
    fprintf(stream, "}\n");
}
fprintf(stream, "\n");
}

fclose(stream);
}

```



2. The analysis program of chapter 4.

With different prime power q , we have the different input for analysis. Furthermore, we use the variable “A” to present the number of server nodes disappear, being compromised or lost connection. Thus, for the following example program, we just take $q = 4$ and $A = 7$ for example.

The program for the first and second analysis:

```

// BIBD, PHF, q = 4
// Number of server node = 7
// PHF_x_analysis.c

```

```

#include <stdio.h>
#include <stdlib.h>
#define q 4

void compare(int, int, int, int, int, int, int);

double count = 0;
double count1 = 0;

void main(int argc, char* argv[])
{
    int A = 7; // number of nodes to be selected
    int i, j;
    double num = 1;
    int flag=0;
    int *ser_n;
    int a,b,index=0;

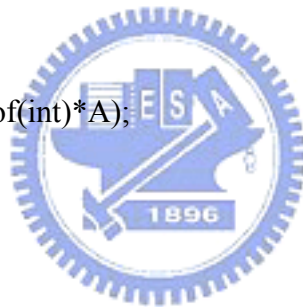
    ser_n = (int *)malloc(sizeof(int)*A);

    //# of Server node = 7
    for(i=0;i<A;i++)
    {
        ser_n[i]=i+1;
    }

    /* first node */
    printf("%d,%d,%d,%d,%d,%d,%d\n",
ser_n[0],ser_n[1],ser_n[2],ser_n[3],ser_n[4],ser_n[5],ser_n[6]);
    compare(ser_n[0],ser_n[1],ser_n[2],ser_n[3],ser_n[4],ser_n[5],ser_n[6]);
    printf("count = %d\n", count);

    while(ser_n[0]<=10)
    {
        for(i=A-1,j=0; i>0; i--,j++)
        {
            if(ser_n[i]==(16-j))

```



```

    {
        /* trigger 的處理 */
        for(a=i-1;a>=0;a--)
            if(ser_n[a] < (16-(A-a-1)))
            {
                ser_n[a]++;
                for(b=a+1;b<A;b++)
                    ser_n[b]=ser_n[b-1]+1;
                a= -1;

                flag=1;
                num++;
                //          printf("%d,%d,%d,%d,%d,%d\n",
ser_n[0],ser_n[1],ser_n[2],ser_n[3],ser_n[4],ser_n[5]);

                compare(ser_n[0],ser_n[1],ser_n[2],ser_n[3],ser_n[4],ser_n[5],ser_n[6]);
            }
        /* End */
        if(flag==0)
        {
            printf("num = %lf\n", num);
            printf("count = %lf\ncount1 = %lf\n", count, count1);
            return;
        }

    }
}
if(flag==0)
{
    num++;

    ser_n[A-1]++;

    //printf("%d,%d,%d,%d,%d,%d\n",
ser_n[0],ser_n[1],ser_n[2],ser_n[3],ser_n[4],ser_n[5]);
    compare(ser_n[0],ser_n[1],ser_n[2],ser_n[3],ser_n[4],ser_n[5],ser_n[6]);
}
flag=0;
//printf("%d,%d,%d,%d,%d,%d\n",

```

```

ser_n[0],ser_n[1],ser_n[2],ser_n[3],ser_n[4],ser_n[5]);
    }
}

```

```

void compare(int a, int b, int c, int d, int e, int f, int g)
{

```

```

    int B[q*(q+1)][q] = {
        {1,2,3,4}, {5,6,7,8}, {9,10,11,12}, {13,14,15,16},
        {4,8,12,16}, {3,7,11,15}, {2,6,10,14}, {1,5,9,13},
        {4,7,10,13}, {3,8,9,14}, {1,6,11,16}, {2,5,12,15},
        {4,5,11,14}, {2,7,9,16}, {1,8,10,15}, {3,6,12,13},
        {4,6,9,15}, {1,7,12,14}, {3,5,10,16}, {2,8,11,13} };

```

```

int comp = q*q;
int comp1 = q*(q+1);
int x,y;

```

```

for(x=0; x<q; x++)

```

```

{

```

```

    for(y=0; y<q; y++)

```

```

    {

```

```

        if(B[x][y] ==

```

```

a||B[x][y]==b||B[x][y]==c||B[x][y]==d||B[x][y]==e||B[x][y]==f||B[x][y]==g

```

```

||B[x][y]==h||B[x][y]==i||B[x][y]==j||B[x][y]==k||B[x][y]==l||B[x][y]==m||B[x][y]==n
||B[x][y]==o||B[x][y]==p)

```

```

    {

```

```

        comp1--;

```

```

        y=q;

```

```

    }

```

```

}

```

```

}

```

```

for(x=q; x<q*(q+1); x++)

```

```

{

```

```

    for(y=0; y<q; y++)

```

```

    {

```

```

        if(B[x][y] ==

```

```

a||B[x][y]==b||B[x][y]==c||B[x][y]==d||B[x][y]==e||B[x][y]==f||B[x][y]==g)

```



```

        {
            comp--;
            comp1--;
            y=q;
        }
    }
}
if(comp == 0)
    count++;
if(comp1 == 0)
    count1++;

return;
}

```

The program for the third analysis:

```

// Check the probability to be a PH
// Check_PH_4.c
// q = 4

```



```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

```

```

#define randomize() srand((unsigned)time(NULL))
#define random(num) (rand()%(num))

```

```

#define delay 1000000
#define q 4
#define T 20 //the number of secret share update

```

```

void main()
{
    int B[q*(q+1)][q] = {
        {1,2,3,4}, {5,6,7,8}, {9,10,11,12}, {13,14,15,16},
        {4,8,12,16}, {3,7,11,15}, {2,6,10,14}, {1,5,9,13},
        {4,7,10,13}, {3,8,9,14}, {1,6,11,16}, {2,5,12,15},
        {4,5,11,14}, {2,7,9,16}, {1,8,10,15}, {3,6,12,13},

```

```
{4,6,9,15},{1,7,12,14},{3,5,10,16},{2,8,11,13} };
```

```
int index = 0;
int flag = 0;
int i, j, k, d, x;
int count[10][q*q];
int sum[q*q];

memset(count, 0, sizeof(count));
memset(sum, 0, sizeof(sum));
randomize();

for(k=0; k<10; k++)
{
    for(i=0; i<T; i++)
    {
        index = random(q*(q+1));
//        printf("index = %d\n", index);

        while(1)
        {
            flag = 0;
            if(B[index][0]==1 && B[index][1]==2 && B[index][2]==3 &&
B[index][3]==4)
                flag++;

            else if(B[index][0]==5 && B[index][1]==6 && B[index][2]==7 &&
B[index][3]==8)
                flag++;

            else if(B[index][0]==9 && B[index][1]==10 && B[index][2]==11 &&
B[index][3]==12)
                flag++;

            else if(B[index][0]==13 && B[index][1]==14 && B[index][2]==15 &&
B[index][3]==16)
                flag++;

            if(flag == 0)
```

```

        break;
        else index = random(q*(q+1));
    }
//    printf("after index = %d\n", index);

    for(j=0; j<q; j++)
    {
//        printf("B[%d][%d] = %d ", index, j, B[index][j]);
        count[k][B[index][j]-1]++;
    }
}

for(x=0, d=0; d<delay; d++)
{
    x++;
//    printf("x=%d\n", x);
}

for(k=0; k<10; k++)
{
    for(i=0; i<q*q; i++)
    {
        printf("count[%d][%d] = %d\n", k, i, count[k][i]);
    }
}

for(i=0; i<q*q; i++)
{
    for(k=0; k<10; k++)
    {
        sum[i] += count[k][i];
    }
}

for(k=0; k<q*q; k++)
{
    printf("sum[%d] = %d ave = %lf\n", k, sum[k], pro);
}

```

