

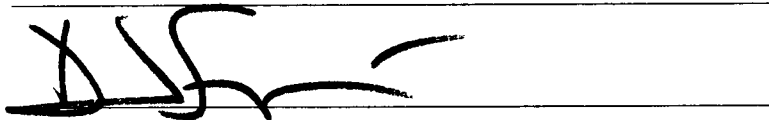
To the Graduate Council:

I am submitting herewith a thesis written by Cathy Xiaoqing Cao entitled "Group Key Management Architectures." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.



Dr. Tom Dunigan, Major Professor

We have read this thesis  
and recommend its acceptance:



Accepted for the Council:



Associate Vice Chancellor  
and Dean of the Graduate School

# **Group Key Management Architectures**

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Cathy Xiaoqing Cao

August 1997

## **Acknowledgments**

I would like to thank Dr. Tom Dunigan for his guidance and advice. I also thank Dr. Stephen Batsell for his support and Dr. David Straight for serving on my thesis committee.

Special thanks to my parents for their encouragement. They helped make this thesis possible.

I would like to thank my wonderful friends, who offered their help in various ways while revising this thesis.

## **Abstract**

This thesis describes an architecture and implementation for doing group key management over a data communications network. The Group Key Management Sample application protocol (GKMS) establishes a shared encryption key among an authenticated and authorized collection of network entities. GKMS uses a simple public key infrastructure for authentication and authorization and uses multicast for rekeying and compromise recovery. An application programming interface multiplexes key management and user application messages. An implementation of GKMS is described and evaluated.

# Contents

|          |                                            |          |
|----------|--------------------------------------------|----------|
| <b>1</b> | <b>Introduction</b>                        | <b>1</b> |
| <b>2</b> | <b>Background</b>                          | <b>5</b> |
| 2.1      | Importance of Internet Security . . . . .  | 7        |
| 2.2      | Key Management . . . . .                   | 8        |
| 2.2.1    | Public-Key Protocols . . . . .             | 10       |
| 2.3      | IPv6 key management protocols . . . . .    | 13       |
| 2.3.1    | New IP Security Options . . . . .          | 14       |
| 2.3.2    | IPv6 Key Management Protocols . . . . .    | 15       |
| 2.4      | IP Multicast . . . . .                     | 18       |
| 2.5      | Multicast Security Protocols . . . . .     | 20       |
| 2.6      | GKMP Objectives . . . . .                  | 22       |
| 2.6.1    | Features of GKMP . . . . .                 | 22       |
| 2.6.2    | Attacks that GKMP Guards Against . . . . . | 24       |

|          |                                                    |           |
|----------|----------------------------------------------------|-----------|
| <b>3</b> | <b>GKMS Architecture</b>                           | <b>25</b> |
| 3.1      | GKMS Protocol Overview . . . . .                   | 26        |
| 3.2      | GKMS Features . . . . .                            | 29        |
| 3.3      | GKMS Operations . . . . .                          | 30        |
| 3.3.1    | Group Establishment . . . . .                      | 32        |
| 3.3.2    | Group Member Deletion . . . . .                    | 33        |
| 3.3.3    | Group Rekey . . . . .                              | 34        |
| 3.4      | GKMS/GKMP Differences . . . . .                    | 34        |
| 3.5      | Unresolved GKMP Issues . . . . .                   | 36        |
| <b>4</b> | <b>Group Key Management Sample (GKMS) Protocol</b> | <b>37</b> |
| 4.1      | The GKMS software . . . . .                        | 38        |
| 4.2      | The Basic Structure of the Program . . . . .       | 40        |
| 4.2.1    | Server Function . . . . .                          | 40        |
| 4.2.2    | Member Functions . . . . .                         | 42        |
| 4.2.3    | Data Structure and Primitives . . . . .            | 43        |
| 4.3      | Message Definitions . . . . .                      | 45        |
| 4.4      | GKMS Function Definitions . . . . .                | 50        |
| 4.4.1    | Group Key Creation . . . . .                       | 50        |
| 4.4.2    | Group Key Distribution . . . . .                   | 52        |
| 4.4.3    | Group Member Deletion . . . . .                    | 53        |

|          |                                                                  |           |
|----------|------------------------------------------------------------------|-----------|
| 4.4.4    | Group Rekey . . . . .                                            | 54        |
| 4.4.5    | Member Join . . . . .                                            | 55        |
| 4.5      | Cryptographic Libraries Used . . . . .                           | 56        |
| 4.5.1    | DES Library . . . . .                                            | 56        |
| 4.5.2    | SSH Library . . . . .                                            | 56        |
| 4.6      | Comparison with GKMP Specification . . . . .                     | 57        |
| 4.7      | Protocol Evaluation . . . . .                                    | 60        |
| 4.7.1    | Some Security Issues Addressed in GKMS . . . . .                 | 60        |
| 4.7.2    | Performance . . . . .                                            | 63        |
| 4.8      | Miscellaneous Issues Associated with the GKMS Software . . . . . | 64        |
| <b>5</b> | <b>Future Work</b>                                               | <b>68</b> |
| <b>6</b> | <b>Summary and Conclusions</b>                                   | <b>71</b> |
|          | <b>Bibliography</b>                                              | <b>75</b> |
|          | <b>Appendix</b>                                                  | <b>80</b> |
| <b>A</b> | <b>Underlying functions of GKMS API</b>                          | <b>81</b> |
| <b>B</b> | <b>GKMS software configuration format</b>                        | <b>85</b> |
| <b>C</b> | <b>GKMS C code</b>                                               | <b>86</b> |



# List of Tables

|     |                                                     |    |
|-----|-----------------------------------------------------|----|
| 4.1 | Group key creation process . . . . .                | 51 |
| 4.2 | Group key distribution process . . . . .            | 53 |
| 4.3 | Group rekey process . . . . .                       | 55 |
| 4.4 | Group join process . . . . .                        | 55 |
| 4.5 | Comparison of group key management schemes. . . . . | 58 |
| 4.6 | Speed of cryptographic operations . . . . .         | 63 |

# List of Figures

|     |                                               |    |
|-----|-----------------------------------------------|----|
| 2.1 | A chain of payload datagram . . . . .         | 17 |
| 2.2 | Network layout for GKMP . . . . .             | 23 |
| 3.1 | Group rekey . . . . .                         | 27 |
| 3.2 | Group key creation and distribution . . . . . | 32 |
| 4.1 | Network data payload . . . . .                | 45 |
| 4.2 | GKMS API Interface . . . . .                  | 66 |

# Chapter 1

## Introduction

The growth of group communication requires an efficient communication protocol rather than the classic point-to-point connection. Multicast (§ 2.4) therefore is attracting increasing attention, especially in the area of audio and video conferencing. With multicast, a source machine no longer needs to send multiple copies of the identical information to a group of recipients: only a single information stream needs to be sent from the source.

Any network host can join and leave a multicast group. To create a private network and protect the information among the qualified group members, privacy, authenticity, and integrity must be supported in an upper level protocol, one of which is the group key management protocol. The traditional structure of a group key management protocol relies on a central Key Distribution Center (KDC). Use of this single point resource introduces potential problems such as

delay and inability to support large groups. Since a traditional KDC does not scale, a new group management protocol (GKMP) [20] has been proposed and designed. The intent of a GKMP is to manage a multicast group without relying on a centralized key distribution center. The group management is delegated down to a single member or a few group members. GKMP is designed to provide perfect forward secrecy and privacy for session members. It is extensible and *scalable*, which means that the group size can be variable. A perfectly scalable system should have no limitations on the size of a group.

Due to the complexity of the protocol, only limited internet experiments with GKMP have been undertaken. In 1994, Harney's group implemented a demonstration application with limited functionality which only demonstrated key distribution and manual key creation. The structure of the demonstration application is very rigid and does not support a multicast environment.

The research presented in this paper implements a GKMP API on the SUN Solaris system. It is called the Group Key Management Sample application protocol (GKMS). The goal was to implement an efficient GKMP implementation, and still remain close to the original proposal, analyzing various group key management architectures, implementing a variation of GKMP and demonstrating it with a test application. In keeping with the GKMP protocol, this GKMS implementation identifies the roles and responsibilities of the system elements. It accomplishes multicast key management (i.e key creation, distribution, synchro-

nization and compromise recovery). Like the GKMP protocol, the GKMS has the following features:

- Key management is transparent to the application.
- There is no centralized key distribution center.
- Keys are synchronized among group members.
- Uses multicast communication protocols.

This is the first implementation of the GKMP protocol with scalable multicast capability.

This research explored the area of multicast key management protocols, and we gained development and performance experience with GKMP. Many problems are not well explained and addressed in the original proposal (e.g, key deletion and compromise recovery). Therefore, it is the designer's opportunity to gain experience in solving realistic issues concerned in a commercial system, especially unresolved issues in the GKMP specification. This study helped discover implementation problems in the area of key management security on a real system.

This thesis is organized into six main parts. The first chapter contains the introduction and purpose of the GKMS. The second chapter provides background on multicast and security related issues. It also explores key management, which is the focus of this study. The third chapter provides a GKMP protocol overview, which is the architecture upon which the design of the GKMS is dependent. The

fourth chapter discusses the actual implementation and performance of the GKMS and its limitations. The fifth chapter discusses the lessons learned and future work that relates to this protocol. The sixth chapter summarizes the research on the GKMP.

## Chapter 2

# Background

It is possible to provide private communication and the secure transmission of sensitive data in a public network. This requires a *cryptographic system* that provides encrypted connections between communicating peers. Each cryptographic system uses *cryptographic algorithms* to protect the sensitive data. Cryptographic *keys* must be created and maintained. The security of a system not only depends on the cryptographic algorithms used, but also on the security of the key. A key can be lost, stolen, or corrupted. Since the management of the key is critical to secure communication, key management protocols are important. This study, in particular, addresses the Group Key Management Protocol (GKMP) [20] in a multicast environment, which allows “point-to-multipoint” connections.

The GKMP protocol employs several cryptographic systems: the public key infrastructure, digital signatures, and checksums. The public key infrastructure

creates public key pairs, associates them with host user identities, and distributes them. It helps to support a large number of communicators. Two important public key algorithms, RSA and Diffie-Hellman, are used in the Group Key Management Sample implementation. The RSA algorithm generates public keys and provides digital signatures, whereas the Diffie-Hellman protocol generates shared-secret keys between communicating peers. A one way hash function (MD5[4]) is used to ensure the integrity of a message.

It is possible to implement a key management protocol in different layers, such as the network layer, transport layer, or application layer. However, the implementation of network and transport layer protocols are restricted by the network architecture. When an underlying infrastructure does not provide security services, an application layer protocol becomes a reasonable solution. GKMS operates at the application layer.

This section first examines Internet security problems and the need for cryptographic systems. Then we examine advantages and protocols for the use of public keys in cryptographic systems. The following section discusses the IP (Internet Protocol) security extension protocol (IPv6) and related key management protocols that contain some of the same features as GKMS. The next two sections introduce and outline security issues for multicast, a protocol that increases the communication efficiency of our key management protocol implementation over prior implementations. The final sections reviews the motivation and objectives

of GKMP.

## 2.1 Importance of Internet Security

As the Internet has become more accessible to the general public, many problems have arisen due to hackers. Suppose Alice and Bob are two network users. Alice wants to send a message to Bob, saying "I will meet you at the library." Without any security protection, an attacker can read Alice's message, change the contents of the message or send a fake message to Bob pretending he is Alice. Or if Alice did not make it to the meeting she arranged, she could also deny that she was the message sender.

To protect data exchanged between Alice and Bob, a cryptographic system can be used on an existing network architecture. This cryptographic system should utilize a secure cryptographic facility and provide secure communications between network entities. The effectiveness of a cryptographic system is measured against four security issues: secrecy, authenticity, integrity and non-repudiation[24].

Secrecy requires that a third party be unable to determine the *plaintext*, which is the original message, corresponding to a given *ciphertext*, which is an encrypted version of the plaintext that has been encrypted with a cryptographic key. If Alice sends Bob a ciphertext message, nobody but Bob should be able to determine the plaintext version.

Authenticity ensures that the claimed sender of a message is the actual sender. It is often achieved by digital signatures [27]. A sender signs his name with his private key, and the receiver decrypts this ciphertext with the sender's public key. This helps Bob to identify that a message is from Alice.

The integrity of a message confirms that a transmitted message was not modified. It is often supported by an encrypted checksum or one-way hash function such as MD5 [31]. If Alice sends a message to Bob that includes an encrypted checksum, then Bob can determine that the message was not changed. Finally, Alice cannot deny sending a message signed with her private key (non-repudiation).

Our GKMS protocol addresses each of these problems through the use of public keys, which will be discussed in the next session.

## **2.2 Key Management**

Internet security problems can be addressed using a cryptographic system. A cryptographic system can provide applications with encrypted connections that provide privacy. Any cryptographic scheme (e.g., encryption) employed in the cryptographic system needs a key to operate. The creation, distribution, and maintenance of the key requires a key management system. There are two types of key management: shared-secret key management and public key management.

In a shared-secret key management system, two entities who wish to commu-

communicate privately must first securely establish a common but secret key. A third party is often used as a courier for this key. An alternative way is for the two parties to obtain a key from a central key authority with whom each can communicate securely. This introduces vulnerabilities due to the concentration of trust. A compromised central authority would compromise the entire system. Some of the disadvantages of using a shared-secret key can be resolved by employing a public key management system.

Without a public key infrastructure, two parties must exchange their public keys, which is more reliable and efficient than exchanging secret keys since public keys do not require encryption or a trusted third party before transmission. Thus, unlike with secret keys, only authenticity and integrity of the key exchange is an issue.

Key management provides key distribution and maintenance services. Key distribution and maintenance are traditionally done by a central Key Distribution Center (KDC). Each member in a group must also distribute its public key to other members through the KDC using a method such as certification [31]. One drawback of using a KDC is that the network can suffer from a single point failure. If the KDC fails, secure communication between group members is impossible. It also suffers from being a performance bottleneck since every request must go through the KDC. Therefore, a protocol that employs a KDC does not scale [7]. This problem is often solved by using a temporary key for the key encryption

(e.g., Diffie-Hellman [15]) and traffic encryption (e.g., DES, IDEA [31]).

The GKMP uses a public-key management system without relying on a centralized KDC. It provides authentication and Perfect Forward Secrecy [14]. Perfect Forward Secrecy (PFS) does not “allow the compromise of keys from previously wire-tapped sessions”[22].

### **2.2.1 Public-Key Protocols**

GKMS is based on public-key cryptography. The two public-key protocols used are Rivest-Shamir-Adelman (RSA) [24] and Diffie-Hellman. They provide mechanisms to negotiate and distribute secret keys and create and verify digital signatures. Public key systems, unlike secret key systems, do not need to rely on a centralized authority to distribute keys; rather, they permit users to establish a common key without using a third party. This kind of system not only provides greater convenience, but also enhances security by not sharing the private keys [11]. Digital signatures are another application of public key systems. Like a handwritten signature on paper, a digital signature guarantees the authenticity of a message to its recipient. It provides authentication, non-repudiation, and integrity.

### **Ravist-Shamir-Adelman (RSA) Algorithm**

The RSA scheme, named after its inventors, Rivest, Shamir, and Adleman, is one of the best known public key schemes. Like other cryptographic schemes, the algorithm does encryption and decryption and allows use of a variable length key. The RSA algorithm depends on two secret, large prime numbers to generate a public key and its corresponding private key. The details of the algorithm are beyond the scope of this paper; the interested reader should see [24]. Like all public key schemes, an application using RSA can encrypt sensitive data with the receiver's public key, and the receiver can decrypt with its own private key.

An application using an RSA digital signature can encrypt the checksum of a message with the sender's private key. This encrypted data block is called the signature. The receiver can verify this signature using the sender's public key. The use of digital signatures provides an efficient way to authenticate messages and provide non-repudiation without using a centralized authority.

RSA signature verification is more efficient than other algorithms such as Digital Signature Standard (DSS) verification [24]. DSS was proposed as a standard for digital signatures by the National Institute of Standards and Technology [27] and is used by GKMP. Since signature verification is used frequently in GKMP, RSA is used by GKMS for efficiency.

## Diffie-Hellman (DH) Exchange

Many key management algorithms use the DH exchange [15] to establish secret keys and provide Perfect Forward Secrecy [14]. Essentially, the Diffie-Hellman key exchange provides a mechanism by which two parties can negotiate a secret session key. It does not do encryption, decryption, or digital signatures. DH can be illustrated as follows: two parties A and B both know a prime number  $p$  and a generator  $g$ . Party A chooses a random number  $x$  and computes  $n = g^x \bmod p$  as its public value. Party B chooses his own random number  $y$  and computes  $m = g^y \bmod p$  as its public value. Then A and B exchange  $n$  and  $m$  values. Party A obtains the commonly-shared secret by evaluating  $n^y \bmod p$ ; and party B computes the shared secret by  $m^x \bmod p$ . The individually-computed shared secrets are identical [24].

The original DH key exchange protocol is designed to be used between two peers, although an extension of DH has been proposed to work with more users [10] [22]. The weakness of Diffie-Hellman is that there is no authentication involved in the public value exchange, so an attacker can launch a man-in-the-middle attack. Man-in-the-middle attacks can be illustrated in the following example. Suppose that A and B wish to communicate privately, using DH, but C wants to eavesdrop on messages between A and B. The scenario might proceed as follows:

1. A sends B A's public key. C intercepts this key and sends B C's public key

2. B sends A B's public key, but C intercepts this key and sends A C's public key.
3. So A sends B a message encrypted with C's public key, thinking it is B's key. C decrypts it with his private key, then encrypts it with B's public key and sends it to B.
4. When B sends A a message encrypted in C's public key, but thinks it is A's key, C decrypts it with its private key, then encrypts it with B's public key and sends it to A.

One solution to prevent this attack is for A and B to sign their public values (m and n) during the public value exchange. Since C does not know A's private key, it cannot sign the bad message to B because B will know the message has been exposed.

## **2.3 IPv6 key management protocols**

Internet Protocol version 6 (IPv6) is designed as an improvement of Internet Protocol version 4 (IPv4). Several key management protocols have been proposed for use with IPv6. Although GKMP and GKMS utilize the more widely used IPv4, it is important to examine the security advantages that could be gained by using IPv6 and understand how IPv6 does key management.

### 2.3.1 New IP Security Options

Both IPv6 and IPv4 are designed for use in interconnected systems of packet-switched computer networks. IPv6 adds more flow control and end-to-end data reliability to IPv4 [1]. Several new security features are associated with this version, including authentication, confidentiality and privacy [32].

Since IPv6 has a modular architecture, it allows users to choose the necessary features to design their secure systems. The optional security headers in IPv6 are normally the Authentication Header (AH) and Encapsulating Security Payload (ESP) header. IPv6 can work with different key management protocols and supports Security Association (SA) [26] by means of AH and ESP.

An SA is typically implemented in the kernel. It is an agreement negotiated between two or more parties and describes how the parties utilize security services. This agreement is presented by a set of information that is shared between all parties. Parameters in an SA include an authentication algorithm, cryptographic keys, key lifetime, and security level. When accessing SA attributes, parties use a pointer or identifier referred to as the Security Parameter Index (SPI). The basic function of the key management protocol is to create, manage and remove security associations.

IP AH and ESP provide integrity checking and authentication for IP datagrams [2]. A keyed-MD5 algorithm is used as its default mechanism for authentication.

ESP not only provides integrity and authentication, but also confidentiality. It does this by encrypting the data to be sent.

### **2.3.2 IPv6 Key Management Protocols**

The basic functions of the IPv6 key management protocol is to create, manage and remove security associations by using IP extension header AH or ESP. The key management protocols provide a framework for secure communication. They may use one of many key exchange mechanisms (e.g., Oakley [29]). Two IPv6 protocol proposals (SKIP and ISAKMP) illustrate key management issues between pairs of network entities.

#### **SKIP features**

One of the proposed IPv6 protocols is the Simple Key-Management for Internet Protocols (SKIP) used with session-less network layer protocols. In the basic SKIP protocol [6], each IP-based source and destination has a DH public value. The public value can be authenticated by X.509 certificates [5].

In SKIP, if a public key distribution infrastructure is not available, an out-of-band key distribution can be used to manually establish a long term master key. The long-term master key  $K_{ij}$  is used to generate traffic encryption keys  $K_p$ . An implicit master key  $K_{ijn}$  is a function of  $K_{ij}$  and counter "n". The details of the key generation algorithm are in [6].

SKIP is not vulnerable to the man-in-the-middle attacks since the DH public key parameters are authenticated. If one of the encryption keys  $K_p$  is ever compromised, the master key update algorithm described in [6] will replace the compromised key, since the master key  $K_{ij}$  and encryption key  $K_p$  are separated.

### **ISAKMP Features**

The Internet Security Association and Key Management Protocol (ISAKMP) “combines the security concepts of authentication, key management, and SAs to establish the required security for government, commercial, and private communications on the Internet” [25]. ISAKMP defines SA’s and sets up a private communication in public networks. It is designed for use in the application layer.

One particular feature of ISAKMP is the use of Oakley Key Determination Protocol [19] to establish and manage Security Associations. The Oakley Key Determination Protocol is based on an authenticated Diffie-Hellman key exchange algorithm[19]. The Oakley payload structure used in the ISAKMP provides flexibility in negotiating security associations. An Oakley message is composed of one or more payloads. A payload can be key, signature, or other security association information. (See Figure 2.1).

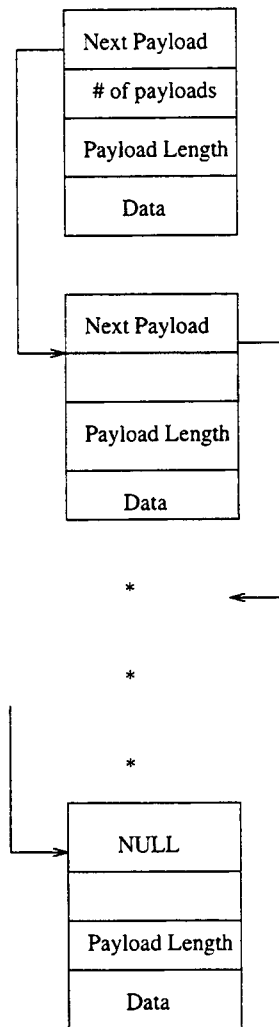


Figure 2.1: A chain of payload datagram

## 2.4 IP Multicast

The previous sections discussed general issues related to Internet security and key management. However, with the rise in popularity of multicast, other issues have begun to arise.

IP multicast has been in use since 1991 [13], but only in recent years, when internet-based audio/video teleconferencing became popular, has its ability to provide efficient group communication become increasingly important. In contrast, unicast has been studied and is well known to network communities. Unicast is the classic IP communication in which two communicating hosts on a network use a point-to-point connection. Information that is sent from one station to multiple stations must be replicated by a third party which is used as a reflector, and multiple copies of the same information are sent through an individual communication channel. The amount of transmission delay for a large group is significant. The IP multicast communication protocol allows transmission of an IP datagram to a group of recipients. It uses the connectionless User Datagram Protocol (UDP). Each member in a multicast group must have a unique Class D address in the range from 224.0.0.0 to 239.255.255.255, whereas in unicast, class A, B, or C type IP addresses are used [34]. The multicast message sender does not have to be a recipient group member [30]. However, a packet is not guaranteed to reach all members of a group since the multicast datagram's reliability is the same as that

of the unicast IP datagram.

Multicast is different from broadcast, which allows a transmitted message to reach all hosts on a cable line, in that it sends messages to a selected group. Only the members in the group can receive the messages, while others discard the messages at the link layer level. However, any host can join or leave a multicast group at any time. A group member has no control as to who joins the group, so an unwanted group member can easily intercept a message transmitted across a multicast group. To establish secure communications, the group members need to share a common key created by a trusted third party. Using the common key, the group members encrypt the message so that no party outside the group can access the message. What is needed is a mechanism to distribute the common key.

Multicast key distribution must be combined with group access control. Without group access control, there is no point in distributing group keys since everyone is a legal group member and sensitive information cannot be protected. Group access can be limited by permission certificates or an access control list. A group access controller should establish a scheme to identify a group member. This group controller can be either a third party or a group member.

## 2.5 Multicast Security Protocols

Since security is a critical issue for multicast, building a trusted multicast protocol is important, We will examine a number of protocols.

### Key Distribution Systems

Key distribution systems are summarized in [9]. There are four models for key distribution systems: star-based systems, tree-based systems, broadcast systems, and cyclic systems [9]. The star-based systems use a DH exchange to distribute a private group key[33] and many protocols with a centralized KDC (e.g, ISAKMP). The protocol described by Wong [33] provides a mechanism in a public key distribution system using DH exchange to negotiate session keys.

Several broadcast systems are also proposed [17][18][23]. One uses a third party as an authentication authority to build a key-distribution protocol using polynomial interpolation and secure keyed one-way hash functions [17]. Others [23][10] use different mathematical models to assist key distribution.

A tree-based and scalable system - Scalable Multicast Key Distribution Protocol (SMKD) that utilizes the multicast routers of the Core Based Tree (CBT), is presented by Ballardie [8] The core based tree multicast protocol uses a tree-based model. SMKD employs a public key infrastructure so that each group member has a public, private key pair. The leaves of the tree are group members and the interior nodes are secondary routers. The group initiator provides a primary core

router with a signed access control list. The primary core establishes the security association parameters, SPI, the Group Traffic Encrypting Key (GTEK), and the Group Key Encrypting Key (GKEK) for the group. The group information is distributed by the primary core router to the secondary router. The key management messages are signed by the primary core router and encrypted in the receiver's public key. The group information includes the ACL, SPI and GTEK/GKEK. A join request from a member will propagate through the routers to the primary core router, if the router does not have the group information. Then the request propagates to the end router. Routers in the delivery tree must be authorized. Each end router verifies the member's permission against the ACL and if it is valid, it sends back group information encrypted in the member's public key.

One problem with many key distribution models is scalability. One solution to scale multicast group is to divide the large group into smaller groups, using an IP router to be the intermediate link layer dividers that can recognize IP protocols and do filtering (i.e, CBT protocol [8]).

However, CBT is a network layer protocol, so it relies on the underlying network architecture. There are limitations associated with CBT. First, all routers must be authorized because they can be compromised as well (i.e, each host needs to establish SAs with the core router and secondary routers). Second, SMKD does not provide perfect forward secrecy by encrypting group traffic keys with the receiver's public key. The compromise of the corresponding private key will

compromise the established private session and all previous sessions.

## **2.6 GKMP Objectives**

As already shown, the SMKD protocol, which is a multicast key management protocol, is scalable, but requires trusted router resources. Therefore, the protocol is not portable when compared to application layer protocols such as GKMS. (However, GKMP was originally designed for network layer, but can be implemented in application layer.) As mentioned earlier, unlike GKMP/GKMS, SMKD does not provide perfect forward secrecy. The other IPv6 based group key management protocols (e.g, SKIP, ISAKMP) do not specifically address group key management. As previously said, group key management and resolving scaling problems are two of the motivations for creating the Group Key Management Protocol (GKMP) [20]. GKMP is proposed for secure multicast communications and can be applied to Internet multicast applications.

### **2.6.1 Features of GKMP**

The GKMP protocol is sender-oriented, which allows the higher level party or authority to initiate group command messages. The network layout is shown in Figure 2.2.

The proposed GKMP architecture supports multicast and is scalable. The protocol delegates the responsibilities of a centralized KDC to a number of network

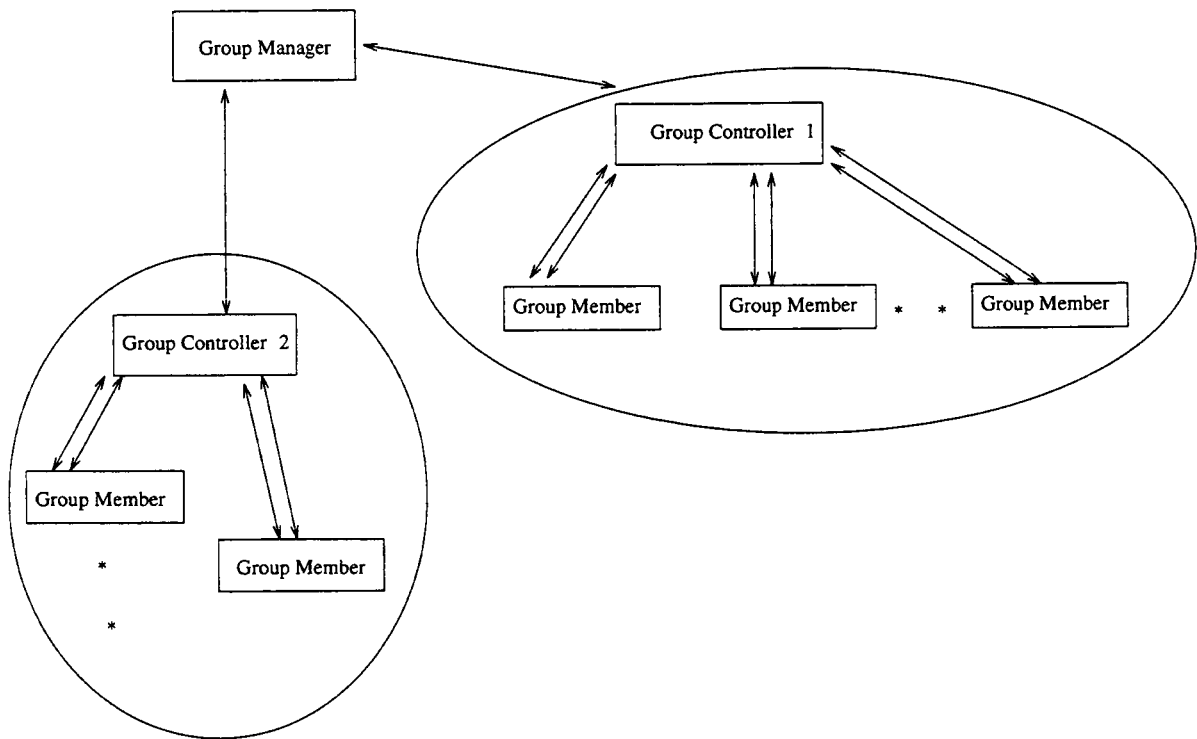


Figure 2.2: Network layout for GKMP

entities: the group manager, group controller, and group member. Any group member has the potential to be chosen as a group controller and to command a group. Details of the responsibilities of each group member are in Chapter 3.

### **2.6.2 Attacks that GKMP Guards Against**

GKMP guards against man-in-the-middle attacks by authenticating DH public exchange values. GKMP provides perfect forward secrecy by using the DH public key exchange to establish group key packages. With the use of the DH public key mechanism, the group controller and a member of a group will create a new group key package independent of the previous package. In the event that a compromise occurs, no information on previous keys can be derived.

This chapter has looked at the problems in multicast security, public-key management systems, and the related protocols. In order to show why GKMS is needed, Chapter 3 will discuss the GKMS architecture in greater detail.

## Chapter 3

# GKMS Architecture

The Group Key Management Sample application protocol (GKMS) utilizes IP multicast to manage group keys and member communication. GKMS provides the ability to create and distribute group keys securely and efficiently. The unique features of GKMS are: providing key management that is invisible to the application, not relying on a centralized key distribution center, synchronizing keys among group members, and using multicast communication protocols. It also combines a secret key and public key infrastructure to help aid the key creation and distribution process. The design of GKMS is based on Group Key Management Protocol (GKMP) architecture [20] and specification [21].

The architecture of GKMS preserves the basic operations and features of GKMP. Essentially GKMS is a more practical and efficient version of GKMP. The access control of GKMS is based on an access control list instead of permission

certificates, and no Certificate Revocation List (CRL) is created or distributed in GKMS. This chapter presents the GKMS/GKMP operations and describes how GKMS differs from GKMP.

### 3.1 GKMS Protocol Overview

The GKMS protocol has two major functions: security management and group controller management.

Security management in the GKMS protocol plays a very important role. It governs group membership access. Its responsibilities include group startup, group member access permission, and compromise recovery. It provides GKMS with a group membership list and relevant control commands. It also provides security, permission management, and member join initialization services. Security management in GKMP creates credentials that uniquely identify the host and its permissions. (see Figure 3.1). During the permission management phase, the *security authority* provides each host on the network with a permission certificate signed by a security authority, which is used as identification during group communication.

The *group manager* is responsible for security management. In the event that a group member is compromised, the group manager will take control to eliminate that compromised group member. Group to member mapping is based on a list of

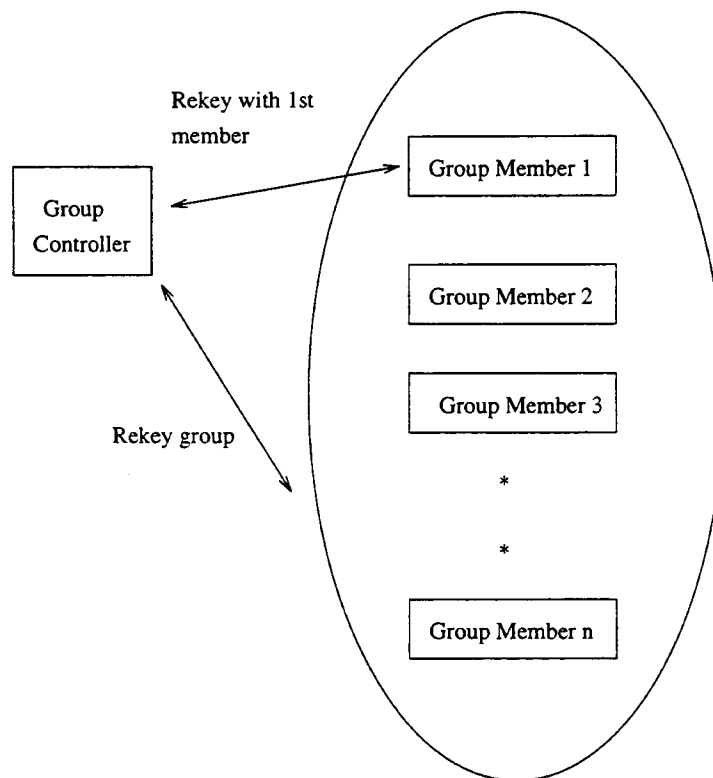


Figure 3.1: Group rekey

valid host addresses which are provided at the group manager initialization stage. GKMP group controllers should also know the valid host addresses.

The *group controller* is a group member with the authority to perform group key creation, distribution, rekey and group recovery functions. The group controller also helps its cryptographic group to maintain key synchronization. All members of a group should share identical symmetric keys. In order to keep track of the group keys over time each key has a version number and lifetime associated with it. If a group member loses track of the current group keys, it will not be able to decipher future information received through the multicast channel. Ambiguity is avoided by allowing only one key controller per group.

A group member may assist a group controller in generating group keys and rekeying a group. A member not only authenticates the group manager and the group controller, but also requests the group keys from the group controller and then communicates with the rest of the group members.

The GKMS protocol delegates the access control, key generation, and distribution functions to the communicating entities, not relying on a centralized key distribution center. It assumes that a pre-existing group manager or group manager is responsible for group membership verification and authentication of identity. Joining members first connect to the group manager to receive permission to access the group.

## 3.2 GKMS Features

The features of GKMS include the supporting of multicast, portability, shorter latency (in comparison to centralized KDC based protocols), extensibility, and reliability. These features of GKMS ensure an advantage in scalability and efficiency.

- Support of multicast

The use of the IP multicast reduces network traffic but introduces problems with security and reliability. Verification on the message sequencing and checksums are needed to provide the reliability. Encryption protocols can be used to provide privacy.

- Portability

GKMS is implemented in the application layer and does not require any special kernel services or privileges. It allows any host on the network to create and manage a secure group, which increases its flexibility.

- Latency

The latency is defined as “the time to set-up or tear down or to re-key a group” [20]. Since a group controller (excluding the group manager) is only responsible for setting up and managing its own group, the latency is reduced. GKMS has essentially divided the responsibilities of the traditional

centralized group key distribution center into two parts: group controllers to handle group creation and a group manager to handle the authentication and permission process.

- Extensibility

By avoiding a centralized KDC, GKMS eliminates the performance bottleneck at the KDC, therefore allowing more groups to co-exist, making the protocol more scalable.

- Reliability

By eliminating a centralized KDC, the system's reliability increases. A centralized KDC can lead to single point failure of the system. With a distributed KDC, the applications using GKMS do not depend on a single KDC and will not be affected by the failure of any other application.

### **3.3 GKMS Operations**

Six basic operations are provided by the GKMS functions. First of all, GKMS uses sender-initiated operations, that is, the sending application has some control in the key distribution and group communication. Second, during the DH exchange, mutual authentication is performed between a group manager and group members, a group controller and its group manager, and a group controller and its group members. Third, GKMS does key creation. The group key management

application selects a group controller and a member of a group to create a group-key package. A group-key packet (GKP) includes a Group Traffic Encrypting Key (GTEK) and a Group Key Encrypting Key (GKEK). GKEK is used to encrypt the new keying material in a rekey process, and GTEK is used to encrypt the group communication messages. Fourth, after a GKP is created, the group controller distributes it to the rest of the group members. It first creates a Session Key Package (SKP) with each group member by computing the DH shared secret. It then distributes the GKP encrypted in the SKP. Fifth, each key has a lifetime associated with it. At the end of its life cycle, a rekey command is initiated by its group controller. The controller selects a group member to create a new GKP and multicasts the new GKP to the rest of the group members. The sixth functionality is group member deletion. When a group member is compromised, the network manager informs the group manager. The group manager then informs the group controller who then initiates a new group session excluding the compromised group member.

In order to form a cryptographic key structure and secure the shared structure among the group members, the protocol needs to securely create, establish and distribute the key. To accomplish this, the protocol must be able to establish groups, delete group members, and perform group rekey.

### 3.3.1 Group Establishment

The creation of a group occurs under the supervision of the group manager. The group manager must identify and authenticate the joining host as part of the group start-up. It first sends out a signed “group creation” command to a targeted group controller. The group controller validates the command and also the group manager’s authority and identification. Next, the group controller creates a key pair with the help of the first group joining member and then distributes this key to the rest of the group members (see Figure 3.2). The controller and group members must mutually authenticate each other’s identity prior to the key distribution.

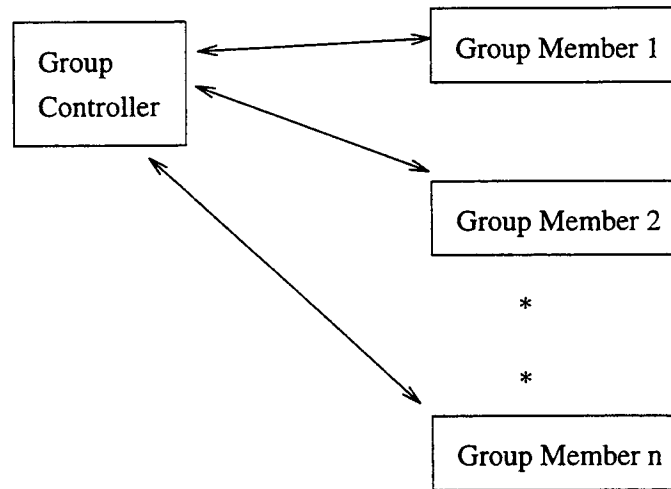


Figure 3.2: Group key creation and distribution

High grade cryptographic keys are created using a random number generator. In creating these keys, the validation of the authority is required to secure the key creation partner. This is done by digital signature using the public key in-

frastructure. Symmetric-key encryption is also used in the group-key distribution process. A unique shared secret key is negotiated the DH between the group controller and a new joining member at the initialization stage, which is prior to the key distribution.

The maintenance of keys is also important. Each key has a definite lifetime. Before a key expires, the group controller needs to generate a new key and distribute it among the group members. During this process, identification and authentication of the membership is assumed, since it was done in the member join stage. The new group key is created in the same manner as in the group establishment using a DH with the first member. The new key is then multicast to the groups.

### **3.3.2 Group Member Deletion**

There are two different types of member deletion. They are cooperative deletion and hostile deletion. Cooperative deletion occurs when a group member leaves voluntarily or is assigned to other tasks. It is no longer a part of the network and in this event the group controller does not need to take any action. In the event of hostile deletion, where a group member is likely compromised or not trusted, compromise recovery is needed to secure the multicast session. To initiate the member deletion process, the network administrator sends the group manager a message that contains the address of the compromised group members. Next, the

controller generates a random string as the new GTEK, and encrypts it with each member's secret key, then sends it to each member. Using the new GTEK, the group controller excludes the compromised member from the multicast communication.

### **3.3.3 Group Rekey**

Group rekey is another important procedure to protect the group communication. Depending on the key's lifetime, the group controller initiates a rekey message to the first joining member periodically. It is a duplication of the group creation process. The new key is then multicast to the rest of the group members. A copy of the previously established key is also kept in the system.

## **3.4 GKMS/GKMP Differences**

The distribution system of GKMS/GKMP eliminates the key exposure by distributing the key only among current group members. Since there is no centralized KDC, GKMS/GKMP is scalable, and since the group size is changeable, group members can join and leave groups. Any application can start up a group and organize it. Therefore, the group itself is relatively autonomous. Performance cost is less expensive in comparison with a centralized model.

Both GKMS and GKMP use the UDP communication protocol required for multicast. However, the individual management communications can use either

Transmission Control Protocol (TCP) or UDP. The GKMS uses TCP communication protocol instead of the UDP connectionless protocol proposed in GKMP. TCP protocol provides message sequencing and integrity checks. It guarantees a message to be received correctly. The UDP protocol does not provide that guarantee. An application that employs UDP needs to timeout and retransmit packets if necessary. Using the TCP protocol, the reliability is guaranteed.

The access control of GKMP is based on signed permission certificates. Each group controller keeps a list of valid signed permission certificates. The permission management is done at the group join stage. Each member holds certificates signed by the group manager. Based on these certificates, GKMP manages its security operation. If someone is compromised, a new security group will be created. To exclude a member from a current group, the group manager will insert the host into the list of compromised hosts, called the CRL, and distribute it across the network. The group controller is responsible for reviewing the CRL and taking action when a group member is compromised. GKMS uses a simpler group access policy, and does not require CRLs or permission certificates. Rather, the access control is based on the signed IP addresses ( signed by the group manager).

### 3.5 Unresolved GKMP Issues

There are several issues to be resolved in the GKMP protocol. They are access control and error recovery.

Since the GKMS/GKMP protocols avoid a centralized authority model, group access control is distributed to group controllers. The GKMS/GKMP protocols have a distributed controller to control the group management. However, each distributed controller may have a large number of clients and could be a performance bottleneck.

Many possible error conditions arise with the use of this protocol. A designer of GKMP application should always assume that the communication channel can be broken due to interference from the Internet. If unicast group management messages are delayed or corrupted during transmission (especially a problem when using UDP transmission protocol), the protocol should provide recovery mechanisms to retransmit messages. GKMS uses the TCP protocol to avoid this problem.

Other unresolved issues associated with both GKMS and GKMP are how to choose a group controller, how to discover the compromised group member, and how to elect a new controller.

This chapter has reviewed the GKMS architecture, operations and features, and some general design issues were also discussed. In Chapter 4, an implementation of GKMS and related issues will be described.

## Chapter 4

# Group Key Management Sample (GKMS) Protocol

The purpose of this implementation of GKMS is to test the design and implementation of GKMP. GKMP, with the proposed protocol architecture, has many unresolved design issues such as group controller rekey and controller election. It is hard to evaluate the scalability of GKMP without a concrete design solution. This implementation of GKMS preserves the basic features of GKMP with minor variations with respect to the GKMP specification. The differences are:

- GKMS simplifies the message negotiation process during key creation, group rekey, and key deletion.
- Access control is based on digital signatures instead of signed certificates.

- GKMS does not implement the Certificate Revocation List (CRL).

The GKMS implementation integrates some of the basic cryptographic techniques such as encryption/decryption and digital signatures. The implementation is based on the GKMP specification [21]. An additional Application Programming Interface (API) is implemented for use with future GKMP applications.

## 4.1 The GKMS software

The GKMS demonstration application is a group-talk program using the multicast communication protocol. A server program is used as a group manager (or security manager) which allows clients to join the group. A separate client program requests a connection from the server program and then is able to communicate with the rest of the group. The group manager is executed as follows

```
talks <my machine> my_port_number.
```

Then the user is prompted for the number of clients to start. Two is the default input in order for GKMS to initiate the key exchange.

`my_port_number` is used by the group manager to open a corresponding socket.

On the other end, group members and group controllers execute

```
talkc <server machine> server_port.
```

A group controller is identified by its Internet address to establish a connection.

The `server_port` should be identical to `my_port_number` in the server program.

The underlying key exchange is invisible to users. The key management functions are tightly coupled into the application (another API solution will be discussed later). However, a user must have a RSA public and private key pair (using *ssh-keygen*) in order to access a group.

A typical run of the demonstration application:

```
A member on eclipse.ns.utk.edu
```

```
UNIX> talkc raptor.ns.utk.edu 7777
```

```
server: qing
```

```
Go ahead and talk
```

```
The controller on raptor.ns.utk.edu
```

```
UNIX> talkC raptor.ns.utk.edu 7777
```

```
server: qing
```

```
Go ahead and talk
```

```
The security manager on raptor.ns.utk.edu
```

```
UNIX> talkS raptor.ns.utk.edu 7777
```

```
Input number of clients:2
```

```
Enter management command
```

```
--1 for deletion
```

The program has also tested with more than two members.

## 4.2 The Basic Structure of the Program

This program uses TCP for unicast messages and UDP for multicast messages. It uses Internet domain sockets on Solaris systems to send and receive communication messages. The server and client programs can be distributed across different host architectures since they convert integers to network byte order before packing them into a buffer. The receiver on the other side will disassemble the buffer and then convert the integers into the local host byte order.

### 4.2.1 Server Function

The server function is executed by the group manager (also security manager) and it should be executed before the client function. The server function first checks

if input arguments are valid, then it loads the configuration file from the local directory, which is owner-readable only. The configuration file (see Appendix B) contains a list of the network addresses of hosts that are permitted to join the group. The list of network addresses is used as an access control list (ACL) by the group manager, which also acts as the security manager.

The group manager reads the clients' joining messages and verifies their authenticity. It also checks the member's identification, which is the signed IP address provided by the sender, against the access control list and the group ID in the request. It then signs the member's IP address. If any mismatch occurs, it denies service to the joining member. It also guarantees that the first accepting member is the controller. If a member contacts the group manager before the controller, the group manager will deny service to this particular member until the controller is up and running.

To initiate a secure group, two members of the group, including the controller, should join at the initial stage. The group manager then reads its private key file, and the public key files sorted by user name (for testing purposes, it can be a host name). It then sends the controller and the first joining member signed initial connection messages. The controller is defined in the configuration file (see Appendix B). It then sends the second joining member a message that the controller has joined, then passes the new group's management control port number served by the controller. The message is signed by the group manager. This process

allows the group manager to manage multiple group controllers and their groups.

After the group controller is up and running, the group manager continues accepting new members on a fixed socket. It uses a *select()* call to poll for clients requesting to join the group. For each member who wishes to join, the group manager verifies its permission and group request, as in the initialization process. Then the member's authorization ID is verified. The group manager then sends the new member a signed message containing the group management port used for the connection to the group controller.

While waiting for more clients to join, the group manager also prompts for possible group administrator commands. It takes input from the keyboard and processes it if it is valid. The only valid command for this sample application is to delete a compromised group member.

#### **4.2.2 Member Functions**

Since a controller can also be a member, the client module combines functions for a group controller and group members. The controller functions verify members and initiate management messages. The member functions can be classified as controller application and client application.

The member functions first check valid arguments, then send the group manager a message claiming its authority and identity as a group member. The member uses its network address as the identity. The message must be signed

with the member's private key. Upon receiving the message, the group manager validates the request as discussed in the server section. It sends back a signed message with the group manager's private key. A client verifies the authority of the group manager. It uses the port number passed from the group manager and the IP address signed by the group manager to establish a connection with the group controller. The first joining member helps the the group controller to establish the group key package. The group key package is then passed along to the rest of the joining members.

#### 4.2.3 Data Structure and Primitives

A group manager keeps a group data structure for access control and synchronization. The structure used in this implementation is:

```
typedef struct mygroup {
int id;                /* group id */
char *GC;              /* group controller */
Dlist grouplist;      /* a list of hosts - ACL */
int port;             /* multicast port, unicast port is one
                        plus the multicast port number */
char *hostaddr;        /* multicast addr */
unsigned char *GKEK;   /* multicast keys */
unsigned char *GTEK;   /* group traffic encrypting key */
int rkitval;           /* rekey interval in seconds */
int level;            /* min security level */
int access;           /* max access area */
} GROUP;
```

The following primitives are built and used during the group management:

- Group access control list

The access control list is implemented with a doubly linked list (Dlist). It holds a list of qualified group members and is kept by the group manager. The group manager uses it as an access control list (read from a configuration file).

- Group controller member list

A group controller keeps a list of active group members. It is implemented as a doubly linked list and used as the controller contacts the members.

- Group ID

Each group has a unique identification number which is a four-byte integer. A joining member is required to provide an existing group ID to join.

- Current Group Key Encryption Key (GKEK)

Unique 16-byte key for the current multicast session.

- Current Group Traffic Encryption Key (GTEK)

Unique 16-byte key for the current multicast session.

- Old GKEK

A 16-byte key from the previous multicast session.

- Old GTEK

A 16-byte key from the previous multicast session.

- Payload

The payload is a fixed formatted data stream sent between group hosts. It has a 1 byte header, a 512 byte message body, a 16 byte checksum of the first 513 bytes, a 128 byte RSA signature, as illustrated in Figure 4.1. Even though RSA signature includes a MD5 digest, it is beneficial to have a single message format.



Figure 4.1: Network data payload

### 4.3 Message Definitions

With the defined payload structure, the key management control messages are transmitted between group members, the group controller and the group manager. Messages sent from the group manager are Command-a-Group (CG) and Joining-Permit (SGID). The controller oriented messages are Create-Session-Keys\_1 (CSK1), Negotiate-Session-Keys\_1 (NSK1), Create-Group-Keys\_1 (CGK1), Negotiate-Group-Keys\_1 (NGK1), Delete-Member(DM) and Rekey-Multicast(RM). The member joining request messages are Client-Joining-Message (CJM), Create-Session-Keys\_2 (CSK2), Negotiate-Session-Keys\_2 (NSK2), Create-Group-Keys\_2 (CGK2) and

Negotiate-Grp-Keys.2 (NGK2). The group member communication messages used by the application are DGM and MG.

- Client-Joining-Message (CJM), sent by members

The joining client first sends a Client-Joining-Message to the group manager signed with the client's private key. The message contains a group ID number, checksum, and member signature.

- Command-a-group (CG), sent by the group manager

When a group member joins the group, its address is verified and compared with the selected group controller address. If the new member is the expected group controller, the group manager rekeys with the Command-a-Group message, allowing the member to take responsibility for initiating a group. The Command-a-Group message provides the multicast port and address. The message contains group address, multicast port, rekey interval in seconds, checksum, and the group manager signature.

- Joining-Permit (SGID), sent by the group manager

Upon receiving a CJM message from a client, the group manager verifies the authority of the sender and sends back an SGID message. The message contains the group controller's address, management port number, checksum, and the group manager signature.

- Create-Session-Keys<sub>1</sub> (CSK<sub>1</sub>), sent by the controller

The message Create-Session-Keys<sub>1</sub> is sent by the group controller to create a session key (SKEK) between the controller and the first group member. The new session key can be used as the new GKEK. The message contains the controller's DH public value, message checksum, and controller signature.

- Create-Session-Keys<sub>2</sub> (CSK<sub>2</sub>), sent by members

Similar to CSK<sub>1</sub>, the function Create-Session-Keys<sub>2</sub> is used also as part of the group key creation with the first joining member. The message is sent out by the member upon receiving a CSK<sub>1</sub>. It helps to create a session key between the controller and the group member. This message contains the member's DH public value, message checksum, and member signature.

- Negotiate-Session-Keys<sub>1</sub> (NSK<sub>1</sub>), sent by controller

Upon receiving the CSK<sub>2</sub> message, the controller computes the shared secret and uses this shared secret (SKEK) to encrypt the current (GKEK, GTEK) to send to the new member. This message contains the GKEK, GTEK, message checksum, and controller signature. Both the message body and the checksum are encrypted.

- Negotiate-Session-Keys<sub>2</sub> (NSK<sub>2</sub>), sent by members

Upon receiving NSK<sub>1</sub>, the member sends an acknowledgement back to the sender. The message contains the group member's address, checksum, and

signature.

- Create-Group-Keys\_1 (CGK1), sent by controller

The message contains the controller's DH public value, message checksum, and controller signature. The message is identical to CSK1.

- Create-Group-Keys\_2 (CGK2), sent by members

The message contains the member's DH public value, message checksum, and member signature. It is sent from the first group member to the controller.

- Negotiate-Grp-Keys\_1 (NGK1), sent by controller

Upon receiving the CGK2 message, the group controller computes the shared secret (GTEK) and uses this shared secret as a seed to compute a new GKEK. It sends the new GKEK, encrypted with the SKEK, (same as the initial GTEK) to the group member. The message contains the GKEK, checksum, and controller signature. GKEK and checksum are encrypted with the SKEK.

- Negotiate-Grp-Keys\_2 (NGK2), sent by members

Upon receiving NGK1, the group member sends this acknowledgement back to the group controller. The message contains the group controller's address, checksum and signature.

- Delete-Group-Member (DGM), sent by the group manager

When a group member is found by the group manager to be compromised, the group manager sends a DGM to the group controller. The group controller then issues the group re-creation message. It also closes the management control channel to the compromised member. The message contains the compromised member address, checksum, group manager signature.

- Delete-Member (DM), sent by controller

Upon receiving DGM, the group controller generates a new group key package and sends out DM messages through the management port to the rest of the group members. The message contains a new group key, checksum, controller signature. The new group key and checksum are encrypted in the members' session keys.

- Group-Message (GM), sent by members or the controller

The message contains application information encrypted in the current GTEK and is transferred between group members. No signature is used in the payload. Recipients of this message are valid group members.

- Rekey-Multicast (RM), sent by controller

Upon receiving NGK2, a controller sends a Rekey-Multicast message through the new multicast port; therefore each multicast member can start passing communication messages to each other. The message contains the new GTEK, checksum, and controller signature.

## 4.4 GKMS Function Definitions

The flow of messages for group creation, key distribution, and compromise recovery are described in the following sections.

### 4.4.1 Group Key Creation

Group key creation is done by two peers, a group controller and its first joining member. This interdependence increases the cryptographic security strength, since both members contribute to the random bits used in key creation.

The key creation process goes through stages (see Table 4.1). Prior to key creation, the group controller obtains a copy of ACL from the group manager via out-of-band means. The first joining member obtains the public key of the controller and authenticates the controller identity. The controller authenticates its joining member's identity by verifying message signatures. It compares the member's IP address against the ACL and verifies the member's group access permission. It then signs the group member's address and sends it back to the joining member.

This process consists of four messages between the controller and its partner. Upon establishing a socket connection with the first member and verifying member's permission, the controller initiates the negotiation. It sends a signed Create-Group-Keys\_1 to the first member, which contains the controller's DH pub-

lic value. The first member verifies the message's signature and then sends a signed Create-Group-Keys\_2 back to the controller. The Create-Group-Keys\_2 contains the member's DH public value. The controller validates the signed Create-Group-Keys\_2 message. Upon receiving the DH public values from each side, each peer computes the shared secret key independently. Finally, the SKEK/GKEK is created.

Upon receiving a Create-Group-Keys\_2 message, the controller uses the new GKEK as a key seed for a random number generator to generate a new GTEK. Next, it sends Negotiate-Group-Keys\_1 to the member encrypted in the GKEK just generated. The first member receives Negotiate-Group-Keys\_1, certifies the identity, verifies the checksum, decrypts the message, and extracts the GTEK. The controller sends Negotiate-Group-Keys\_2 back to the controller, and the group key creation is complete.

Table 4.1: Group key creation process

| controller                                               | joining member           |
|----------------------------------------------------------|--------------------------|
| The creation group command is received by joining host A | ← command create group   |
|                                                          | Create-Group-Keys_1 →    |
|                                                          | ← Create-Group-Keys_2    |
|                                                          | Negotiate-Group-Keys_1 → |
|                                                          | ← Negotiate-Group-Keys_2 |

#### 4.4.2 Group Key Distribution

Each group member must have the group key in order to communicate with the rest of the group. The group key is distributed between a group controller and its joining members, excluding the first member.

The key distribution process consists of four messages between the group controller and its members (Table 4.2). The group controller first creates a session key (SKEK) with the joining member, then encrypts the current [GTEK,GKEK] package in the SKEK and sends this to the member. Then the group controller sends its member a signed Create-Session-Keys\_1, which contains the DH public value. The recipient of this message validates the controller's signature, stores the public key, and builds and sends Create-Group-Keys\_2 message to the controller. The controller validates the Create-Session-Keys\_2 message and then stores the joining member's DH public value. Each peer computes the shared secret independently. The shared secret is the session key (SKEK). Upon receiving Create-Session-Keys\_2, the group controller encrypts the current GKEK in SKEK, builds a signed Negotiate-Group-keys\_1 message, and send the message to the joining member. The joining member decrypts the message, validates its authority, verifies the checksum, and stores the group key. It then sends Negotiate-session-keys\_2 to the group controller to indicate the completion of the key distribution. At the completion of the negotiation, upon receiving the Negotiate-session-keys\_2 mes-

sage, the group controller can process other key management requests, such as rekey, or another key distribution.

Table 4.2: Group key distribution process

| controller                 | joining member |
|----------------------------|----------------|
| Create-Session-Keys_1 →    |                |
| ← Create-Session-Keys_2    |                |
| Negotiate-Session-Keys_1 → |                |
| ← Negotiate-Session-Keys_2 |                |

#### 4.4.3 Group Member Deletion

There are two types of group member deletions: cooperative deletion and hostile deletion. Of these two, only hostile deletion threatens security and needs to be taken care of immediately (e.g, compromise recovery). The GKMS protocol proceeds with the compromise recovery in three steps. Basically, the group controller needs to create a new group that excludes the compromised member. First, the group manager sends the group controller a signed Delete-Group-Member (DGM) message. The controller then validates the message, extracts the address of the compromised member and then sends current qualified members a signed Deleting-Member (DM) message containing a new group key package. The message and checksum are encrypted using each member's secret session key shared with the controller to prevent eavesdropping by the deleted member. This message is sent through the management control channel to each valid member (not to the deleted member) and the controller. Next, group members decrypt the message, and val-

idate checksums and signatures. The new key package is then loaded.

#### **4.4.4 Group Rekey**

Each group keeps an old group key package and a current group key package. The old group key package is initialized to the current group key package. Since each key package has a lifetime, the controller needs to create a new set of keys before the current key package expires. The group manager decides the lifetime of a group key package and passes it to the controller.

A timer is set by the controller to rekey its group before the current key package expires. The rekey operation uses three messages (Table 4.3). The group controller first saves the current group keys into the old group keys. It then computes a new DH public value, signs it, and passes it to the first group member. (If the first member is compromised, rekey will proceed with one of the members in the new group.) The recipient member validates the signature, extracts the DH value, then it sends the controller its new DH public value packed in a signed Create-Session-Keys\_2 message. Next, the controller computes the shared secret based on the DH exchange, and uses the shared secret as the new GKEK to generate a new GTEK. It then encrypts the new (GTEK,GKEK) in the old GKEK, packs it into the Rekey-Multicast message, and signs it and multicasts it to the group members. Upon receiving this message, group members save their old group key package, decrypt the Rekey-Multicast message, verify the checksum, and validate

the controller signature, completing the rekey.

#### 4.4.5 Member Join

The group member join is illustrated in Table 4.4. A new joining member communicates with the group manager first to obtain the correct management port number and the group manager's signature. It then opens the management port, waits for the controller to verify membership (based on the IP address signed by the group manager), initiates a DH session key exchange and receives the current group key package. Two fewer messages (compared to the join operation in GKMP) are implemented in GKMS key distribution process because real permission certificates are not implemented.

Table 4.3: Group rekey process

| controller              | joining member          |
|-------------------------|-------------------------|
|                         | Create-Session-Keys_1 → |
| ← Create-Session-Keys_2 |                         |
|                         | Rekey-Multicast →       |

Table 4.4: Group join process

| controller              | joining member          |
|-------------------------|-------------------------|
|                         | ← Request-Group-Join    |
|                         | Create-Session-Keys_1 → |
| ← Create-Session-Keys_2 |                         |
|                         | Download-Group-Keys_1 → |
| ← Download-Group-Keys_2 |                         |
|                         | Rekey-Multicast →       |

## **4.5 Cryptographic Libraries Used**

The Cryptographic libraries used in the program are the SSH [35] and DES [28] libraries. The SSH library also includes the IDEA and Blowfish [31] encryption/decryption functions; however, they are not discussed in this section since only DES is used in this implementation.

### **4.5.1 DES Library**

The DES [28] library is a portable, efficient and free library of primitives for building cryptographic applications. Since the discussion on secret-key encryption is beyond the scope of key management, interested users can see [31]. The DES Cipher Block Chaining mode [28] is used by this application. This permits the encryption of a message larger than 64 bits and ensures that if the same block is repeated in the plaintext, it will not cause repeats in the ciphertext. The initial vector is an 8 byte long string, and is set to 0. Memory used in this routine is zeroed out right after use to decrease the probability of disclosure.

### **4.5.2 SSH Library**

Secure Shell (SSH) [35] is a program used to log into another computer over a network, execute commands and access file systems. SSH provides strong authentication and encryption, and uses client RSA symmetric key authentication to prevent impersonation (by routing or DNS spoofing) and man-in-the-middle

attacks.

The software can be installed and used (with restricted functionality) even without root privileges. *ssh-keygen* is used to generate user public key pair (However, running *ssh* is not required for GKMS.) A user can create any number of user authentication RSA keys for his/her own use. Each user has a file which lists the RSA public keys for which proof of possession of the corresponding private key is accepted as authentication. User authentication keys are typically 1024 bits.

The SSH protocol library provides many reusable functions. The functions used in GKMS are the encryption/decryption functions, the random number generator and MD5 digest function. It includes cryptographic software such as IDEA and DES.

## 4.6 Comparison with GKMP Specification

The comparison of the features of the GKMS and GKMP specifications are listed in Table 4.5. In the table, “N” means not supported; “Y” means supported. A blank in a table entry shows that the corresponding feature is not specified by the current specification. As the table shows, the GKMS software is portable to UNIX systems with multicast kernel support. However, in order to port this software on a PC/windows platform, the UNIX socket calls need to be changed to winsock calls. GKMS also has limited scalability as a consequence of using the

Table 4.5: Comparison of group key management schemes.

| Feature                | GKMS | GKMP spec |
|------------------------|------|-----------|
| authentication         | Y    | Y         |
| PKI                    | Y    |           |
| ACL                    | Y    | Y         |
| CRL                    | N    | Y         |
| permission certificate | N    | Y         |
| rekey                  | Y    | Y         |
| delete                 | Y    | Y         |
| reliable               | Y    | Y         |
| multicast              | Y    | Y         |
| PFS                    | Y    | Y         |
| portable               | Y    |           |
| scalable               | Y    | Y         |

TCP protocol.

To objectively measure the performance of GKMS/GKMP, it is important to know the differences between the implementation of GKMS and GKMP protocols.

1. In the GKMP protocol, the authors proposed the use of signed permission certificates issued by a “system wide authority”[20]. However, with GKMS, the public-key signature (signed by the group manager) is used as a permission certificate to the controller allowing for greater efficiency.
2. GKMS is scaled down to enhance the efficiency. The access control and permission management in GKMS do not use permission certificates as specified in GKMP. However, with GKMS, the public-key signature (signed by the group manager) is used as a permission certificate to the controller for greater efficiency. The GKMS implementation uses fewer states and messages for

key management negotiations for better efficiency. (The negotiation states used in GKMP during key distribution are not necessary.)

3. The message definitions are slightly different since some message types are eliminated from the original GKMP specification. For example, during group key distribution, the message payload “negotiate session keys” is ignored in the sample application since the key version number and group certification are not implemented in GKMS software.
4. Only a single session key is created in each session. The GKMP architecture requires the session key to have two parts: one for session traffic encryption and another for session key encryption. In the GKMS implementation, only one key which is the session key encryption key (SKEK) [20] is needed. Therefore only one session key is created during the member joining process.
5. The GKMS rekey process also differs from the GKMP specification. Instead of having a group member send the controller a message to initiate a rekey operation, the GKMS implementation uses the controller to send the “create group keys” command to the first group member. (If the first member is deleted, it finds the next available member.)
6. An end user (e.g., administrator) is responsible for the detection of a compromised member, which simplifies the duty of the management function. The user with privilege to access the group manager selects a compromised

group address and passes it to the group manager. Compromise recovery was designed by the GKMS programmer since the GKMP specification did not specify the actual implementation.

7. Some data structures are eliminated from GKMS. The CRL is not used in this protocol since no actual signed certificates are available. It would be more secure to add the CRL or use certificates with short lifetimes[20].

## **4.7 Protocol Evaluation**

The evaluation of GKMS focuses on primarily security and somewhat on efficiency.

### **4.7.1 Some Security Issues Addressed in GKMS**

Several security issues are addressed in the GKMS protocol:

- **Authentication**

The public-key infrastructure provides authenticity for each member. A public key signature at the end of a payload provides authenticity, the receiver verifies the sender's identity with the sender's public key.

- **Privacy**

Message encryption is provided by GKMS. DES, 3-DES, IDEA and blowfish algorithms [31] are implemented in SSH library and are optional in this implementation.

- MD5 checksum

A MD5 digest is used on each message to verify the integrity. It is also a part of the digital signature. Since not all messages contains signatures, to make the data payload consistent, each payload has a digest block attached.

- Random number generation

The GKMS protocol uses functions from the SSH library to create its own random number generator during group key creation. The generator allows users to use the DH shared-secret as a random seed, and use the functions *random\_add\_noise()* and *random\_stir()* to add more noise to the environment. The random generator created with SSH library provides sufficient randomness and speed (as shown in § 4.7.2) in comparison with *truerand()*.

The random number generator used in DH exchange is *mpz\_random()* from the multi-precision library, which uses *random()* from the C Library Functions. This generator is relatively fast (see § 4.7.2) compared to *truerand()* and to the random routine that GKMS uses. However, it is very weak and provides limited randomness.

### **Strengths of the GKMS Implementation**

- GKMS protocol provides secure key distribution. Since each member negotiates a session key with the controller using DH exchange, and the group

key is encrypted with the member's session key.

- Program memory that contains private keying material (e.g., the DES secret key) is zeroed out immediately after use to prevent accidental disclosure by released memory.
- Mutual authentication authenticates both clients and the server.
- The protocol provides perfect forward secrecy, since no long term keys are used in key generation or distribution and since the new generated key is not derived from previous keys.
- The GKMS API (§ 4.8) is scalable and supports multi-group communication simultaneously. The current group management data structure should allow for multi-group management, however, a multi-group application has not been tested.

### **Weaknesses of GKMS Implementation**

- The session key established between a group member and the controller is long term. The compromise of the session key may lead to the compromise of the group since a new group key package (in rekey process) is encrypted in the member's session key.
- The protocol does not prevent a denial of service attack [31]. Since each message type is not encrypted, and message is in a fixed format, an attacker can

replay a signed control message across the net and force the group member or group controller to verify the message and responded to the sender.

- With TCP/IP, only a limited number of file descriptors can be associated with a port. Therefore, on the UNIX platform, there can be no more than 64 group members in a group at any one time.

#### 4.7.2 Performance

Efficiency is measured by the time used for a key creation, distribution, and deletion. The key creation and group rekey processes take the most time. In these processes, most of the time is spent on the DH public value exchange. The speed of the program on a SUN Solaris 5.5.1 system with a 143 Mhz processor is summarized in Table 4.6. Each table entry is presented in microseconds (usec).

Table 4.6: Speed of cryptographic operations

| Operation              | avg CPU time (usec) |
|------------------------|---------------------|
| DH (public gen)        | 18793               |
| DH (compute secret)    | 30770               |
| MD5(513 bytes)         | 698                 |
| sign(513 bytes)        | 439819              |
| verify(513 bytes)      | 8376                |
| encryption (DES)       | 492                 |
| decryption (DES)       | 490                 |
| GKMS random (16 bytes) | 30533               |
| truerand(4 bytes)      | 221131              |

They present average CPU performance. “GKMS random” refers to the random number generator designed for group traffic key generation. As the table shows,

it costs 1908 *usec* per byte, and is faster than *truerand()*, which is 55282 *usec* per byte, but slower than *mpz\_random()* of the multi-precision library which is used in the DH exchange, and costs 61 *usec* per byte.

One may also notice that verifying of a signature takes less time than signing. The major reason is that the public key of RSA is a smaller structure (6 bits) than the corresponding private key (1024 bits). Since time used to encrypt/decrypt a message depends on the key size, verifying with a public key uses less time than signing with a private key [24]. Another reason is that the MD5-hash computation is not required in this version of verifying since the integrity check (always done before verifying) provides a MD5 digest. Users should expect, on average, 2.4 seconds for the group key creation and group key distribution, 1.8 seconds for the group rekey.

## **4.8 Miscellaneous Issues Associated with the GKMS Software**

Besides security issues, a few other issues are important to this implementation:

1. SSH public and private key pair:

A local host should have a public and private key pair generated by *ssh-keygen*. The group controller and group manager should have the public keys of all members. The unicast management messages use TCP/IP, the

multicast messages use UDP. With the TCP/IP connection protocol, each transmission is acknowledged, and if a message sent between two peers damaged or lost, the TCP layer will initiate a retransmission, which guarantees delivery [12]. This simplifies the implementation of GKMS and improves its reliability.

2. Compromise detection:

How to discover a compromised member is another important issue in group management, and it remains unsolved in this protocol.

3. Controller compromise:

If a controller is compromised, then the group manager needs to select another group member as the controller, and then proceed with the controller deletion. Selecting an alternate controller remains an open problem.

4. Security of an operating system:

The security of the cryptographic system also depends on the implementation and security of the underlying operating system.

5. Cryptographic application programming interface (API):

The GKMS API is a portable cryptographic tool for UNIX. It provides a mechanism for enabling GKMS enhanced applications. The current API modules are used for member applications. A member can not be a controller

with this API, both a separate group controller and a group manager must be executed prior to use of the application. The application architecture is showed in Figure 4.2. Another set of functions are implemented in combination with the SSH library in order to access information at the API layer. All functions are coded in C. Interested users may send requests for source code to [cao@cs.utk.edu](mailto:cao@cs.utk.edu). (Distribution is restricted to U.S exportation[31].)

The data structures are included in the *gkmp\_API.h*. Designers must include that file in their source files, and also link the object files into their applications. The SSH library should be incorporated with the GKMS interface in designing a GKMS application. Since each payload contain maximum of 512 bytes of data, applications with large data structures should fragment the data before sending it.

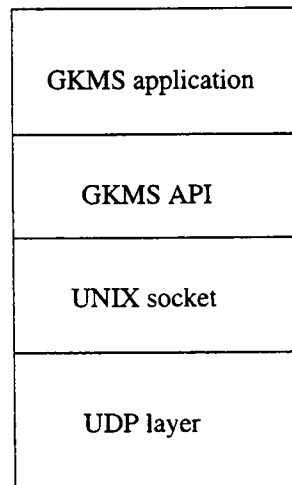


Figure 4.2: GKMS API Interface

The following list briefly describes the functionalities of the API modules. A sample code using the GKMS API is also provided in Appendix C. The underlying function prototypes and details are in Appendix A.

- *GKMP\_join()*

This function initiates a connection to the group manager, and receives a pair of file descriptors for the multicast and management channels.

- *GKMP\_recv()*

This function receives application data from the multicast channel or management channel. Underneath, the function handles key management on the management channel. Key management messages are opaque to applications.

- *GKMP\_send()*

This function sends the application data to the multicast channel.

## Chapter 5

# Future Work

This GKMS implementation is a variation of the GKMP specification. A few differences between the official document by Harney and the GKMS implementation are presented in the Chapter 4, some features may limit the performance of GKMS or GKMP. To eliminate limitations associated with GKMS, some work can be done in the future.

1. A switch from a TCP-based key management protocol to a UDP-based protocol would improve the scalability. (e.g, TCP restricts the group size down to 64.) However, UDP is not reliable, that is, a transmitted packet may be lost, so the implementation would have to provide timeout and retransmission of packets.

2. A large group, such as a group of thousands of members, may also be a potential bottleneck. Multiple group controllers might be deployed, however, the architecture of GKMS does not address this issue.
3. Integrating group key management with the new IP security options would be an interesting research topic [16].
4. Certificates can be used to enhance the security. Since GKMS RSA public keys do not have a certification authority signature, the keys located on a local system may be corrupted. However, there are various formats and conventions for public keys that are signed by different authorities, e.g., the X.509 encodings [31].
5. The Oakley model used with ISAKMP is a good design choice for variable content message. The GKMS implementation uses a fixed length payload to transmit data. Instead of having one big constant size buffer, one can have a few smaller buffers, this not only improves the consistency and flexibility, but also may be more efficient.(i.e., smaller packets)
6. Additional testing over other platforms are needed. The current experiments were done on the SUN Solaris operating system. Future testing should include distributing the application programs to different platforms across the Internet.

7. A cookie exchange method as used in ISAKMP [25] may be an approach to limit denial-of-service attacks.
8. The key-escrow concept has been debated over the past two years. Its intent is to develop a cryptosystem that “both protects individual privacy but at the same time allows for court-authorized wiretaps”[31], which will allow the government to eavesdrop on individuals. From the user’s perspective, there are not any advantages using key-escrow. key-escrow only guarantees that the government can eavesdrop on a user’s conversation. It has considerable disadvantages. For example, the fact that the key-escrow agencies themselves can be compromised. Future GKMS implementations could provide an escrow payload in key distribution messages.

## Chapter 6

# Summary and Conclusions

As multicasting becomes more popular, secure group communication issues become more critical. Traffic integrity, authentication, and confidentiality become important issues. In order to protect multicast communication, cryptographic systems are designed where all members share a common key. Security mechanisms depend on not only cryptographic algorithms used, but also the implementations of protocols (e.g, key management protocols). The security of a cryptographic system relies on the maintenance of cryptographic keys; therefore, key management becomes an important issue for IP secure multicast protocols. Key management responsibilities are key creation, distribution, avoiding key exposure, and detecting group member compromise. The key exchange must be combined with authentication.

Traditionally, the key distribution function is assigned to a trusted third party(KDC).

The centralized KDC entity model burdens a single resource, thereby limiting the multicast capacity. GKMS/GKMP achieves scalable multicast key distribution by using distributed KDC and it passes the load of key management onto the group member(s). It was originally designed for military systems, however, it also has applications in the commercial world. This work presents a sample GKMP application and a cryptographic API. It analyzes the advantages and the disadvantages of the implementation.

Some of the advantages are:

- use of multicast communication protocol
- certain level of scalability
- secure
- distributed model
- supports access control
- efficient

Some of the disadvantages are:

- limited scalability
- use of TCP for individual management connections
- do not support signed certificates.

In conclusion, the GKMS protocol is a group key management protocol that is a variation of the GKMP architecture. It is designed at the application level because IP security protocols [3] are not widely deployed. Due to the level of complexity and security requirements, the GKMP system has never been completely implemented before. Many implementation issues are not addressed in the original documents. This implementation of GKMS application software addresses the design issues of GKMP on a smaller scale, and provides key management solutions to allow privacy, authenticity, integrity, and non-repudiation of communication. The design and implementation of GKMS is based on two criteria : security and efficiency. Only limited assumptions are made for the protocol implementation. The overall system delegates the responsibilities of key management to distributed group members. The deviations of GKMS from GKMP are pointed out in the later section of this thesis. The GKMS application software does preserve the basic structure and features of the protocol, plus minor modifications depending on the underlying protocols used and the needs of the application. A GKMS API library is also available in expectation that this implementation will be beneficial to future application development.

Future work should support certificates and more scalability. Also, exporting protocol applications to other platforms is needed. With the use of the UDP unicast protocol, it may be possible to deploy multiple group controllers to handle a large group. Establishing connections between the GKMS API and the SA in

the kernel under the new IP options may also improve the protocol performance.

# Bibliography

# Bibliography

- [1] "Darpa Internet Program, protocol specification". Technical report, Information Sciences Institute, University of Southern California, 1981.
- [2] R Atkinson. "IP Authentication Header". *RFC 1826*, 1995.
- [3] R Atkinson. "Security Architecture for the Internet Protocol". *RFC 1825*, Aug 1995.
- [4] R Atkinson. "The MD5 Message-Digest Algorithm". *RFC 1321*, Apr 1992.
- [5] A. Aziz, T. Markson, and H. Prafullchandra. "X.509 Encoding of Diffie-Hellman Public Values". *Internet Draft*, Dec 1995.
- [6] Ashar Aziz. "Simple Key-Management for Internet Protocols (SKIP)". Technical report, 1995.
- [7] A. Ballardie and J. Crowcroft. "Multicast-Specific Security Threats and Counter-Measures". *ISOC Symposium on Network and Distributed System Security*, Feb 1995.

- [8] Tony Ballardie and Jon Crowcroft. "Multicast-specific Security Threats and Counter-Measures". Technical report, University of College London.
- [9] Mike Burmester and Yvo Desmedt. "A secure and Efficient Conference Key Distribution System". Technical report, University of London.
- [10] Mike Burmester and Yvo Desmedt. "Efficient and secure Conference Key Distribution". *International Workshop on security protocols*, 1189, May 1996.
- [11] Santosh Chokhani. "Toward a National Public Key Infrastructure". *IEEE Communications Magazine*, Sep 1994.
- [12] Douglas E. Comer. *Internetworking with TCP/IP*. Prentice Hall, Englewood Cliffs, New Jersey, third edition, 1995.
- [13] Stephen Edward Deering. "*Multicast Routing in a Datagram Internetwork*". PhD thesis, Stanford University, Dec 1991.
- [14] W. Diffie, M Wiener, and P. Van Oorschot. "Authentication and Authenticated Key Exchanges". In *Designs, Codes and Cryptography*.
- [15] Whitfield Diffie and Martin Hellman. "New Directions in Cryptography". *IEEE Transactions on Information Theory*, Vol. IT-22:644-654, November 1976.
- [16] Thomas Dunigan and Cathy Cao. "Group Key Management". Technical report, Oak Ridge National Laboratory, Jul 1997.

- [17] Li Gong. "New Protocol for Third-Party-Based Authentication and Secure Broadcast". Technical report, Computer Science Laboratory.
- [18] Li Gong. "On efficient and Secure Broadcasting". Technical report, Computer Science Laboratory, Mar 1992.
- [19] D. Harkins and D. Carrel. "The resolution of ISAKMP with Oakley". *Internet Draft*, Jun 1996.
- [20] Hugh Harney, Carl Muckenhirn, and Thomas Rivers. "Group Key Management Protocol (GKMP) Architecture". *IETF draft gkmp - arch*, 1994.
- [21] Hugh Harney, Carl Muckenhirn, and Thomas Rivers. "Group Key Management Protocol (GKMP) Specification". *IETF draft gkmp - spec*, 1994.
- [22] M. Just and S. Vaudenay. "Authenticated Multi-Party Key Agreement". Technical report, Carleton University, 1996.
- [23] Mike Just, Evangelos Kranakis, Danny Krizanc, and Paul van Oorschot. "On Key Distribution via True Broadcasting". Technical report, Carleton University, Aug, 1994.
- [24] Charlie Kaufman, Radia Perlman, and Mike Speciner. *Network Security: private communication in a public world*. Englewood Cliffs, New Jersey, 1995.
- [25] D. Maughan and M. Schertler. "Internet Security Association and Key management Protocol (ISAKMP)". *Internet Draft - Work in Process*, Feb 1997.

- [26] Douglas Maughan and Mark Schertler. "Internet Security Association and Key Management Protocol". Technical report, 1996.
- [27] NIST. "The Digital Signature Standard, proposal and discussion". *Communications of the ACM*, 35, Jul 1992.
- [28] US National Bureau of Standards. "DES Modes of Operation". *Federal Information Processing Standard (FIPS) Publication - 81*, Dec 1980.
- [29] H. K. Orman. "The OAKLEY Key Determination Protocol". May 1996.
- [30] Pekka Pessi. "Secure Multicast". Technical report, Helsinki University Of Technology.
- [31] Bruce Schneier. *Applied cryptography*. John Wiley & Sons, Inc, first edition, 1993.
- [32] S. Deering and R. Hinden. "Internet Protocol, Version 6 (IPv6) Specification". Technical report, 1995.
- [33] C.K. Wong. "A Conference Key Distribution System". *IEEE Transactions on Information Theory*, Sep 1982.
- [34] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated*, volume 2. Addison-Wesley publishing company, Jan, 1995.
- [35] Tatu Ylonen. "The Secure Shell (SSH) Remote Login Protocol". Nov 1995.  
URL: <http://www.cs.hut.fi/ssh>.

# Appendices

## Appendix A

# Underlying functions of GKMS

## API

- *GKMP\_init(int rekey\_fd, int rekey\_msg\_type, int digestlen, int pcksize)*

Function first generates a DH public and private pair, and sends it to the first member ( wait at file descriptor *rekey\_fd*), User should specify the rekey message type, and len of message portion. The message is checksummed and signed at the application.

- *GKMP\_get\_key(char \*filename, RSAPublicKey \*host\_pub\_key)*

Function extracts public a key *host\_pub\_key* from file *filename*. It first open and reads the file, then initializes and stores the key structure.

- *GKMP\_encrypt(char \*encrypted\_payload, char\* payload, int len, int ci-*

*pher\_type*, *CipherContext cipher\_context*, *unsigned char \* KEK*)

Function first set the encryption algorithm *cipher\_type*, set the key *KEK* then encrypt *payload* with the key.

- *GKMP\_decrypt* (*int cipher\_type*, *char \*dest\_buf*, *char\*src\_buf*, *int len*, *unsigned char \* KEK*)

Function first sets the decryption algorithm *cipher\_type*, set the key *KEK* then decrypt *src\_buf* with the key.

- *GKMP\_random* (*unsigned char \*tmpkey*)

Function generates a group traffic encryption key provided a seed. It first initialize random number generator, then save a random seed, and add noise, stir the random pool to receive a random stream of bytes.

- *GKMP\_rekey* (*int rekey\_fd*, *int reky\_msg\_type*, *int digestlen*, *int pcksize*, *RSAPrivateKey prv\_key*)

Function computes DH public and private pair then send the rekey message of type *reky\_msg\_type* to the first group member at socket fd *rekey\_fd*.

- *void GKMP\_dh\_gen*()

Function generates a DH public and private pair with multi-precision library.

- *GKMP\_getsecret* (*char \*pubval*)

Function computes the shared DH secret providing *pubval* as public value of its partner.

- *char \* GKMP\_sign (char \*tmp,int lth,MP\_INT \* sig, char \*sig\_str, RSAPrivateKey private\_key , int \*return\_len)*

Function signs a string *tmp* and compute the signature of the string. The signature is computed by first compute MD5 hash on *tmp*, then encrypts the MD5 hash. The hash is returned and *return\_len* stores the length of the hash.

- *GKMP\_verify (char \*hash,char \* sig,RSAPublicKey pubkey,int len)*

Function verifies the given RSA signature *sig* by decrypting *sig* with the *pubkey* then compare with *hash*. Function returns 0 if succeed, -1 otherwise.

- *GKMP\_makegrp (int id , int port, char\*hostaddr, Dlist grouplist)*

Function initializes a new GKMP group structure. The new group has group id *id*, multicast port *port*, multicast address *hostaddr*, and a list of possible group members.

- *GKMP\_mcast\_connect (int port, char \*hostaddr)*

Function establishes a multicast connection to the given class D address and port number.

- *GKMP\_packmsg (char type, unsigned char \* message,int totlen)* Function

composes a stream of data of length *totlen1*, with *type* as first byte, the *message* as the body starts at the second byte.

- *GKMP\_send* (*int fd, unsigned char \*tmpstr, int mypcksize, struct sockaddr\_in him*)

Function send a stream of bytes to a given socket file descriptor.

- *GKMP\_recv* (*int fd, unsigned char \*tmpstr, int mypcksize, struct sockaddr\_in from, char \*srcname*)

Function receives a stream of bytes from a given socket file descriptor(*fd*) and extract the claimed ip address.

## Appendix B

# GKMS software configuration format

- first line: A 4 byte integer indicating a group id.
- second line: A 4 byte integer indicating a multicast port number.
- third line: A 4 byte integer indicating the security level for the net host (ranging from 1 to 6), a comma as separator, followed by another 4 byte integer for the group access( ranging from 1 to 6).
- fourth line: a name of the group controller in lower case. (i.e raptor.ns.utk.edu)
- successive lines: List of names of each host on the network, one per line.

## Appendix C

### GKMS C code

```
#include <stdio.h>
#include <stdlib.h>
#define GKMS_PORT 8000

main(int argc, char **argv, char **envp)
{
    char    *hn;
    int      *grpfd;
    fd_set  readfds, masterfds;
    char    buf[512];

    if (argc != 2) {
        fprintf(stderr, "usage: test <hostname> \n");
        exit(-1);
    }

    hn = argv[1];

    FD_ZERO(&masterfds);
    FD_SET(0, &masterfds);
```

```

/* get a pair of fds for grpfd
0 is used for rekey management, 1 is
used for mcast messages*/
grpfd = join(hn, GKMS_PORT);
FD_SET(fd,&masterfds);

for (;;) {
readfds = masterfds;
if (select(getdtablesize(), &readfds, NULL, NULL, NULL)
    < 0) {
    perror("Error on select");
    exit(-1);
}

if (FD_ISSET(0, &readfds)) {
    bzero(buf,512);
    if (gets(buf) != NULL) break;
    GKMS_send(grpfd[1],buf);
}

/* receive some from mcast socket*/
if (FD_ISSET(grpfd[1],&readfds)) {
    GKMS_recv(grpfd[1]);

    if (GKMS_buf != NULL){
        printf("%s\n",GKMS_buf);
    }
}

/* refresh keys */
else(FD_ISSET(grpfd[0],&readfds)) {
    GKMS_recv(grpfd[0]);

    if (GKMS_buf != NULL){
        puts("error!!");
    }
}
}

```

}

### **Vita**

Cathy Xiaoqing Cao was born in Beijing, China. She graduated from a high school in Beijing and travelled to United States for higher education. She received a Bachelor of Arts degree as a college scholar from the University of Tennessee at Knoxville in August 1995. In August 1995, she entered the graduate program in Computer Science at the University of Tennessee, and was awarded the Master of Science degree in Computer Science in August 1997.