The Design and Implementation of a Scalable Secure Multicast System

Zhao Yu Chi

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements for

the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

August 2006

© Zhao Yu Chi, 2006

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-20773-4
Our file Notre référence
ISBN: 978-0-494-20773-4

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Abstract

The Design and Implementation of a Scalable Secure Multicast System

Zhao Yu Chi

Multicast is an efficient way to distribute data to multiple receivers simultaneously. However, security, scalability, and group management issues still prevent the wide deployment of multicast due to the open model of multicast group membership. Our research group has developed frameworks for multicast protocols to solve these problems. Nevertheless, most work was concentrated on one aspect of the whole multicast system, and most of the achievements were architecture design, specification, and validation. In this thesis, a Scalable Secure Multicast System is designed, and a Multicast Data Security module is implemented. The Scalable Secure Multicast System, which integrates two major achievements in our research group, makes great efforts toward providing a scalable and secure multicast system for the real world. The implementation of the Multicast Data Security module proves the feasibility of proxy encryption with multicast and achieves a satisfactory performance in the Linux kernel. A new key distribution scheme based on proxy encryption is proposed to further improve the scalability and security of multicast communication.

Acknowledgements

First and foremost, I wish to express my gratitude to my supervisor, Dr. J. William Atwood, for his expert guidance, continued support and financial assistance during this research and the preparation of this thesis. The comments and time given by Dr. David K. Probst and Dr. Lata Narayanan have greatly improved and clarified this work.

Special thanks should be given to my friends and my student colleagues for their assistance and encouragement.

Finally, my deepest appreciation goes to my parents, my wife and my sister, for their love, encouragement, and support.

Table of Contents

1. Introduction 1

1.1 Motivation 1

1.2 Scope of the Thesis 2

2. Overview of Multicast 3

2.1 Multicast and Group Communication Model 3

2.2 Internet Group Management Protocol (IGMP) 5

2.3 Protocol Independent Multicast - Sparse Mode (PIM-SM) 7

3. Scalable Multicast 8

4. Secure Multicast 13

4.1 Security Architecture for the Internet Protocol 13

4.2 The Multicast Group Security Architecture 15

4.2.1 The Centralized Multicast Security Reference Framework 15

4.2.2 The Distributed Multicast Security Reference Framework 19

4.3 The Multicast Security Group Key Management Architecture 20

4.4 Key Updates 22

4.5 The Proxy Encryptions 23

4.6 The Scalable Infrastructure for Multicast Key Management (SIM-KM) 25

4.7 Key and Security Association 26

5. Building Blocks Used in the Implementation 28

5.1 XORP 28

5.1.1 Overview of XORP 28

5.1.2 Interaction with IGMP and PIM-SM 32

5.1.3 Extension Needed to XORP 34

5.1.3.1 PIM-SM Interface Extension 34

5.1.3.2 PIM-SM Client Interface Extension 35

5.2 The Netfilter / Iptables 37

5.3 The Linux Kernel Module and Data Structures 38

5.4 RSA Algorithm and Proxy RSA 40

5.5 Scalable and Secure Multicast System 41

6. The Design and Implementation 43

6.1 The High-level Design 43

6.2 The Design of the Multicast Data Security Module 49

6.3 Implementation Issues Concerning GSAs 53

6.4 Data Structures Used in the Implementation 54

6.5 The Implementation of the Multicast Data Security Module 55

6.5.1 The Implementation of the Sender Module 58

6.5.2 The Implementation of the Receiver Module 61

6.5.3 The Implementation of the Router Module 63

6.6 Proxy RSA Algorithm and Implementation 64

6.6.1 Implementation Issues. 64

6.6.2 Key Size and Limitations 65

6.7 Additional Key Management and Multicast Testing Tools 66

6.7.1 Multicast Key Management Tools 66

6.7.2 GUI Multicast Sender and Receiver Programs. 68

7. Testing the Multicast Data Security Module 71

7.1 Testing Environment 71

7.2 Testing and Results 72

8. Scalable Key Management 76

9. Conclusion and Future Work 80

9.1 Conclusion 80

9.2 Future Work 81

List of Figures

Number *Page*

Figure 2.1 Unicast and Multicast....4

Figure 3.1 Distribution Trees in a Large Network....9

Figure 3.2 Distribution Trees in a Hierarchical Structure....9

Figure 3.3 Hierarchical Topology....11

Figure 4.1 Centralized Multicast Security Reference Framework....16

Figure 4.2 The Distributed Multicast Security Reference Framework....19

Figure 4.3 Group Key Management Architecture....22

Figure 5.1 XORP Process Model....30

Figure 5.2 Netfilter/Iptables Hooks....38

Figure 6.1 Network Topology....44

Figure 6.2 Sender....45

Figure 6.3 Receiver....46

Figure 6.4 Router (Mesh Node)....47

Figure 6.5 Router (Service Node)....48

Figure 6.6 Packet manipulation technologies....50

Figure 6.7 Module's registration points....52

Figure 6.8 ESP Packet Format....55

Figure 6.9 Flowchart of the Sender Module....59

Figure 6.10 From UDP Packet to ESP Packet....60

Figure 6.11 Flowchart of the Receiver Module....62

Figure 6.12 Flowchart of the Router Module 63

Figure 6.13 Key Message Format 67

Figure 6.14 Multicast Sender 69

Figure 6.15 Multicast Receiver 70

Figure 7.1 Network Configuration 71

Figure 7.2 Logical Network Topology 72

Figure 7.3 Packet Delay 74

Table of Acronyms

AAA	Authentication, Authorization, and Accounting
AH	Authentication Header
API	Application Programming Interface
AS	Autonomous System
BGP	Border Gateway Protocol
BSR	Bootstrap Router
CGC	Control Group Controller
CLI	Command Line Interface
CPU	Central Processing Unit
DATA	Data Security SA
DES	Data Encryption Standard
DGC	Data Group Controller
ESP	Encapsulating Security Payload
FEA	Forwarding Engine Abstraction
GCKS	Group Controller and Key Server
GSA	Group Security Association
GUI	Graphical User Interface
HSPL	High Speed Protocols Laboratory
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
IP	Internet Protocol
IPC	Interprocess Communications

IPsec	Security Architecture for the Internet Protocol
IP-TV	Internet Protocol Television
IS-IS	Intermediate System to Intermediate System
MBone	Multicast Backbone
MFEA	Multicast Forwarding Engine Abstraction
MLD	Multicast Listener Discovery Protocol
MN	Mesh Node
MPE	Multicast Proxy Encryption
NTP	Network Time Protocol
OSPF	Open Shortest Path First
PC	Personal Computer
PIM-SM	Protocol Independent Multicast - Sparse Mode
QoS	Quality of Service
REG	Registration SA
REKEY	Re-key SA
RFC	Request For Comments
RIB	Routing Information Base
RIP	Routing Information Protocol
RP	Rendezvous Point
RSA	Ron Rivest, Adi Shamir and Len Adleman
SA	Security Association
SIM-KM	Scalable Infrastructure for Multicast Key Management
SN	Local Group Services Node
SNMP	Simple Network Management Protocol

SPI	Security Parameters Index
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
XORP	eXtensible Open Router Platform
XRL	XORP Resource Locater

Chapter 1

Introduction

1.1 Motivation

For many years, multicast [1], demanded by the next generation network services, has been attracting many researchers' attention due to its efficiency in data delivery between multiple senders and receivers. However, due to the native open model of multicast, security, scalability, and group management issues still prevent the wide deployment of multicast [2]. The High Speed Protocols Laboratory (HSPL) at Concordia University has developed frameworks for multicast protocols to solve these problems. However, most work is concentrated on one aspect of the whole multicast system, such as managing membership, securing communication, or identifying participants, and most of the achievements are architecture design, specification, and validation. Therefore, we need a system for experiments and demonstrations of the work that has been done. The Scalable Secure Multicast System explores such a system.

In this thesis, a Scalable Secure Multicast System is designed, and a Multicast Data Security module is implemented. The system combines two major achievements in the HSPL, "Hierarchical Topology for Multicasting" [3] and "Proxy Encryptions for Secure Multicast Key Management" [4], so that the system can provide multicast data security for very large groups. This design also follows most of the guidelines described in

RFC3740 [5], the Multicast Group Security Architecture, in order that the demonstration is compatible with standards. The data security module runs in the Linux kernel, captures specified multicast data packets, and performs proper security transformations on these packets using a proxy RSA [6] algorithm. A new key distribution scheme based on proxy encryption is proposed to further improve the scalability and security of multicast communication. Extensions to the eXtensible Open Router Platform (XORP) [7] are proposed. These can be used for future experiments, developments, and demonstrations.

1.2 Scope of the Thesis

The thesis starts with reviewing, in chapter 2, the multicast technologies, including the group communication module, Internet Group Management Protocol (IGMP) [1], and Protocol Independent Multicast Sparse-Mode (PIM-SM) [8]. Then, the approach that was developed in our research group to improve the scalability of multicast is given in chapter 3. In chapter 4, technologies related to securing group communication are introduced. Chapter 5 presents the building blocks used in our implementation, including the routing platform, kernel module, and encryption algorithm. The design and detailed implementation are presented in chapter 6. In chapter 7, the evaluation and performance are discussed. A new key distribution scheme based on proxy encryption is proposed in chapter 8. Finally, in chapter 9, the thesis is ended with conclusions and further work.

Chapter 2

Overview of Multicast

Multicast, also called group communication, provides an efficient way to distribute data to multiple receivers. An overview of multicast and the group communication model is given in section 2.1. In sections 2.2 and 2.3, a local group management protocol, IGMP, and a multicast routing protocol, PIM-SM, are introduced respectively.

2.1 Multicast and Group Communication Model

Group communication [1], as opposed to the classical one-to-one communication, is a way to send a set of data to multiple receivers simultaneously. An increasing number of applications, such as audio-on-demand, video-on-demand, IP-TV, tele-conference, distance-education, multi-player gaming and file update or distribution, inspire the development of multicast technology [9]. Short-term and long-term solutions have been proposed, and a lot of progress has been made in research, implementation and deployment [2].

IP multicast was first introduced by Steve Deering in 1988, and it was first widely applied to "audiocast" at the 1992 IETF meeting [10]. To support multicast on top of traditional networks, the Multicast Backbone (MBone) was created, which consisted of

host-to-host unicast tunnels [11]. With the development of group communication, most newly produced routers naturally support multicast.

In unicast, when multiple receivers require the same data, a sender has to send out multiple copies of data to them one by one, as shown in Figure 2.1(a). However, in case of multicast, the sender sends out only one copy of the data to immediate multicast routers, and the routers in the network forward the data only to the ports that connect to interested receivers, as shown in Figure 2.1(b).

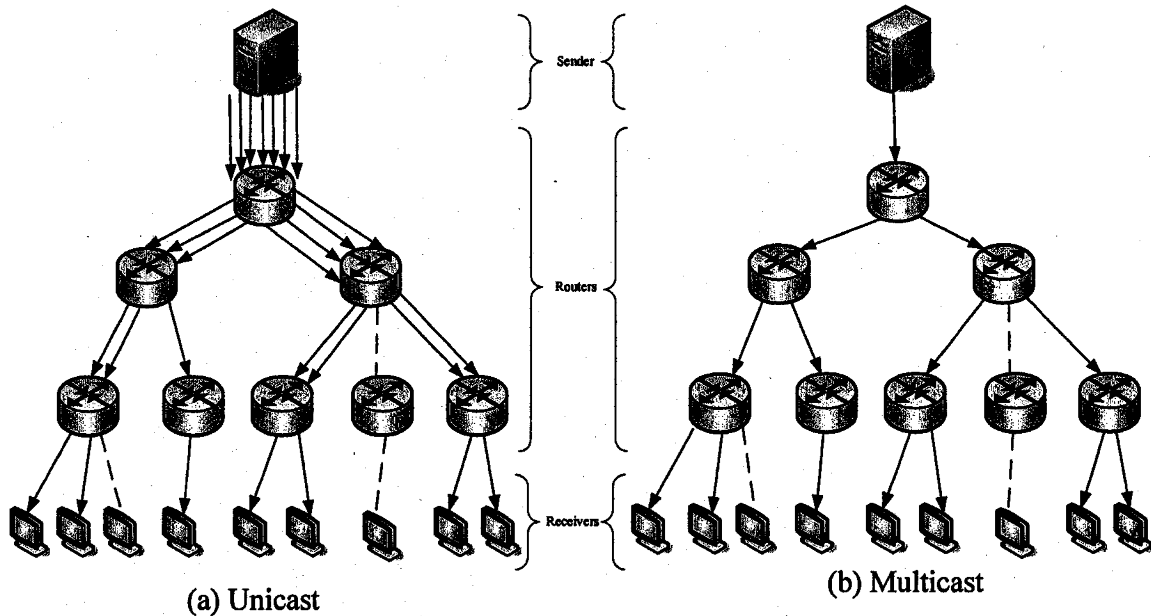


Figure 2.1 Unicast and Multicast

The efficiency of multicast comes from two factors: fewer packets sent from a source and fewer packets forwarded within the network [12]. A considerable amount of time and bandwidth could be saved when the number of receivers is large enough.

Multicast is based on the concept of a group, which consists of an arbitrary number of interested participants. The group is open to everybody, having no physical or

geographical boundaries [1]. Anybody can join in a multicast group, without any authorization; a host can belong to many different groups without any restriction; a source can send multicast data to a group at any time, even though it is not a member of the group; the group is dynamic and one can join or leave a multicast group at any time; the number and identity of group members are known neither to the sources nor to the receivers [1].

A Multicast group is identified by an IPv4 class D address or an IPv6 address starting with "0xFF." Ipv4 class D addresses fall in the range from 224.0.0.0 through 239.255.255.255 [1]. These addresses only appear as destination addresses in IP packets. Unlike a unicast address, a multicast address is dynamically assigned to a group at present [1].

2.2 Internet Group Management Protocol (IGMP)

IGMP [1] is the IPv4 protocol that is widely used to manage the group membership between hosts and their immediate neighbor routers. It is to inform the nearest multicast router in the same LAN about the presence of hosts interested in the traffic sent to a group, so that these edge routers can forward multicast data only to these ports connecting receivers [1]. Even though there is more than one host in the LAN, the router sends only one copy to them. When there is no receiver connected to a port, the router stops forwarding corresponding traffic to the port. Currently, there exist three versions of IGMP.

IGMPv1 specification is described in RFC1112 [1]. Two types of messages, Host Membership Query and Host Membership Report, are defined. Hosts send out the Report message to explicitly join in a multicast group. The destination address of the Report message is 224.0.0.2, indicating all routers on the subnet. Routers periodically send out Query messages to check if there is any host still interested in the specified group. The destination address of the Query message is 224.0.0.1, indicating all hosts on the subnet. If the router has not received any Query response from hosts connected to a certain port for a predefined time, it stops forwarding data to that port. This time-out leave mechanism wastes time and resource [9].

RFC2236 specifies the IGMPv2 [13]. It works the same way as version 1, except adding an explicit Leave message. Hosts can send out a Leave message to indicate leaving the group. After receiving this message, routers query if there is any remaining receiver interested in this group connected to this port. If no replies are received, the router stops forwarding corresponding multicast data. The amount of traffic and wasted time are reduced [13].

The latest IGMP version is v3 defined in RFC3376 [14]. It added support for "source filtering," which enables the receiver to specify which group and source it wants to join or which group and source it is not interested in [8]. The Report message includes two modes: INCLUDE and EXCLUDE. The join and leave group operations can be indicated by sending a message in EXCLUDE or INCLUDE mode respectively with empty source address list. This new feature was added for supporting single source multicast [14].

2.3 Protocol Independent Multicast - Sparse Mode (PIM-SM)

PIM-SM [8] is a multicast routing protocol designed to be deployed in a large scale network where receivers are spread out thinly [8]. PIM-SM is a protocol independent multicast routing protocol, which means that routing decisions are based on whatever underlying unicast routing table exists.

In PIM-SM, routers have to join and leave a multicast group explicitly [11]. A router does not forward multicast packets to downstream routers if they do not send a join message to announce their interest. Per-group Rendezvous Point (RP), as a "meeting place" for sources and receivers, is applied to create RP-rooted shared tree [8]. Each router should have the information about which routers are RPs and the mappings of multicast groups to RPs before communication. This information can be obtained dynamically by the Bootstrap Router (BSR) Mechanism [8]. Routers that want to join a group send an explicit Join message toward the RP of the group. All the intermediate PIM-SM routers on the path toward the RP join the group. A single shared tree, rooted at the RP, is built for each group. The tree is a reverse shortest path tree, because the join messages follow a reverse path from receivers to the RP. Sources have to register with the RP by sending Register messages before distributing multicast data. The intermediate PIM-SM routers on the path join the group and build a delivery path from the source to the RP. Multicast data are sent from sources to the RP, and are distributed from the RP, the root of the shared tree, to receivers. A source-based tree can be created if receivers have shorter paths toward the source than passing through the RP [8].

Chapter 3

Scalable Multicast

The early multicast technologies had limitations in scalability [11]. Many short-term and long-term solutions have been proposed; however various problems still exist [11]. To address these problems, the High Speed Protocols Laboratory (HSPL) at Concordia University has developed a hierarchical topology framework. In this chapter, this framework will be discussed.

In a large network, there may be many large groups, and each group has a lot of group members and many intermediary multicast enabled routers. The senders of these groups may be in different locations, and distribution trees may have to share some of these routers. As an example shown in Figure 3.1, there are three groups, and some receivers are interested in different groups. Some multicast enabled routers, black dots, are in the middle of the network, and shared by different distribution trees. Some local multicast enabled routers, circles, are in the same LANs as receivers. That means that they are directly connected to receivers.

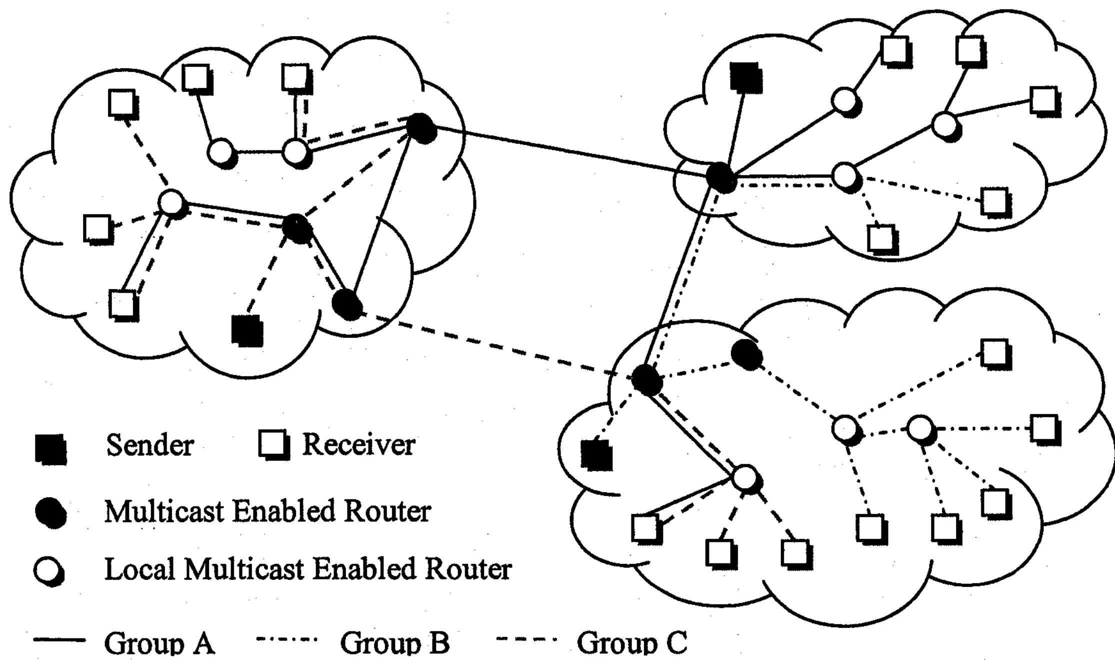


Figure 3.1 Distribution Trees in a Large Network

We can organize the distribution trees into a hierarchical structure, as shown in Figure 3.2. It has four-level structure, including Senders, Multicast Enabled Routers, Local Multicast Enabled Routers, and Receivers

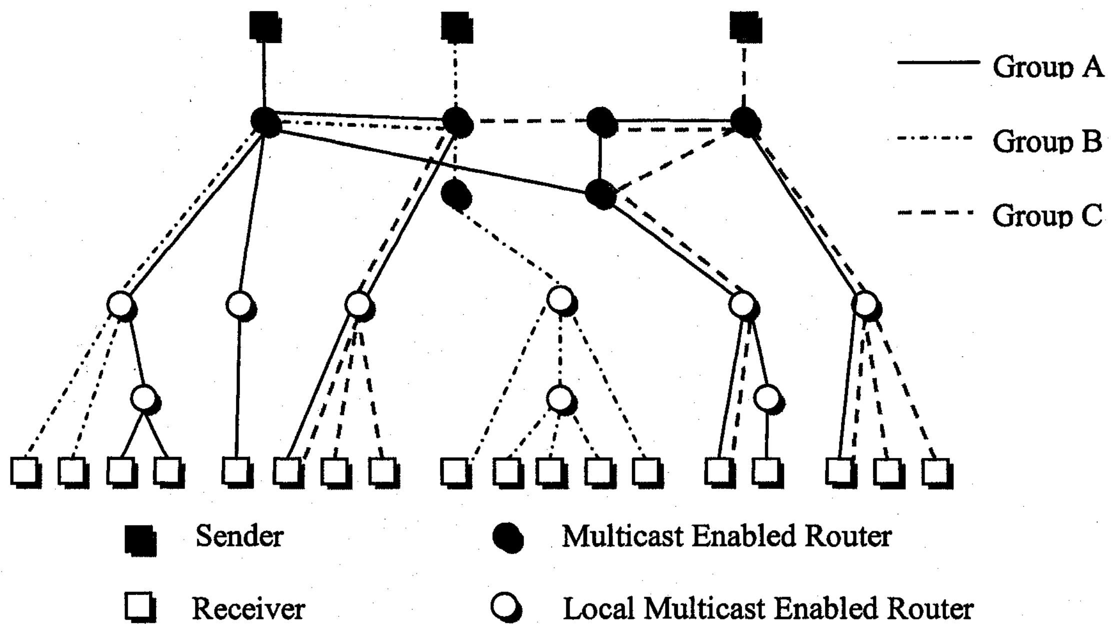


Figure 3.2 Distribution Trees in a Hierarchical Structure

The hierarchical topology framework is based on the previous structure, and consists of a small number of senders, pre-deployed mesh nodes, and a large number of local groups that have a local server node and some receivers in each local group [3].

As shown in Figure 3.3, there are four main components in the framework. Pre-deployed Mesh Nodes (MNs) are the core part of the topology. They can automatically be configured as the root of a group and shared by different groups. A small number of Senders are directly connected to MNs. They can be in different locations. Local Group Services Nodes (SNs) are distributed in the network to control and manage receivers. They are multicast enabled routers or hosts for the networks without multicast routers. On the edge of the topology are a large number of Receivers, which are organized into local groups. Each local group is managed by an SN, which is connected to other SNs or MNs toward a root node. A multicast forwarding tree is rooted at an MN that is directly connected to a Sender, and consists of MNs and SNs as intermediate nodes and Receivers as leaf nodes [3].

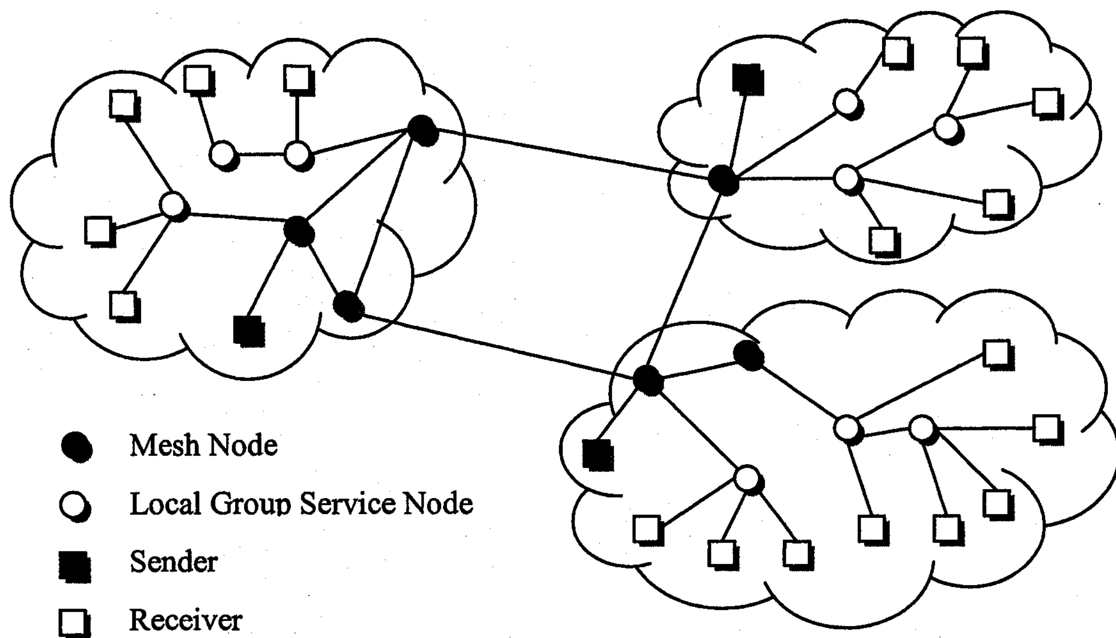


Figure 3.3 Hierarchical Topology [3]

The Mesh approach was originally proposed in the IETF draft, named "Reliable Multicast Transport Building Block: Tree Auto-Configuration" [15]. It has some advantages over traditional approaches, such as shared tree and source-based tree [3]. A fixed root and an additional backup root of a forwarding tree required in traditional approaches are not necessary in the Mesh network. With the auto-configure functionality of the Mesh approach, a root can be chosen dynamically if changes occur in the network topology. Mesh Nodes configured in different Autonomous Systems (ASes) can further improve the scalability of a multicast network. However, the original Mesh approach has a limitation that receivers have to be bound with Mesh Nodes. Our hierarchical topology framework eliminates this drawback by introducing the concept of a Local Group [3].

Receivers in a LAN are organized into a group that is managed by a local group controller. This controller, a Local Group Service Node (SN), can be a router or a host if the nearest router is not multicast enabled. The SN is an intermediate node in a

forwarding tree. Receivers are directly connected to the SN, and the SN also can be the parent node of other SNs [3].

Our group has developed the algorithms used in automatically configuring Mesh Nodes and Local Group Service Nodes. The functionalities of the components in the topology are defined, and the node joining processes are also presented. Further information can be found in [3] and [15].

Chapter 4

Secure Multicast

To apply a multicast system in the real world, security is an essential functionality to avoid unauthorized access. In this chapter, current secure technologies are introduced, including the Security Architecture for the Internet Protocol [16] discussed in section 4.1, the Multicast Group Security Architecture, RFC3740, [5] introduced in section 4.2, the Multicast Security Group Management Architecture, RFC4046, [17] described in section 4.3, the Proxy Encryptions for Secure Multicast Key Management [4] summarized in section 4.4, and the Scalable Infrastructure for Multicast Key Management [18] reviewed in section 4.5. Then, section 4.6 ends with a discussion of symmetric key and asymmetric key encryptions in a secure multicast system, and the proposed modifications to RFC3740 and RFC4046.

4.1 Security Architecture for the Internet Protocol

The Security Architecture for the Internet Protocol (IPsec) defines a suite of protocols for securing traffic at the IP layer by using cryptographic technologies [16]. IPsec is designed to support interoperable, high-quality, cryptographically based security communication for IPv4 and IPv6 applications [20]. IPsec provides security services at the IP layer by enabling a system to select the required security protocols, determine the algorithms to use for the services, and put in place any cryptographic keys required to provide the

requested services [16]. IPsec can be applied to protect one or more traffic flows between two hosts, between two security gateways, or between a security gateway and a host.

The set of security services provided by IPsec include access control, connectionless integrity, data origin authentication, detection and rejection of replayed packets, confidentiality, and limited traffic flow confidentiality [16]. Since these security services are provided at the IP layer, they can be used by any higher-layer protocol, such as TCP and UDP.

IPsec uses two protocols to provide secure traffic: Authentication Header (AH) [21] and Encapsulating Security Payload (ESP) [22]. The AH provides connectionless integrity, data authentication, and an optional anti-replay service. The ESP protocol may provide confidentiality and traffic flow confidentiality. It also may provide connectionless integrity, data origin authentication, and an anti-replay service. Both AH and ESP are vehicles for access control, based on the distribution of cryptographic keys and the management of traffic flows relative to these security protocols [20].

These protocols may be applied alone or in combination with each other to provide a desired set of security services in IPv4 and IPv6. Each protocol supports two modes of use: transport mode and tunnel mode. In the transport mode the protocols provide protection primarily for upper layer protocols; in the tunnel mode, the protocols are applied to tunneled IP packets [20].

The concept of a Security Association (SA) is defined in the architecture. An SA is a simplex "connection" that provides security services to the traffic carried by it [16]. It is a

set of policy and cryptographic keys that provide security services to network traffic that matches that policy [5].

4.2 The Multicast Group Security Architecture

The Multicast Group Security Architecture, described in RFC 3740, outlines the security services required to secure large multicast groups [5]. It explains various elements used in the architecture, and defines Multicast Security Reference Frameworks to organize these elements [5].

Two designs of the Reference Frameworks are shown in the RFC for different scalability requirements. They are a centralized design for small groups, and a distributed design for large groups.

4.2.1 The Centralized Multicast Security Reference Framework

The Centralized Multicast Security Reference Framework is designed for a small multicast group. It is a basic Reference Framework, and shows the functional areas, functional elements, and the relationships between them [5].

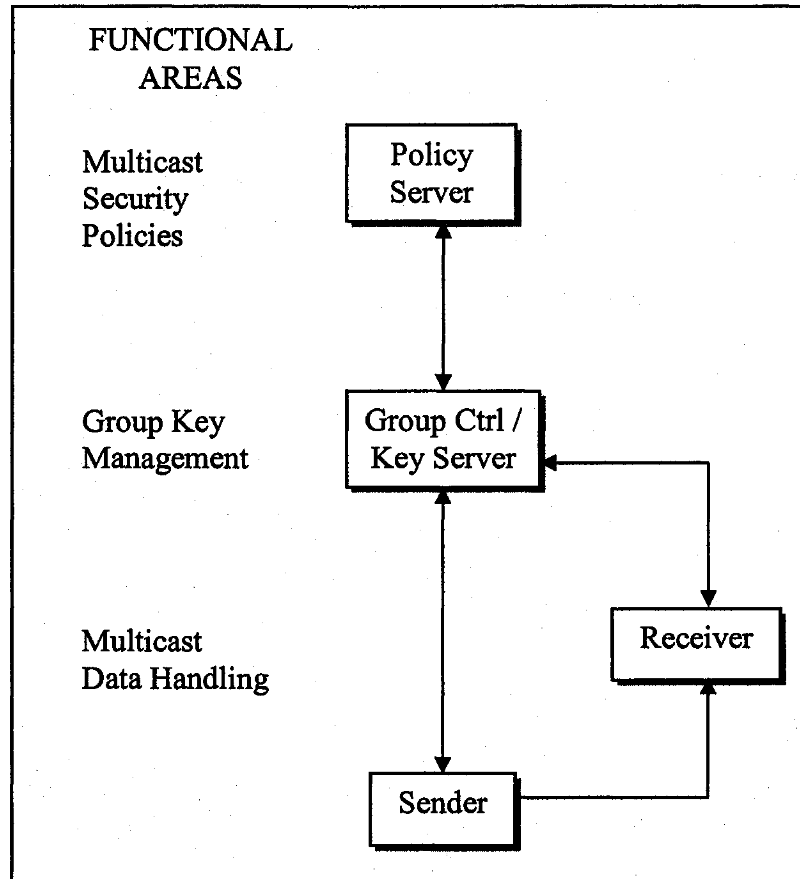


Figure 4.1 Centralized Multicast Security Reference Framework [5]

The blocks in Figure 4.1 are functional entities and the arrows are the interfaces between them.

The functional entities include the Policy Server, the Group Controller / Key Server, the Sender and the Receiver. They are organized into three functional areas, including the Multicast Security Policies, the Group Key Management, and the Multicast Data Handling [5].

The Multicast Security Policies area involves aspects of policy in the context of multicast security [5]. Due to the complexity of multicast communication, Multicast Security Policies should involve more consideration and provide more information than

point-to-point transmission. Open topics still exist in several areas, including policy creation, high-level policy translation, and policy representation [5].

The Group Key Management area covers how to securely distribute and refresh keying material [5]. The keying material includes the keys to a group, the states of the keys, and other security parameters regarding the keys. A Group Security Association (GSA), which is the multicast counterpart of a unicast Security Association (SA), is defined in this Reference Framework, and GSA, which will be described further below, is an essential element for group key management [5].

The Multicast Data Handling area includes the secure treatment of multicast data by the senders and receivers. Typically, the multicast data need to be encrypted and /or authenticated. In this architecture, data encryption can be achieved using a similar scheme to the one used in IPsec. To achieve data integrity and source verification, multicast authentication takes two flavors: group authentication, and source authentication and data integrity. Group authentication aims to guarantee that the data are sent from a node that has the group key. Source authentication and data integrity guarantee that the data are generated by the trusted sender and are not modified by anyone else [5].

Security Associations for group key management are more complex than point-to-point key management algorithms. Group management may require up to three or more SAs; however, point-to-point communication usually uses two SAs for key management and data protection. A Group Security Association (GSA) is a bundling of SAs that

together define how a group communicates securely. A GSA may be a superset of SAs or an aggregation of SAs. Three categories of SAs are defined in GSA [5]

- Registration SA (REG): A separate bi-directional unicast SA between the GCKS and each group member, no matter whether the group member is a sender or a receiver or acting in both roles.

- Re-key SA (REKEY): A single multicast SA between the GCKS and all of the group members. It cannot be negotiated, because it is uni-directional and all the group members must share it.

- Data Security SA (DATA): A multicast SA between each multicast source speaker and group's receivers. It is obtained from the GCKS, and is shared among the group members. There may be as many Data SAs as there are multicast sources allowed by the group's policy.

Working in the Multicast Security Policies functional area, the Policy Server is in charge of the creation and management of security policies for a given multicast group. It incorporates the Group Controller and Key Server (GCKS) entity to install and manage the security policies regarding the membership and keying material for this given multicast group [5].

The GCKS represents both the entity and functions used to issue and manage the keying material regarding a multicast group. It belongs to the Group Key Management functional area. User authentication and authorization checks on group members are

included in GCKS. Group Controller and the Key Server can be implemented in a single element or separated into different functional entities.

The Sender and the Receiver belong to the Multicast Data Handling functional area. They must interact with the GCKS regarding group and key management, including user and/or device authorization, obtaining key material at the beginning, obtaining re-keying updates, getting security parameters and other messages regarding key and group management [5].

4.2.2 The Distributed Multicast Security Reference Framework

The Distributed Multicast Security Reference Framework meets the scalability requirements of a large group that exists across wide geographic regions of the Internet [5].

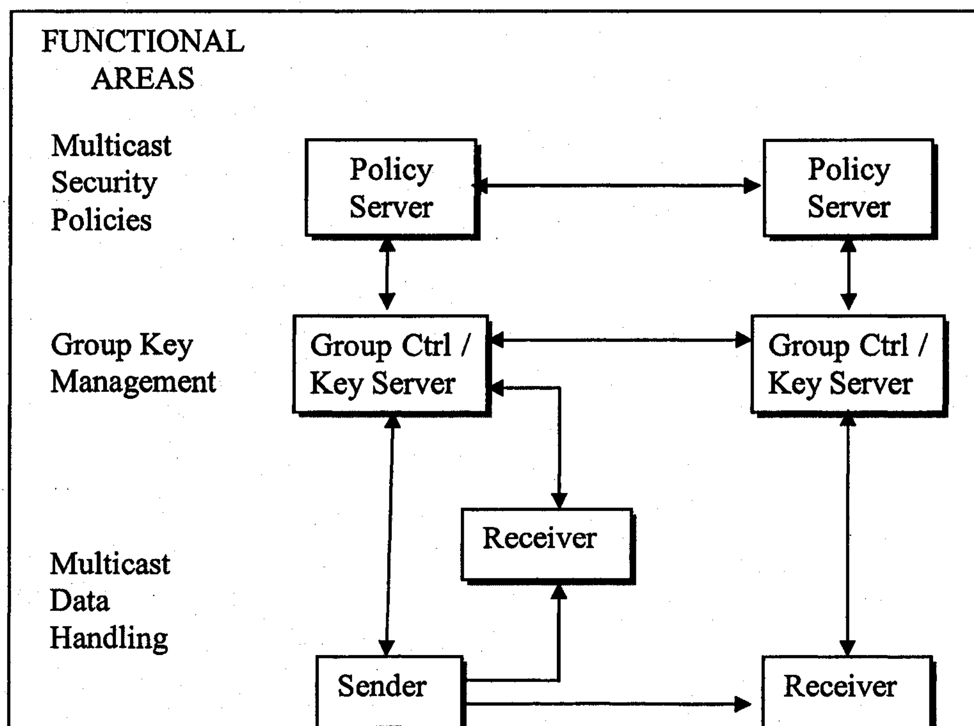


Figure 4.2 The Distributed Multicast Security Reference Framework [5]

As shown in Figure 4.2, Policy Servers securely interact with each other to keep the security policies consistent. Each GCKS can interact with one or more Policy Servers, and all GCKS entities interact with each other to provide the functionality required in the Group Key Management functional area. Senders and Receivers work as in the Centralized Reference Framework, except they may communicate with different GCKS entities [5].

The Multicast Group Security Architecture is "end to end", and it is independent of multicast routing protocols, such as PIM-SM. It also does not require any IP multicast admission control protocols, such as IGMP or MLD. Therefore the group management of this architecture is independent of IP multicast group management [5].

4.3 The Multicast Security Group Key Management Architecture

To support a variety of application, transport, and network layer security protocols, a common architecture for multicast security key management is defined in RFC 4046 [17]. It also defines the Group Security Association (GSA) and describes the key management protocols to help establish a GSA [17]. Frameworks and guidelines given in RFC 4046 provide a modular and flexible design of group key management protocols for a variety of different settings that are specialized to application needs [17].

The main goal of a group key management protocol is to securely provide group members with an up-to-date Security Association (SA). This SA that contains the needed information for securing group communication is called the Data SA [17]. To achieve

such a goal, a Registration protocol and a Rekey protocol are defined. The Registration protocol is a unicast protocol exchanging information between the Group Controller and Key Server (GCKS) and a joining receiver. After the joining member is authorized, the GCKS provides the information for initializing the Data SA and the information for initializing the Rekey SA to the joining member. The Rekey protocol is used between the GCKS and group members to update or change the Data SA. The Data SA may be periodically updated based on the group policy or may be changed in the case where one of the following events happens: the group membership changes, the group policy changes or the key expires. The Rekey protocol can be a unicast protocol or a multicast protocol, protected by the Rekey SA [17].

The overall design of Group Key Management Architecture is shown in Figure 4.3. Each group member, a sender or a receiver, uses the Registration protocol to get authorized and authenticated access to a particular group, its policies, and its keys. The GCKS distributes the Rekey information to members by the Rekey protocol [17].

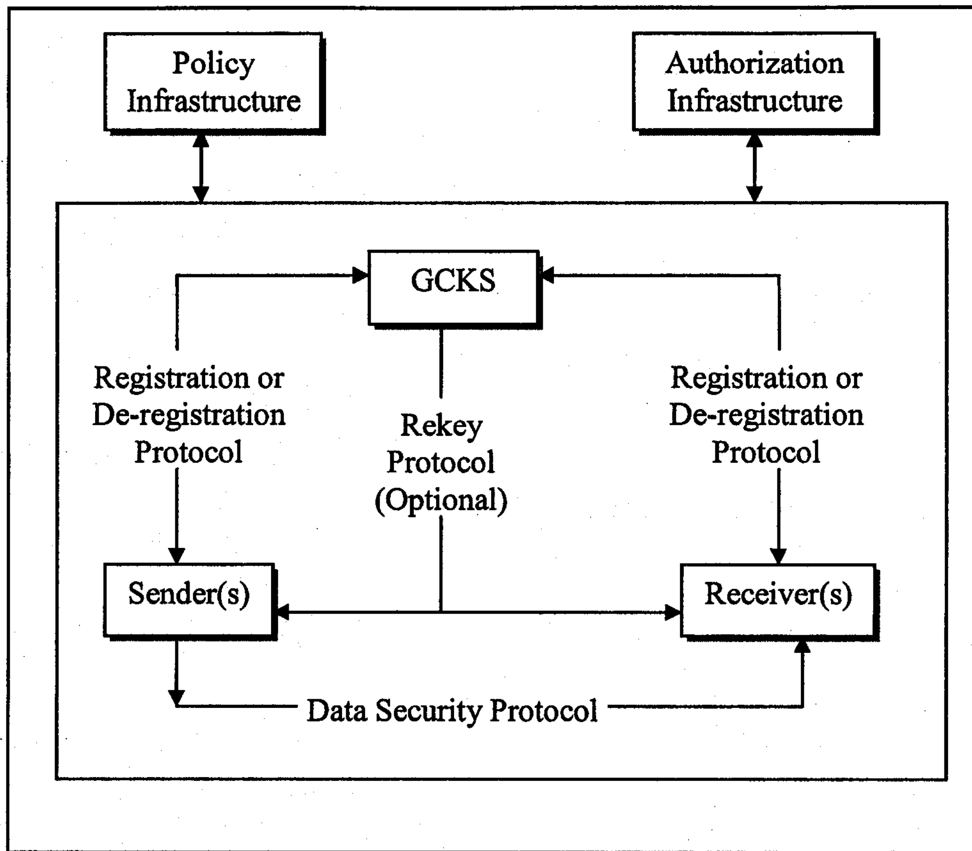


Figure 4.3 Group Key Management Architecture [17]

4.4 Key Updates

To make communication secure, we should have an efficient way to update keys as necessary. In unicast, the key updates involve only two nodes, the sender and receiver. However, in multicast, it is complicated, since there may be thousands of group members.

In multicast, a leaving member should not have access to the later data, that is called perfect forward secrecy. A joining member should not have access to earlier data, that is called perfect backward secrecy. In a symmetric key system, the sender and receiver share the same encryption / decryption key. If we want to ensure perfect forward secrecy

and perfect backward secrecy, we have to update the key for the whole group when a group member joins or leaves. The simplest way to do this is to send a unicast update to each member. If the changes happen in a very large group, it would cost a lot of bandwidth and time to re-key the whole group in this way.

To build a scalable secure multicast system, we should reduce the amount of key update traffic. We can divide a distribution tree into sub-trees, and only update the sub-trees that have changes. Proxy Encryption allows us to update the key for only a part of the distribution tree. Mukherjee and Atwood [4] discuss various other approaches to key traffic reduction.

4.5 The Proxy Encryptions

Proxy cryptography [18] was introduced by Blaze and Strauss in 1998. It works for asymmetric key encryptions. Proxy encryption can transform ciphertext corresponding to one key into ciphertext for another key without revealing any information about secret decryption keys or the clear text [23]. This makes it possible that a non-trusted intermediary router or host can transform an encrypted message into another form without the knowledge of the original message or the secure keys.

The operations of asymmetric proxy encryption can be described as follows:

Two keys, *e* and *d*, are created by a key generation algorithm for encryption and decryption respectively, so that message $m = D(E(m,e),d)$, where $E(m,e)$ is the encryption function of message m using encryption key e and $D(c,d)$ is the decryption function of ciphertext c using decryption key d. The sender generates message m and

sends out the encrypted message $c = E(m,e)$. The receiver gets ciphertext c and decrypts it to the original message $m = D(c,d)$. When it is necessary to perform proxy transformation in an intermediary router, the proxy encryption key generation algorithm produces two proxy decryption keys d_1 and d_2, so that $m = D'(T(c,d_1),d_2)$, where $T(c,d_1)$ is the transformation function of ciphertext c using proxy decryption key d_1 and $D'(c',d_2)$ is the proxy decryption function of transformed ciphertext c' using proxy decryption key d_2. Then, the intermediary router transforms the ciphertext c to c' using the $c' = T(c,d_1)$ function, and the receiver decrypts the transformed ciphertext c' to the original message m using the $m = D'(c',d_2)$ function [18].

To apply proxy encryption in multicast, the sender can encrypt and send data using $E(m,e)$ function all the time, no matter whether there is a router that performs transformation between the sender and the receivers or not. Initially, all receivers decrypt data using the decryption function $D(c,d)$ with key d. When it is necessary to perform transformation in a intermediary router, the decryption key d is split into two parts, key d_1 and d_2. The intermediary router performs transformation function $T(c,d_1)$ with the transformation key d_1 on data to the group. Receivers in the sub-tree rooted at the router perform the proxy decryption function $D'(c',d_2)$ with key d_2. Other receivers can still use key d to decrypt data. The key update happens only in the specified sub-tree, and the sender and other receivers are not affected. When it is necessary to re-key the whole group, all transformations are stopped, and all group members receive new keys and work as in the beginning.

Several encryption schemes can be used for proxy encryption, including El Gamal, RSA, and IBE [23], and formal notations and proofs were discussed in 2003 [23]. In this thesis, the proxy RSA algorithm was used for data protection.

4.6 The Scalable Infrastructure for Multicast Key Management (SIM-KM)

Taking advantage of the proxy encryption and the subgroup technology, the Scalable Infrastructure for Multicast Key Management (SIM-KM) provides an efficient key management for large scale multicast communication [19]. It was presented in 2003, and it allows key updates to be distributed only in a sub-tree, and makes re-keying less expensive. SIM-KM defines three types of operational entities, including Group Manager, Control Group Controller (CGC) and Data Group Controller (DGC). To securely distribute and update keys, detailed operations are presented for different scenarios in [19].

The Group Manager is configured with group and access control information. The access control information includes a list of identities of potential group members, a list of CGCs that have the capability to allow members to join, and a list of excluded Group Controllers that will not receive messages [19]. A Group Manager can be dynamically elected from among the CGCs in case of a fault. The CGC is a trusted group controller, and it stores the Group Manager's address, the access control list, the lists of CGCs and the key information [19]. It is responsible for membership authentication and authorization. The CGC can apply the proxy encryption function on specified multicast traffic on request. The DGC is an un-trusted entity in the network, and it does not have

access to keys to any clear text message. It can transform an encrypted message from one form to another using proxy encryption algorithm with a proxy encryption key received from a CGC [19].

The CGC and the DGC perform the proxy transformation function on a given multicast traffic flow, when it is necessary to change the key in a sub-tree. The transformation on a group is dynamically enabled, based on the changes to the distribution tree. In the path from a sender to a receiver, the transformation must take place only once for a given group [19]. A CGC or a DGC in a higher position in the tree has higher priority to perform transformation, because it can cover more receivers in the sub-tree rooted at it.

Detailed operation steps for each possible scenario are discussed in Mukherjee and Atwood [19]. These scenarios include group create, member join, member leave, data transmission, re-keying, batching, and shutdown.

4.7 Key and Security Association

RFC3740 defines the Re-key SA and the Data Security SA, and requires that they are not negotiated and but are shared within a group [5]. RFC4046, following the guidelines in RFC3740, further discusses the GSAs and key management protocols. It requires that the GCKS creates keys and downloads them to each member in the entire group and all members in the group share the same set of keys [17]. Therefore, the encryption keys, which are essential elements in SAs, are the same in the group and the encryption algorithms are symmetric.

However, in our system, we use the proxy encryption, which is a variety of the asymmetric key encryption. The senders, routers and receivers use different keys, so the GSAs in senders, routers, and receivers are different.

The usage of proxy encryption makes it possible that an intermediary router can be involved in the data protection, and the keys for senders and receivers can be managed separately. With the power of modern processors, it is possible to perform an asymmetric encryption algorithm, such as the proxy encryption algorithm in this thesis, to protect data on-the-fly.

In order to work with asymmetric key encryptions, the RFC3740 and the RFC4046 need to be modified. In the RFC3740, the limitation that all group members must share the same set of Re-key SA and Data Security SA should be eliminated. The SAs could be different not only in a sender and a receiver, but also among receivers in different subgroups. The Re-key SA and Data Security SA could be either multicast or unicast. In RFC4046, the SAs in all the group members should be allowed to be different. The key distribution mechanism should allow different sets of keys to be distributed. The key messages could be unicast or multicast. The Rekey protocol could distribute different keys for different sets of participants.

The multicast system in this thesis, which will be discussed in chapter 6, is based on an asymmetric encryption and subgroup key management. The keys are different not only between a sender and a receiver, but also among receivers in different subgroups. The Rekey messages are unicast or multicast, and are different for different subgroups.

Chapter 5

Building Blocks Used in the Implementation

The implementation in this thesis took advantage of some existing platforms and technologies. The eXtensible Open Router Platform (XORP) [24] as a software router implementation was used in the system, and it will be introduced in section 5.1. Since the data security subsystem is implemented in the Linux kernel using Netfilter/Iptables [25] and RSA [6] encryption algorithm, these technologies will be introduced in the following sections. In section 5.5, the problem that can be solved using these building blocks is discussed.

5.1 XORP

5.1.1 Overview of XORP

The eXtensible Open Router Platform (XORP) is an open source software router platform, running on common hardware [24]. It is designed to be a stable research tool and an extensible deployment platform. The latest version, Release 1.2, can be built on FreeBSD-6.1, OpenBSD-2.8, Linux with kernel 2.6.X, MacOS X 10.4 and Windows Server 2003 [7].

XORP consists of two subsystems. The lower-level subsystem manages the forwarding paths and provides APIs for the higher-level subsystem to access. The higher-

level subsystem includes unicast and multicast routing protocol processes, routing information base and management processes. The lower-level subsystem decides how a packet is routed according to the routing tables imported from the higher level; the higher level communicates with other routers and generates the routing tables [7].

The lower-level subsystem runs inside an operating system kernel. The native forwarding engine provided by FreeBSD or Linux is used for packet processing by default. In addition, the Click modular router [31], a software architecture for building flexible and configurable routers with high packet forwarding rate, can be used for this propose. APIs provided by XORP abstract the low level interfaces and provide minimal dependencies on the underlying forwarding system [7].

The higher-level subsystem, running in user space, uses a multi-process architecture with one process per routing protocol and additional support processes. These processes can run on different hosts in a distributed fashion, and they exchange information using a novel inter-process communication mechanism, called XORP Resource Locaters (XRLs) [26]. XRLs are similar to URLs in concept and transparent to the underlying transport protocols.

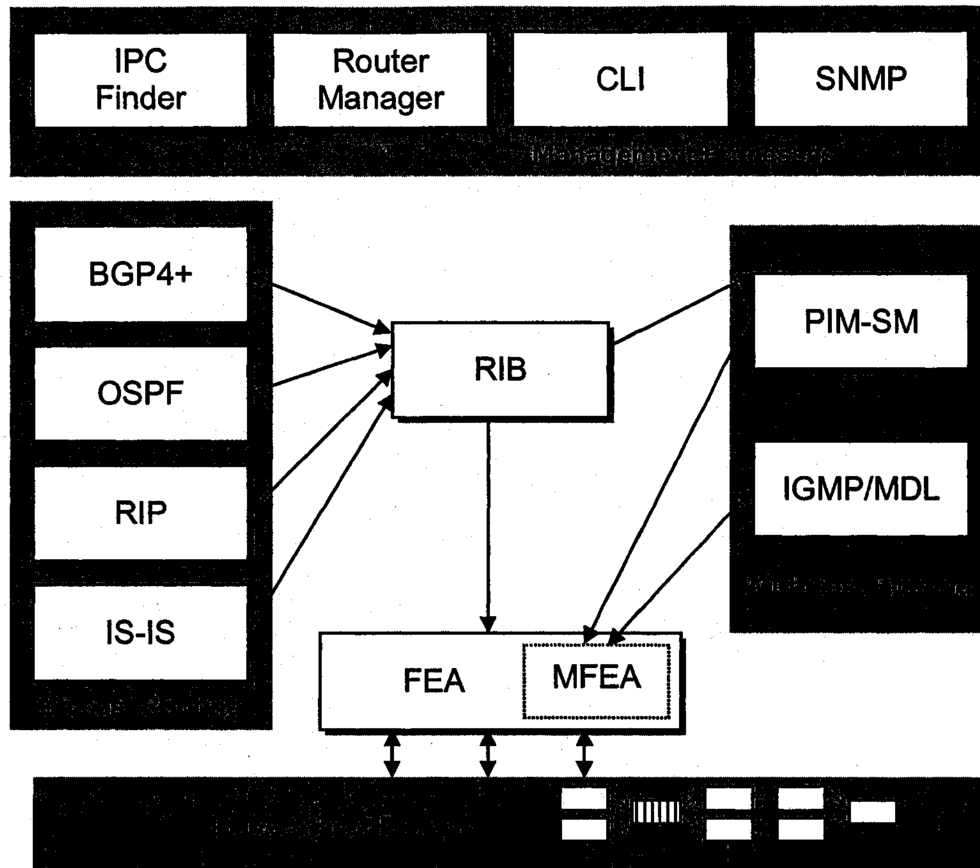


Figure 5.1 XORP Process Model [24]

Figure 5.1 shows the processes in XORP [7]. For simplicity, only the main communication flows used for routing information are shown as arrows, and the control flows between processes are not shown. There are four core processes that are particularly worthy to mention. They are the IPC finder [27], the Router Manager [28], the Routing Information Base (RIB) [29], and the Forwarding Engine Abstraction (FEA) [30].

The IPC Finder, as an IPC redirector, is needed by the communication method used among all XORP components [24]. It contains mappings between abstracted application requests and these components. Each of the XORP components registers with the IPC

Finder so that the finder can assist XRL communications between these XORP components. If a component wants to communicate with another component, it does not need to know explicitly the location of that process, or how to communicate with it. The calling component just sends an XRL request to the finder, and the finder forwards the request to the proper component [27].

The Router Manager is a process that manages the router as a whole. It maintains the configuration information, starts/stops other processes, such as routing protocols and SNMP process, monitors and restarts any failing processes. It also provides the interface for the CLI module to manage the router configuration [28].

The Routing Information Base (RIB) process holds routing information provided by routing processes running in XORP, and arbitrates which routes should be used to forward packets [24]. The winning unicast routes are propagated to the Forwarding Engine Abstraction (FEA) process and hence on to the underlying forwarding engine. Multicast routing information is not propagated to the FEA, since it is only used for providing topology information to multicast routing protocols [29].

The Forwarding Engine Abstraction (FEA) process provides a platform independent interface to the basic routing and network interface management functionality [7]. It abstracts the details of underlying forwarding path implementation, so the forwarding engine can be native FreeBSD or Linux kernel, Click modular router [31], or other approaches. The FEA performs four separate roles: interface management, forwarding table management, raw packet I/O, and TCP/UDP socket I/O. The FEA contains a logical part, Multicast Forwarding Engine Abstraction (MFEA), which abstracts the underlying

multicast forwarding engine and provides a consistent interface to multicast related modules such as PIM and MLD/IGMP [30].

Other modules, such as BGP4+, OSPF, RIP, PIM-SM and MLD/IGMP, implement user-level routing protocols. They interact with the RIB to exchange routing information using XRL. The CLI provides a consistent command line interface to users; the SNMP process enables a XORP router to be managed through SNMP [7].

5.1.2 Interaction with IGMP and PIM-SM

XORP implements three multicast-relevant protocols, MLD, IGMP and PIM-SM. The MLD and IGMP are implemented in one module called MLD/IGMP, and PIM-SM is running in another module called PIM-SM. These three protocols provide XRL interfaces to other XORP modules.

Normally, a XORP module provides two types of XRL interfaces to other modules. The first type, called a protocol interface, provides functions that can be called by other modules; the second type, called a protocol client interface, enables other modules to be informed when events that they are interested in happen [32]. XORP provides an XRL interface specification and tools to assist with generation of C++ code from interface specifications [26].

The MLD/IGMP module provides these two types of interface. Other modules can call the functions specified in the MLD/IGMP interface specification, which can be found in "*xrl/interfaces/mld6igmp.xif*". These functions include Start/Stop or Configure

MLD/IGMP process, or Send/Receive MLD/IGMP control packets to/from the routing module [33].

The MLD/IGMP client interface, specified in "*xrl/interfaces/mld6igmp_client.xif*", enables modules to receive the changes to local multicast membership. Other modules register in the MLD/IGMP process using the XRL interface, and implement the client interface functions that are called when corresponding events happen.

Practically, the register and de-register functions specified in the protocol interface specification are *add_protocol4()*, *add_protocol6()*, *delete_protocol4()*, and *delete_protocol6()*. The callback functions that are specified in the protocol client interface specification and should be implemented by client modules are *add_membership4()*, *add_membership6()*, *delete_membership4()*, and *delete_membership6()*. The functions with suffix 4 are used for IPv4, and the functions with suffix 6 are used for IPv6.

The PIM-SM module only provides the first type of interface, the protocol interface. It includes the methods to Start/Stop/Configure PIM-SM, to Send/Receive PIM control packets to/from the routing unit, or to get protocol-related statistics [34]. It does not provide an interface to allow other modules to register with it, and there is no protocol client interface.

5.1.3 Extension Needed to XORP

To obtain the changes to distribution tree as they occur, PIM-SM protocol interface should be extended and a protocol client interface should be added. In this section the extensions to XORP protocol interface and client interface are discussed.

5.1.3.1 PIM-SM Interface Extension

The extended XRL protocol interface should allow other modules to register interest with PIM-SM. The interested modules will receive Join/Prune information for the particular virtual interface (vif). "*add_protocol4*" is used to register interest for IPv4 protocols; "*add_protocol6*" is used to register interest for IPv6 protocols; "*delete_protocol4*" is used to deregister interest for IPv4 protocols; "*delete_protocol6*" is used to deregister interest for IPv6 protocols.

The added XRL Interfaces Specification segment is shown as follows:

```
interface pim/0.1 {
            :
            :
    add_protocol4 ? xrl_sender_name:txt                              \
                  & protocol_name:txt & protocol_id:u32              \
                  & vif_name:txt & vif_index:u32
    add_protocol6 ? xrl_sender_name:txt                              \
                  & protocol_name:txt & protocol_id:u32              \
                  & vif_name:txt & vif_index:u32
    delete_protocol4 ? xrl_sender_name:txt                           \
                  & protocol_name:txt & protocol_id:u32              \
                  & vif_name:txt & vif_index:u32
```

```
delete_protocol6 ? xrl_sender_name:txt                                \
                   & protocol_name:txt & protocol_id:u32              \
                   & vif_name:txt & vif_index:u32
}
```

The arguments are described as follows:

xrl_sender_name: a text string that indicates the XRL name of the originator of this XRL.

protocol_name: a text string that holds the name of the protocol to add/delete.

protocol_id: an unsigned 32-bit integer that represents the ID of the protocol to add/delete (both sides must agree on the particular values).

vif_name: a text string that holds the name of the vif that the protocol is added to or deleted from.

vif_index: an unsigned 32-bit integer that represents the index of the vif that the protocol is added to or deleted from.

5.1.3.2 PIM-SM Client Interface Extension

The XRL protocol client interface is used for clients to receive distribution tree change information. "*add_subtree4*" is used to inform clients of receiving an IPv4 Join message from a downstream router; "*add_subtree6*" is used to inform clients of receiving an IPv6 Join message from a downstream router; "*delete_subtree4*" is used to inform clients of receiving an IPv4 Prune message from a downstream router; "*delete_subtree6*" is used to inform clients of receiving an IPv6 Prune message from a downstream router; "*leave_tree4*" is used to inform clients of sending an IPv4 Prune message to an upstream router; "*leave_tree6*" is used to inform clients of sending an IPv6 Prune message to an upstream router.

The added XRL protocol client interface specification segment is shown as follows:

```
interface pim/0.1 {
                :
                :
    add_subtree4  ? xrl_sender_name:txt                          \
                  & vif_name:txt & vif_index:u32                 \
                  & source:ipv4 & group:ipv4
    add_subtree6  ? xrl_sender_name:txt                          \
                  & vif_name:txt & vif_index:u32                 \
                  & source:ipv6 & group:ipv6
    delete_subtree4  ? xrl_sender_name:txt                       \
                     & vif_name:txt & vif_index:u32              \
                     & source:ipv4 & group:ipv4
    delete_subtree6  ? xrl_sender_name:txt                       \
                     & vif_name:txt & vif_index:u32              \
                     & source:ipv6 & group:ipv6
    leave_tree4  ? xrl_sender_name:txt                           \
                 & vif_name:txt & vif_index:u32                  \
                 & source:ipv4 & group:ipv4
    leave_tree6  ? xrl_sender_name:txt                           \
                 & vif_name:txt & vif_index:u32                  \
                 & source:ipv6 & group:ipv6
}
```

The arguments are described as follows:

xrl_sender_name: a text string that indicates the XRL name of the originator of this XRL.

vif_name: a text string that indicates the name of the vif where the event happened.

vif_index: an unsigned 32-bit integer that represents the index of the vif.

source: an address structure that holds the source address that has been joined/pruned.

group: an address structure that stores the group address that has been joined/pruned.

5.2 The Netfilter / Iptables

Netfilter is a packet filtering framework inside the Linux 2.4.x and 2.6.x kernels [25]. As the successor of the previous Linux 2.2.x ipchains and Linux 2.0.x ipfwadm, it provides similar APIs to allow packet filtering, network address translation, and packet mangling.

Netfilter provides a series of hooks to allow kernel modules to register callback functions with the network stack in the Linux kernel [26]. A hook is a set of rules that specify how to process each passing packet. Every packet that meets a rule will be forwarded to the specified callback function. In the Netfilter framework, there are five hooks, Pre-routing, Forward, Post-routing, Input, and Output. Figure 5.2 shows the positions of hooks in the Netfilter framework. The boxes are the hooks; the diamonds show the route decision; the arrows indicate the flows of packets [27].

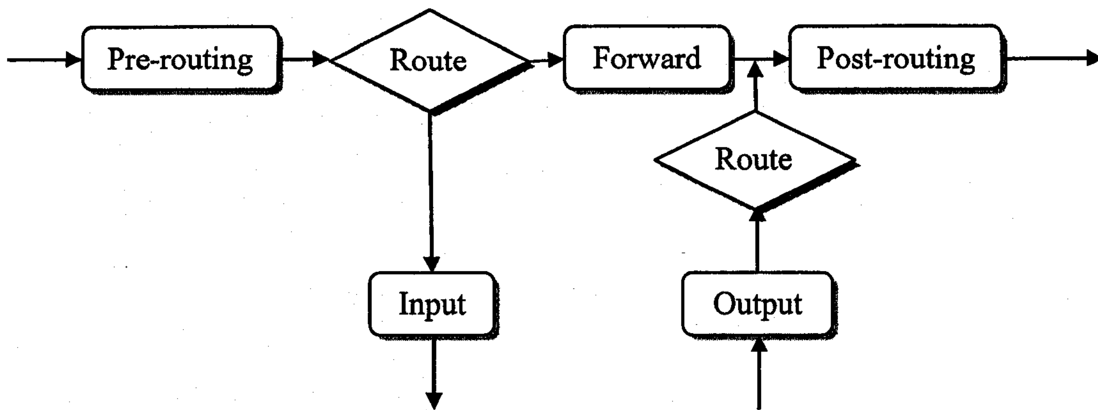


Figure 5.2 Netfilter/Iptables Hooks [27]

The pre-routing hook processes incoming packets before routing code. Prior to that, only a few simple consistency checks are performed, including checks with regard to the version, length, and checksum fields in the IP header. All incoming packets addressed to the local host pass the Input hook. All incoming packets that are not addressed to the local host pass the Forward hook. All outgoing packets generated by the local host pass the Output hook. Post-routing hook can be used to manipulate all outgoing packets before they are sent to the lower layer [25].

5.3 The Linux Kernel Module and Data Structures

Linux provides an interface to allow a user to add a piece of code into the system kernel so that this code can be executed faster, avoiding context switches. This piece of loadable code is called a Linux kernel module [35]. A kernel module works in the operating system kernel, so there are some restrictions. The only functions that a kernel module can call are the ones provided by the kernel, and there are no standard libraries to use. Compared to a user-space program, a kernel module has limited stack area to use. It can be as small as a single 4096-byte page, and it has to be shared with other kernel

functions. Coding a kernel module should be done very carefully, because a fault in the module may crash the whole operating system [35].

An important structure, which is at the core of the network subsystem of the Linux kernel, was used in this thesis. That is the socket buffer, "*sk_buff*" [36]. A socket buffer is the buffer that holds network packets in the Linux kernel. A packet received by a network card is stored in a socket buffer and then passed to the network stack. The Linux kernel uses the socket buffer to represent and manage a packet all the time [36]. A socket buffer consists of two parts, packet data and management data. The packet part stores the data actually transmitted over a network including the MAC header, IP header, and IP payload. The management data contains additional data that are not stored in an actual packet but are necessary for processing in the Linux kernel. These data include pointers, timers, and information exchanged between protocol instances [36].

The Linux kernel provides operations on socket buffers; only the operations relevant to the implementation in this thesis are introduced in this section [36].

"*skb_copy_expand()*" creates a new and independent copy of the socket buffer and packet data; the user can indicate the space reserved before and behind the packet data by passing "*newheadroom*" and "*newtailroom*" respectively.

"*kfree_skb()*" frees the specified socket buffer and related data structures.

"*skb_pu()t*" appends data to the end of the current data range of a packet and update the corresponding data structures. It is useful in our case, because in the implementation

UDP packets should be encapsulated into ESP packets, and trailers of ESP packets are needed to be generated at the end of these packets.

5.4 RSA Algorithm and Proxy RSA

The RSA algorithm [6], named after its inventors Ron Rivest, Adi Shamir and Len Adleman, is the most widely used asymmetric cryptosystem. It can be used to provide both security and digital signatures, and its security is based on the difficulty of factoring large numbers [6]. The RSA algorithm is so powerful that it has become the de facto standard for industrial-strength encryption, especially in the Internet.

The RSA algorithm is used in a Public key cryptosystem, and it uses two different keys: Public Key and Private Key. The Public Key is used to encrypt plain text; the Private Key is used to decrypt cipher text and reveal the original message. Public and Private keys are generated in pairs so that only a specific pair of keys can perform the encryption and decryption functions. Any keys other than the specific pair will not work [6].

Public Key and Private Key can be generated by the following steps [6]:

1. Generate two large random primes p and q, each roughly the same size.
2. Compute $n = pq$ and $\varphi = (p\text{-}1)(q\text{-}1)$.
3. Select a random integer e, $1< e < \varphi$, such that the greatest common divisor of e and φ is equal to 1.
4. Compute the secret exponent d, $1< d < \varphi$, such that $ed = 1(\text{mod } \varphi)$.

5. The Public Key is (*n, e*) and the Private Key is (*n, d*). The values of *p, q,* and φ should also be kept secret.

To send a confidential message, the sender represents the plain text message as a positive integer m, and computes the cipher text $c = m^e \bmod n$ using the Public Key (*n, e*), then sends the cipher text *c* out. To reveal the original message, the receiver uses the Private Key (*n, d*) to compute $m = c^d \bmod n$ [6] [37].

It is worth mentioning that the message *m* should be smaller than the modulus *n*, $0 \le m < n$ [6], otherwise the algorithm will fail.

To apply RSA in proxy encryption, we need to split the decryption key *d* into two parts d_1 and d_2 such that $d = d_1 d_2 (\bmod \varphi)$ [23]. The intermediary entity gets the proxy transformation key (d_1, n) and the receiver uses the new decryption key (d_2, n). They perform the identical decryption algorithm to the original one to process received messages, and the receiver can reveal clear messages as normal [23].

5.5 Scalable and Secure Multicast System

With the existing technologies and platforms outlined in the previous sections, we can build a scalable and secure multicast system. The system has a hierarchical topology structure and follows most of the guidelines in the MSEC Working Group. Encryption keys are managed by SIM-KM, which follows most of RFC 4046. The data propagated in the network are protected according to IPsec. The routers in the network run XORP, which enables IGMP and PIM-SM. Hosts report their interest by IGMP, and routers distribute multicast data based on multicast routing tables calculated by PIM-SM. The

multicast security module works in the Linux kernel, protecting multicast data using proxy RSA encryption on request.

Chapter 6

The Design and Implementation

With the technologies discussed in previous chapters, a multicast system can be built to demonstrate the works in the HSPL. The high-level design of the Scalable Secure Multicast System in section 6.1 provides an overview of the system. In section 6.2, the design of the Data Security Module that is implemented in this thesis is described. The implementation issues of Group Security Association (GSA) and the data structures used in the implementation are discussed in section 6.3 and 6.4. In section 6.5, detailed implementation information of the Multicast Data Security module is provided. In section 6.6, implementation issues related to key size are discussed. In the last section of this chapter, 6.7, additional key management and testing tools are presented.

6.1 The High-level Design

The Scalable Secure Multicast System is designed to secure traffic for large multicast groups. The network topology of the system is based on the hierarchical topology framework [3] discussed in Chapter 5. The Scalable Infrastructure for Multicast Key Management [19] approach is used for key management. Multicast data are protected by the proxy encryption scheme. This system follows most of the guidelines of the Multicast Group Security Architecture [5] and the Multicast Security Group Key Management Architecture [17].

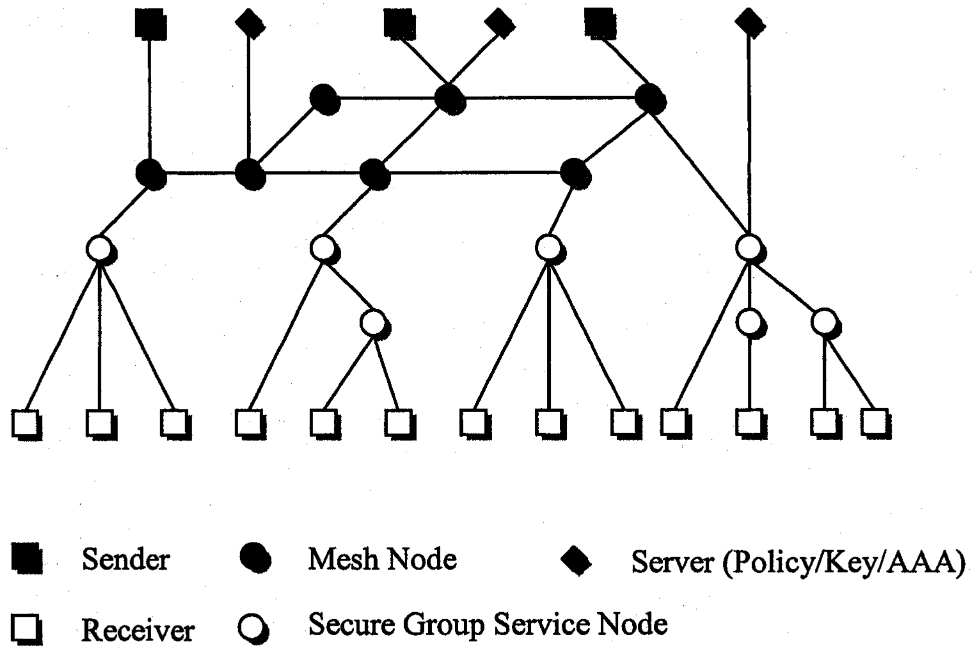


Figure 6.1 Network Topology

The topology of the Scalable Secure Multicast System is shown in Figure 6.1. Some Senders are directly connected to Mesh Nodes, which are pre-deployed in the network. The Mesh Nodes can be the root nodes of the multicast distribution trees. They are native multicast routers with additional software, and are the core of the network. Several Group Server Nodes are distributed in the system. They can be multicast enabled routers with additional functionality or hosts for the sub-networks that lack multicast capacity routers. A lot of receivers, as leaf nodes, are directly connected to the Group Server Nodes. Servers can be connected to either Mesh Nodes or Service Nodes. They can be Policy Servers [5], AAA Servers or Key Servers.

A sender in the system is responsible for generating multicast traffic securely. Figure 6.2 shows the overall design of a sender.

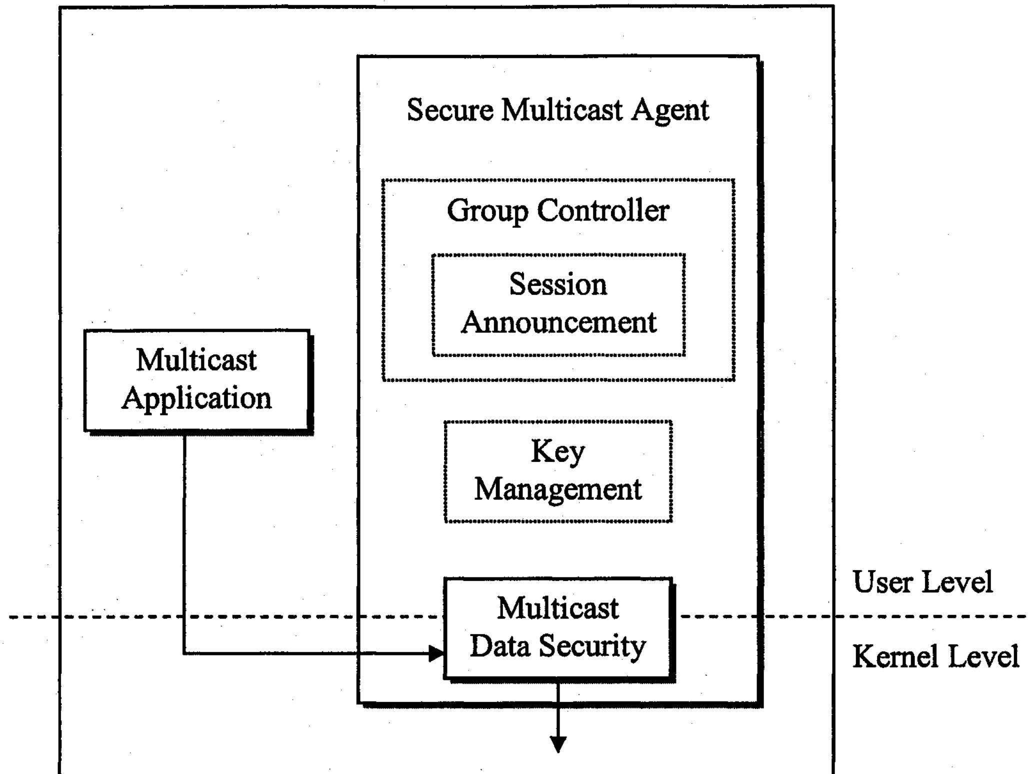


Figure 6.2 Sender

The boxes are the functional modules, and the arrows show multicast data flows. A sender is an un-trusted entity in the system, so it should get authorized and authenticated before actually joining the group. The Group Controller module performs such functionality. Besides, the module performs Session Announcement as the function required for a sender. The Key Management module handles GSAs and receives re-keying messages. The Multicast Data Security module encrypts data for given groups using security information provided by the Key Management module.

A receiver, as a client, is an un-trusted entity in the system. Its main goal is to receive data correctly. Figure 6.3 shows the overall design of a receiver.

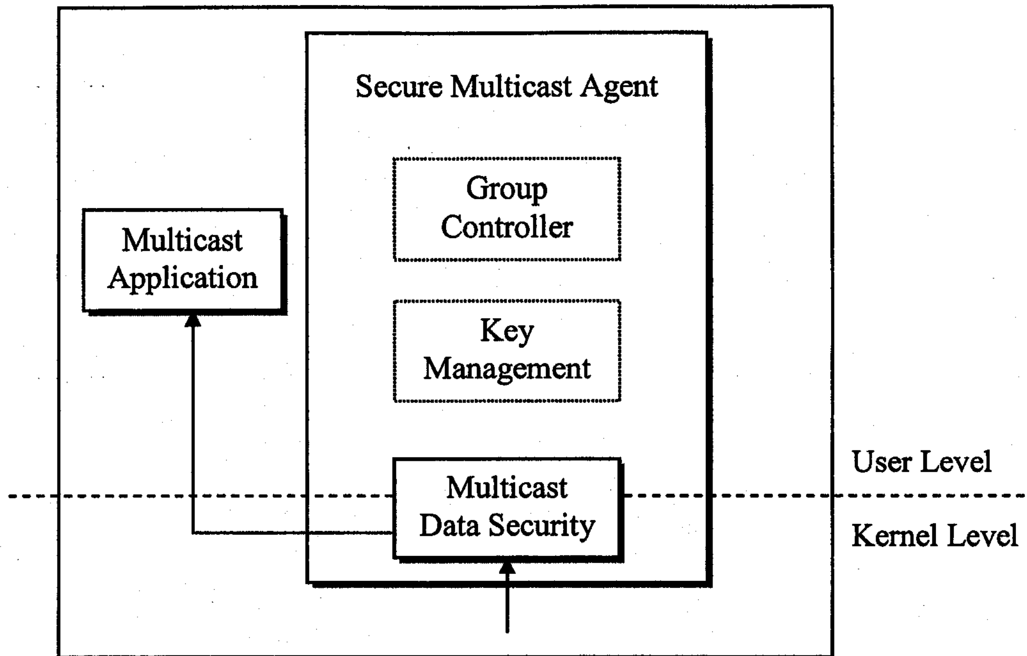


Figure 6.3 Receiver

The Group Controller module interacts with a Service Node (SN) router to verify its End User Identifier and Host Identifier [38] [39], join/leave a group, and obtain information for the Key Management module. The Key Management keeps communicating with the SN router regarding key changes and managing security information after the group is established. The data decryption function is performed in the Multicast Data Security module based on the information installed by the Key Management.

A Mesh Node router, as a core router, should provide high-speed transport, so it should perform as few functions as possible, besides routing and forwarding. In our system, the main functions added to a MN router are to perform the tree auto-configuration algorithm and group management.

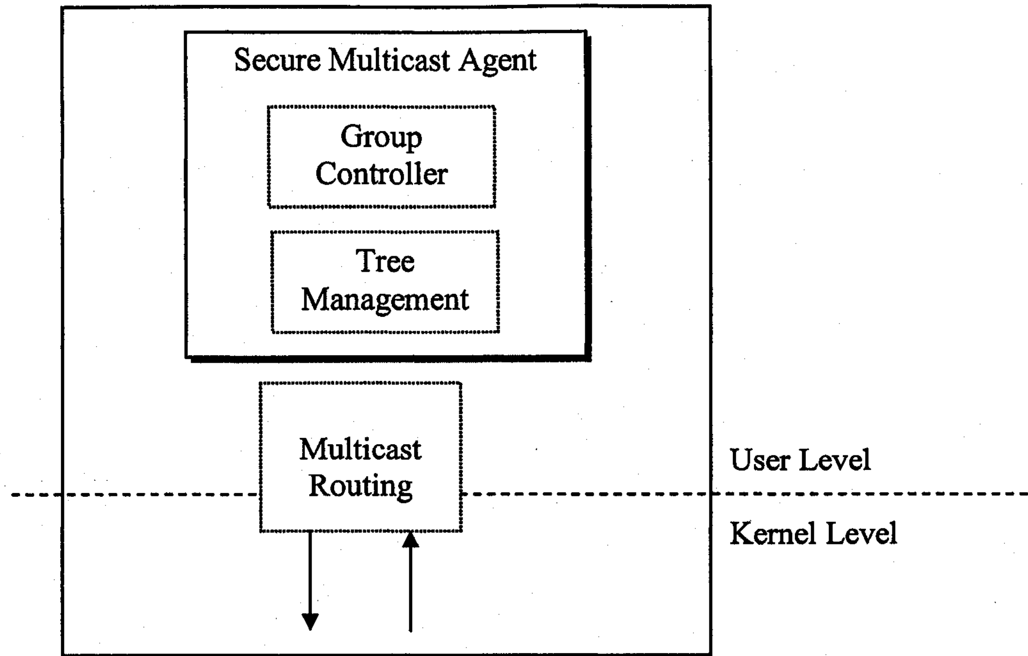


Figure 6.4 Router (Mesh Node)

As shown in Figure 6.4, it does not manipulate passing multicast data and just forwards them based on multicast routing table, so these functions can be user-space processes in XORP. The Group Controller is used to authenticate and authorize senders and to process session announcements. The Tree Management module performs the Tree Auto-configure algorithm to construct a Mesh, and builds the distribution tree rooted at the nearest MN to a sender. Details of these operations can be found in [3].

A Service Node router performs most of the add-value functions in our system. It is a node in the distribution tree, forwards multicast data and transforms these data on request. The high-level design blocks are shown in Figure 6.5. The boxes are functional modules, and the arrows indicate the multicast data flows.

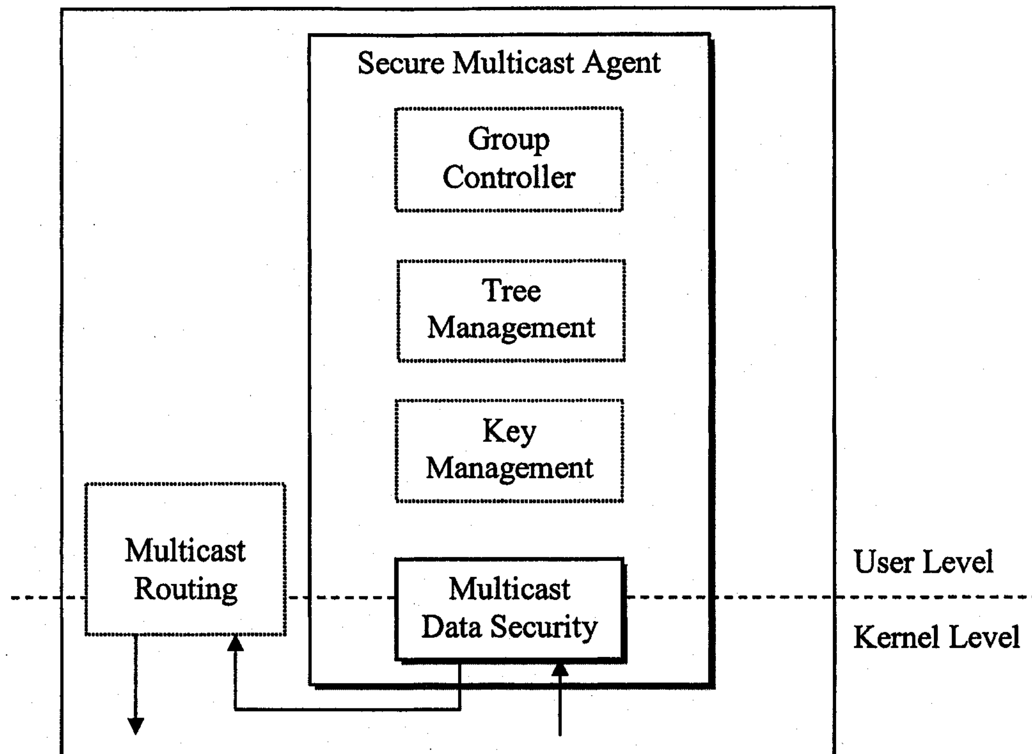


Figure 6.5 Router (Service Node)

The Service Node router may be trusted or un-trusted. In a trusted router, the Group Controller module communicates with Policy Servers and AAA Servers to obtain the group policies and group access control information. It processes receivers' and other SN routers' Join/Leave requests, and provides group change information to the Tree Management module. The Tree Management module takes part in building the distribution tree for a given group. The Key Management module interacts with Key Servers to get updated keying material. The Multicast Data Security module transforms passing multicast data on request.

In an un-trusted router, the Group Controller module merely forwards these requests to a trusted router and acts according to its decision. The other modules perform the same functions as in a trusted router.

The Service Node routers next to the Mesh Node routers are the boundary routers that perform proxy transformation function in distribution trees, because the Mesh Node routers do not have such functionality. If the transformation is required to be performed in a higher level in a tree, the key server should update the key for the whole group.

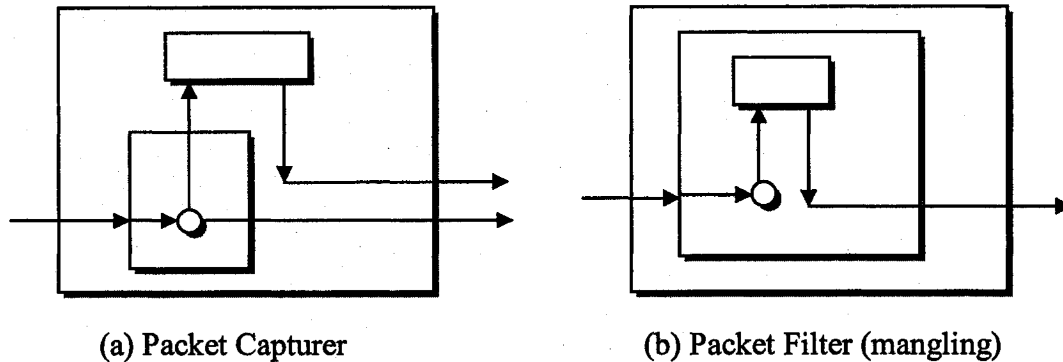
6.2 The Design of the Multicast Data Security Module

The Multicast Data Security Module is an essential part of the whole system. It falls in the Multicast Data Handling functional area defined in RFC3740. This module covers the security-related treatment of multicast data by the senders, routers, and receivers for large groups.

The module interacts with other modules in the system and performs data security transformation on given multicast traffic flows. This transformation performs processor intense encryption operations and applies on every matched packet. If the module works in user-space, it will involve two context switching times for each packet, copying a packet from kernel to user-space and copying the transformed packet back from user-space to system kernel, so a user-space module to process every matched packet would have unacceptable performance. Taking the performance issue into account, the main functionality of the module works in the operating system kernel.

There are a number of low-level packet manipulation technologies in Linux, such as libpcap, tcpdump, Ipchains, and Iptables. All of them provide interfaces to allow a user to capture specified packets and apply some operations on them. However, the packets captured by a user are copies of the original packets, and these original packets will be

processed as normal in kernel, as shown in Figure 6.6 (a). If there is no mechanism to prevent the original packets from being forwarded, the end user will receive more than one copy of these packets. This is undesirable.



(a) Packet Capturer (b) Packet Filter (mangling)

Figure 6.6 Packet manipulation technologies

To overcome it, Netfilter/Iptables, using packet filter technology, is applied in our system. It provides functionality that allows users to manipulate interesting packets in the Linux kernel. As shown in Figure 6.6 (b), the matched packets are forwarded to a user defined module, and this module decides how to process them: accept, deny, or alter. Using this scheme, only one copy of the data can be received by a receiver efficiently, so the Multicast Data Security Module uses Iptables as the low-level packet manipulation mechanism.

As described in Chapter 5, Iptables has 5 hooks where users can register interest. Registering in a certain hook allows users to examine only the specified type of flows. Since the hosts that run the module may be at different positions in a distribution tree, such as sender, router, or receiver, we should find a proper scheme to register their interest.

At a sender's side, the traffic that needs to be protected should be encrypted right after the packets are generated, so the security module registers its interest in the OUTPUT hook. At the receiver's side, the module should be only interested in the traffic addressed to the local host, therefore the INPUT hook is used to match these packets. As the result, the Iptables has fewer traffic flows to check; the performance can be further improved consequently.

In a router, the process becomes more complicated. Normally, a multicast router receives one copy of traffic flow, and it may send out multiple copies of this flow through different output ports based on the multicast routing tables. In our system, if a router is involved in protecting a multicast traffic flow, all the participants in the sub-tree rooted at this router should get the same data. That means that all the outputs are identical. Therefore the module in a router should transform multicast data before the packets are copied, otherwise multiple output streams should be transformed independently with the same operations. In Netfilter/Iptables, there are two hooks that meet this requirement. They are PRE-ROUTING and FORWARD. Considering the fact that the router could act as a receiver to monitor the traffic at the same time, the PRE-ROUTING hook is used in a router. The hooks used for the Multicast Data Security module are shown in Figure 6.7.

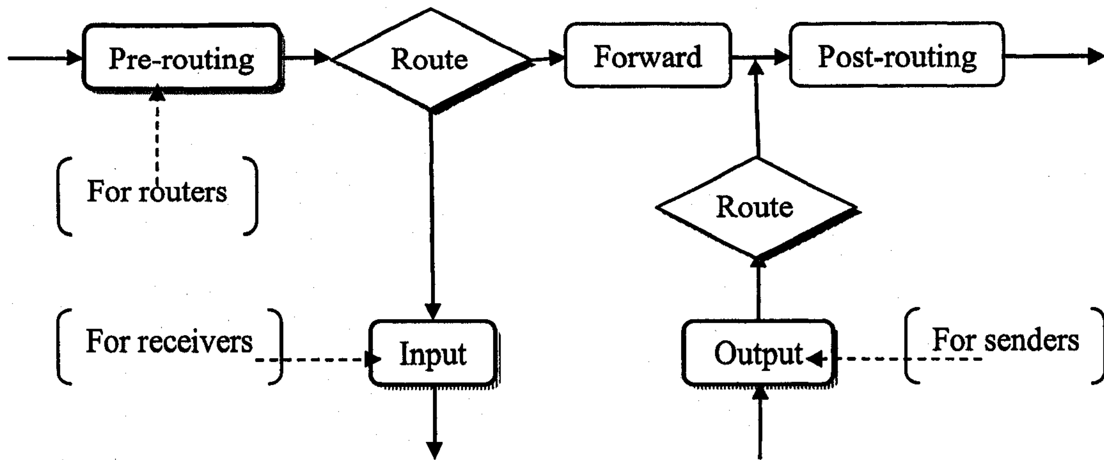


Figure 6.7 Module's registration points

From Figure 6.7, the traffic flow can be observed in a host. If the host has only one role to perform, the flow is clear. If a router also needs to receive the same traffic besides transforming it, the host first transforms the traffic received from the PRE-ROUTING hook. After route decision, the host receives traffic from the INPUT hook and decrypts these packets, then sends it to the upper-layer application. The module is not designed for a host that routes the packets sending from itself, that is, the host acts as a sender and a router, because it is not feasible in practice. In the case that a host acts as a sender and a receiver, multicast packets from the upper-layer application are matched at the OUTPUT hook and then sent to the module to encrypt. After encryption and route decision, packets are sent to the INPUT hook (this is not shown in the figure), because the host, as a receiver, joined the group that it is sending to. Then these packets are forwarded from the INPUT hook to the module, and decrypted there. Finally, packets are sent to the upper layer from the INPUT hook.

6.3 Implementation Issues Concerning GSAs

A Group Security Association (GSA), which contains the needed information for securing group communication, is a set of Security Associations (SAs), including Registration SA, Re-key SA, and Data SA [5]. The first two categories of SAs ensure that the group keys and other needed information are installed and distributed securely. Therefore, they are designed to be implemented outside the Multicast Data Security Module. In this module, we need only to deal with the Data SAs.

The Data SAs in our module includes the following attributes: group addresses, protocol numbers, destination port numbers, Security Parameter Indexes (SPIs), algorithms, key lengths, and keys. These attributes are divided into two parts. One part is used for matching packets to a given session, and another part is used for protecting data to the session. In the current implementation in this thesis, the algorithm and the key length are fixed. Only the RSA algorithm encryption is implemented and the length of the keys is 32 bits. Taking the advantage of the packet matching mechanism in Iptables, this implementation uses the first part of the information, including group address, protocol number, and port number or SPI, to match needed packets, and sends matched packets to the module along with needed security information.

At the sender's side, the Data Security SAs for a given group include group IP addresses, protocol numbers, destination port numbers, encryption algorithms, key lengths, and keys. They are installed into Iptables' OUTPUT hook as a rule using the Iptables' user-space interface. The group IP address, protocol number, and port number are used to match packets, and the key is used to encrypt matched packets.

In a router, the Data Security SAs contain group IP address, protocol number, SPI, algorithm, key length, and keys. Since the multicast packets transmitted in the network are encapsulated into ESP packets, the router has no idea about the original port number. The SPI should be assigned by a Group Controller/Key Server, and it is unique for a given group, so the combination of group address and SPI can distinguish the multicast traffic flow that should be transformed.

The situation at the receiver's side is similar to that in a router. The incoming packets are encapsulated using ESP, and the group address and SPI are used to match these packets. However, the difference is that output packets of a receiver are UDP/TCP packets rather than ESP packets for a router.

6.4 Data Structures Used in the Implementation

There are three important structures in the implementation in this thesis. One defines the information shared between user-space and kernel-space Iptables modules. The other two define the ESP packet formats.

The structure "*ipt_MPE_info*" defines the information needed by the Iptables kernel module. It includes *algorithm*, *key* (*e/d, n*), *key_len*, *SPI*, and *seq*. Most of the information is the part of the Data SAs, including the algorithm, key, key length and SPI. The "*seq*" field that holds the sequence number for a given multicast flow required by ESP is used only in the kernel module. When a user or Key Management module wants to specify a traffic flow to be protected, an Iptables rule is installed using user-space interface with parameters. Some of these parameters, such as address, port number, and SPI, are used

by Iptables to match packets, and other parameters, such as SPI, and key, are stored in an instance of this structure. This instance is associated with the rule. Every packet that satisfies the rule is sent to the module with a pointer to this instance as a parameter. Therefore, the security information can be shared between user-space and kernel-space. The kernel module can receive a matched multicast packet with proper transformation information in this structure.

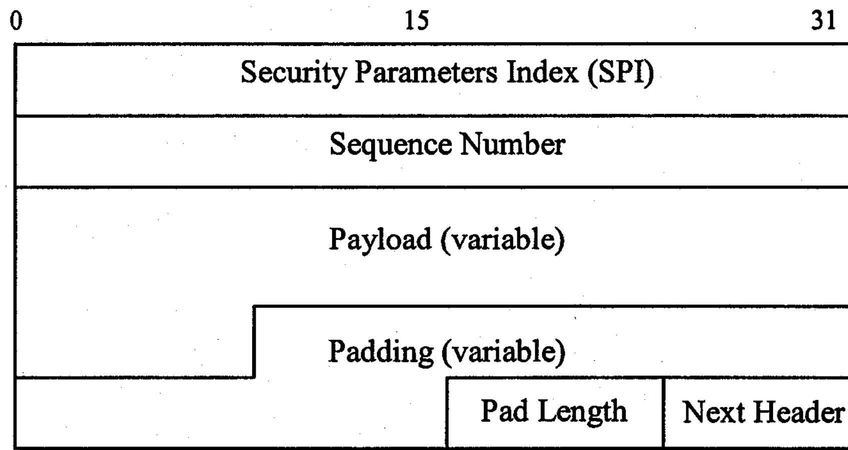


Figure 6.8 ESP Packet Format [22]

ESP packet format [22], as shown in figure 6.8, is defined by two structures. One is ESP header, and another is ESP trailer. The format follows the specification in RFC4303. The header includes a SPI field and a Sequence Number field. The trailer structure contains Pad Length and Next Header.

6.5 The Implementation of the Multicast Data Security Module

The Multicast Data Security module in the Scalable Secure Multicast System is implemented as an Iptables module. This module includes two parts: user-space shared library and kernel-space extension module. The user-space part receives parameters

required for securing a given group, and installs the corresponding rules into the Iptables framework. Then the kernel module can receive all packets that match the given rules from Iptables. The kernel module can decide how to manipulate these incoming packets based on security parameters and the role of the host where the module is running.

The user-space library is used to control the kernel model. It is called by the Iptables user-space interface. The user-space library starts from a function called "*_init()*", which is automatically called upon loading. In this function, "*register_target()*" is called, with a "*iptables_target*" structure as parameter, to register the kernel module with Iptables.

The *iptables_target* structure is defined by Netfilter/Iptables, and is an important data structure for extending Iptables. Some important fields are described as follows:

.name: the name of the user-space library, for our case, the *.name* is "MPE" (Multicast Proxy Encryption). The name should be the same as the name for the kernel module.

.extra_opts: a pointer to an "*option*" structure, which contains the command line option arguments.

.help: a pointer to a function that prints out help information.

.init: a function pointer that can initialize extra space in the "*iptables_target*" structure.

.print: a pointer to a function that is called by the chain listing code to print the target information. The information includes the key and algorithm.

.final_check: a pointer to a function that is called after the command line has been parsed. This function gives us a chance to make sure that all parameters have been specified. In our case, the validity of the key is checked.

.parse: a pointer to a function that is used to verify if arguments are used correctly and set the values used by kernel module. This function will be called one or more times, depending on the number of arguments.

The Iptables kernel module is the main part of the implementation. It includes an implementation of multicast proxy encryption and part of the IPsec ESP protocol. The module receives all packets that meet the rules specified by the user-space program, and appropriately transforms them.

As an Iptables kernel extension, the module calls the "*ipt_register_target()*" function in the module initialization function, "*init()*", to register itself as a Iptables target. An "*ipt_target*" structure is passed as the parameter of the target registration function. Important fields in this structure are described as follows:

name: A string that holds the module name, this name should be the same as the name specified in the user-space library. In our case, the name is "*MPE*".

checkentry: A pointer to a function that checks the specifications for a rule. In practice, the validity of the key pair is checked, and the table that the module is called from should be the "*mangle*" table.

target: A pointer to a target function, which is the core function of the module. It takes the socket buffer, the hook number, target information, and user information as parameters. The socket buffer stores the packet that should be processed in this module. The target information points to an "*ipt_MPE_info*" structure that contains security information for the packet in the socket buffer. The hook number indicates where the packet comes from.

The Multicast Data Security kernel module has different behaviours depending on the roles in distribution tree (sender, router, or receiver). The role can be determined by the

hook where the module is associated. Detailed implementation for these three roles will be discussed in the following three sections.

6.5.1 The Implementation of the Sender Module

A sender in the Scalable Secure Multicast System generates multicast packets, performs required security transformation, and sends these packets out. The data security module in a sender captures every packet that needs to be protected, encrypts it and encapsulates it into an ESP packet, and then sends it back to the protocol stack. The flowchart of the kernel module is illustrated in Figure 6.9.

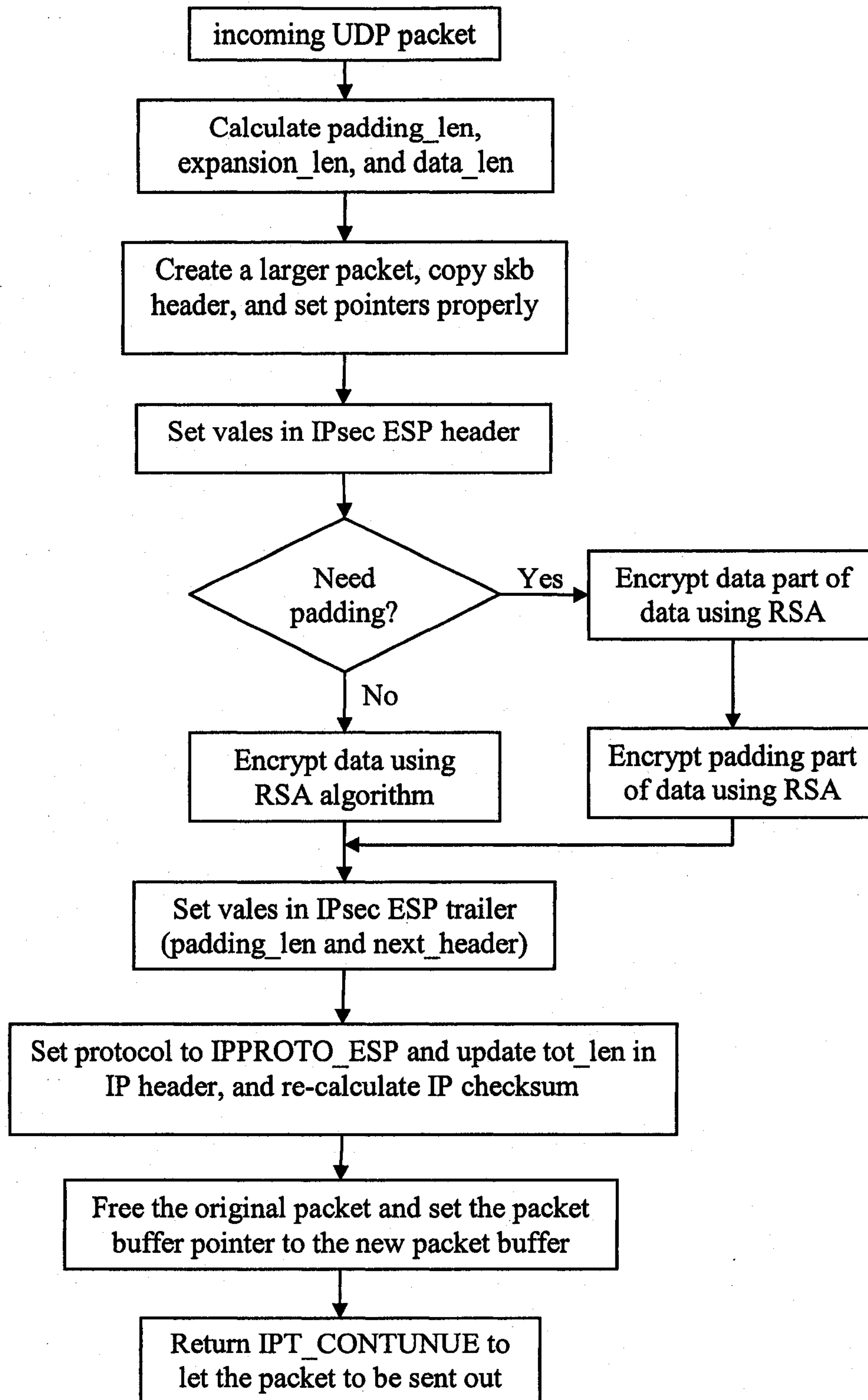


Figure 6.9 Flowchart of the Sender Module

The matched incoming UDP packet is sent from OUTPUT hook by Iptables. Since in this module a UDP packet will be encapsulated into an ESP packet, the sizes of data structures used for transformation should be calculated first. The RSA algorithm is a block cipher, so the transformed packet needs padding to fill the new packet to the size required by the RSA algorithm. The padding length, "*padding_len*", is the size of padding added into the new packet. The expansion length, "*expansion_len*", is the size of the ESP packet, including the lengths of UDP header, ESP header, payload with padding, and ESP trailer. The "*data_len*" indicates the length of payload that can be encrypted without padding. The transformation and these lengths are illustrated in Figure 6.10.

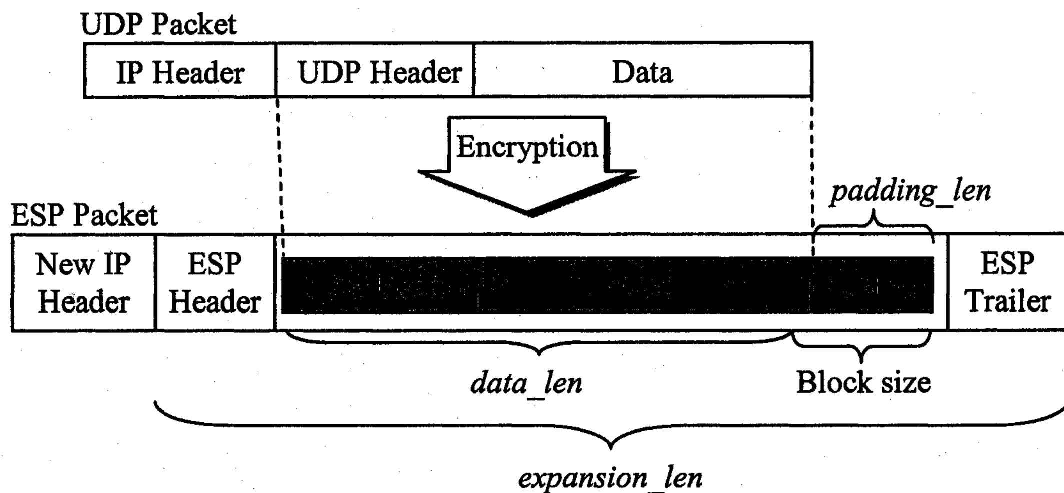


Figure 6.10 From UDP Packet to ESP Packet

After determining the new ESP packet size, "*skb_copy_expand()*" is called to create a new and independent copy of the original socket buffer and packet data. The socket buffer and IP header are copied into the new buffer, so it is unnecessary to create a new IP header. Then, an inline function, "*skb_put()*", is called to append data to the end of the current data range of the new ESP packet. This function increases the pointer "tail" and

reserves the space for later use. Before manipulating the new packet, pointers to new IP header, ESP header, and ESP trailer should be set properly.

Values in ESP packet are set in order. Firstly, the fields in ESP header are set, including SPI and sequence number. Secondly, the UDP packet is encrypted using the RSA algorithm and put into the ESP payload field. If the packet needs padding, the data is separated in two parts to be encrypted. Finally, the padding length and next header in ESP trailer are set.

After filling the ESP packet, the IP header needs to be updated. The protocol field is set to "*IPPROTO_ESP*". The total length of IP packet, "*tot_len*", is updated by adding IP header length and expansion length. IP checksum is re-calculated based on the new packet. Now, the original socket buffer can be freed by calling "*kfree_skb()*"; then, set the original socket buffer pointer to the new socket buffer so that the kernel can continue processing this packet with the transformed data. The last thing for this module is giving this packet back to kernel by returning "*IPT_CONTINUE*".

6.5.2 The Implementation of the Receiver Module

A receiver in the Scalable Secure Multicast System receives multicast packets from the wire, performs required security transformation, and sends these packets to the upper-layer application. The data security module in a receiver performs almost the reverse process as in a sender. It captures every packet that is protected, de-capsulates and decrypts it from an ESP packet, and sends back to the protocol stack. The flowchart of the receiver kernel module is illustrated in Figure 6.11.

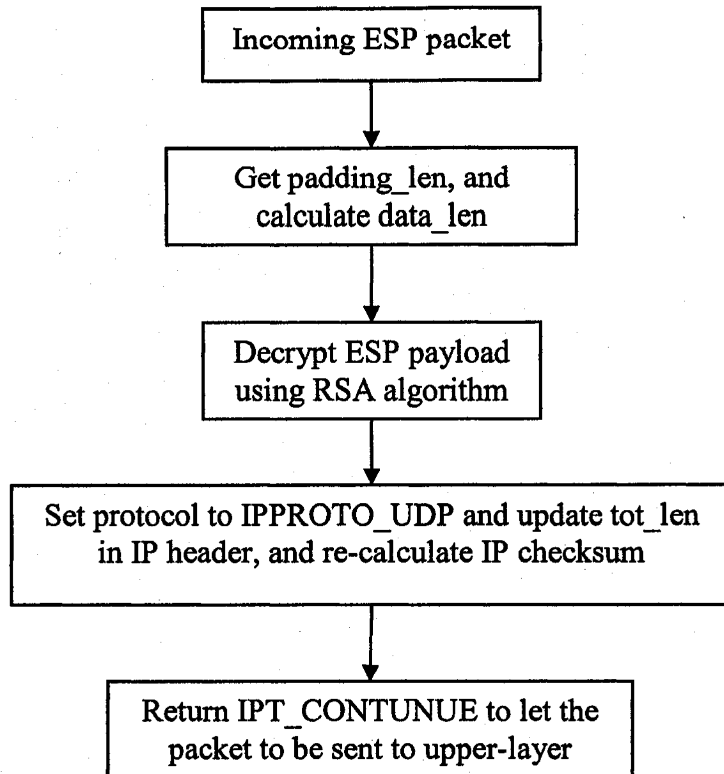


Figure 6.11 Flowchart of the Receiver Module

The incoming packets for a receiver are always ESP packets captured from the INPUT hook. Similar to that in the sender end, before performing decryption, size of data structures should be calculated and pointer should be set properly. The decryption task is relatively easier, because the packet data length satisfied the size required by the RSA algorithm after transformation in the sender end. After decryption and de-capsulation, the IP header should be updated. The protocol field is set to "*IPPROTO_UDP*", and the IP packet length and checksum are re-calculated. Finally, this module sends this packet back to the kernel protocol stack by returning "*IPT_CONTINUE*".

At the receiver's side, a new socket buffer is not created, due to two reasons. First, the output UDP packet is smaller than the input ESP packet, so no extra space is needed. Second, the ESP header is located after the IP header and before the UDP header and

data, so the module can use the ESP header's space and the decrypted data can be written right after the IP header without overwriting the data that have not been processed. Such an approach not only saves space, but also improves the performance.

6.5.3 The Implementation of the Router Module

A router in the Scalable Secure Multicast System performs required security transformation on specified passing multicast data packets. The data security module in a router captures every ESP packet that needs to be transformed, transforms it with a transformation key, and sends it back to the protocol stack. The functionality performed in the router module is relatively simple, because the transformations between UDP and ESP are left to the sender and receiver. The flowchart of the router kernel module is illustrated in Figure 6.12.

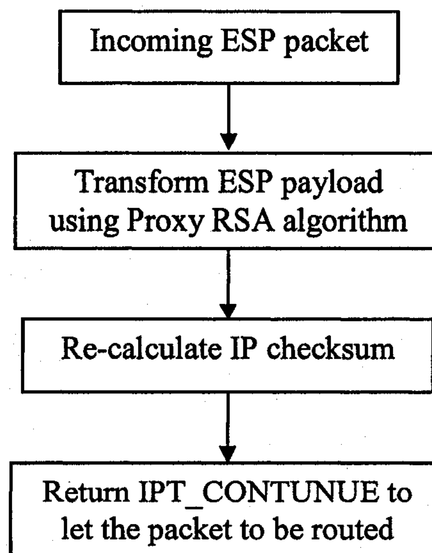


Figure 6.12 Flowchart of the Router Module

Upon receiving an ESP packet from PRE_ROUTING hook, this module sets the pointers properly. Then the payload in the ESP packet is transformed based on the

multicast Data SAs associated with current traffic flow. Similar to the receiver end, the transformation task is relatively easier, because the ESP payload length matches the size required by the RSA algorithm after transformation in the sender end. There is also no new socket buffer needed, because the transformed data block is written to the same offset as the original one, and no overwriting can occur. After transformation, checksum in IP header is re-calculated, and the packet is sent back to the Linux kernel.

6.6 Proxy RSA Algorithm and Implementation

6.6.1 Implementation Issues.

The core function in implementing RSA algorithm is performing modular exponentiation. The modular exponentiation problem can be represented that given base *m*, exponent *e*, and modulus *n* to calculate *c*, such that $c = m^e \bmod n$.

A straightforward approach to compute the *c* is to calculate m^e directly, then to take the result modulo *n*. This method requires O(*e*) multiplications to complete, and needs a very large space to hold the result of m^e. Moreover, the module is implemented in Linux kernel where only plain C can be used, so there is no power function, "*pow()*", available as in standard C library. Therefore this approach cannot be applied in this implementation.

An optimized binary left-to-right approach, which significantly reduces both the number of operations and memory usage, is used in this kernel module [41]. To compute $c = m^e \bmod n$, let *e*, *s* bits long, be represented in base 2 as $e = a_{s-1}a_{s-2}\cdots a_1a_0$,

Then, the problem can be solved in a computer with fewer operations and less memory usage by [41]

$$c = m^e (\bmod n) = \prod_{i=0}^{s-1} \left(m^{2^i}\right)^{a_i} (\bmod n) = \prod_{i=0}^{s-1} \left(\left(m^{2^i}\right)^{a_i} (\bmod n) \right) (\bmod n)$$

The running time complexity of this approach is $O(\log e)$ [41]. In implementation, this method can be further optimized by scanning over the exponent bits from left to right.

6.6.2 Key Size and Limitations

The size of the RSA key determines the strength of the confidence level and the key lifetime. In general, a longer key can provide better protection and has longer lifetime. However, there are several implementation issues that affect the key size.

In general, a longer key requires more generation time, and encryption and decryption with a longer key take more CPU cycles. The router that performs the multicast proxy transformation may process multiple multicast traffic flows at the same time; however, simultaneously, it may perform other tasks, such as unicast and multicast routing, packet forwarding, and network management. Therefore, the transformation should not take too much time, and thus the key should not be too long, even in a platform with hardware cryptographic accelerators. Considering the possibility of low power receivers, such as low-end PCs or Set-Top-boxes, in a multi-media multicast session, the strength of the RSA key should not affect the QoS.

Generating a strong RSA key requires high cost of CPU cycles. In a large Scalable Secure Multicast System, new keys may often be required due to the changes of network topology and group membership. Splitting a large key for proxy encryption also needs a lot of processor power, so the size of key for the Scalable Secure Multicast System should not be too large.

The RSA algorithm is a block cipher, which divides the bit stream into blocks of a given size and the encryption algorithm acts on that block to produce a cryptogram block. If the block size is bigger than the packet size, one block will span two or more packets. In the case that one packet involving a block is lost, all other packets that contain part of this block cannot be decrypted successfully. Practically, due to the factor that the message m should be always smaller than modulus n, the block size of RSA algorithm is slightly bigger than the size of a key. Therefore, the length of the key should smaller than the size of one packet.

Another implementation issue that limits the key size is that the padding length field of an ESP packet is 8 bits [22]. That means that the maximum padding size is 255. This fact limits the length of key to 255 bits, if packet fragmentation is unacceptable.

6.7 Additional Key Management and Multicast Testing Tools

6.7.1 Multicast Key Management Tools

To verify the correctness of the multicast data security module, several multicast key management tools were developed. They are not an implementation of the Key

Management module in the multicast system, but some functions can be used in the future Key Management implementation.

The "*mkeygen*" program randomly generates and outputs encryption and decryption keys for RSA algorithm. This program also splits the generated decryption key, and output proxy decryption key pairs for routers and receivers.

A daemon program, "*mkeyd*", running in the background, receives key update messages sent by "*mnewkey*" or "*mrekey*" and manages Iptables rules, which contain security information for given groups. The "*mnewkey*" issues new key messages to senders and receivers; the "*mrekey*" sends out specified proxy decryption key messages to routers and receivers. The key message is put in a UDP packet, and the packet format is shown in Figure 6.13.

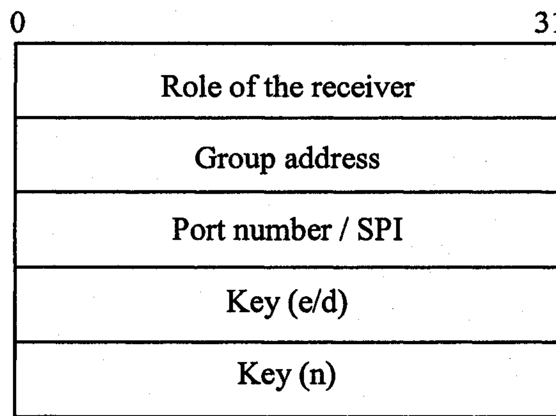


Figure 6.13 Key Message Format

The first field, 32 bits, indicates that the message is for a sender, a router, or a receiver. Because the same daemon is used for receiving key messages in all hosts, the daemon has to know how to install the key information into Iptables from such a field.

The group address field stores the address of group. The next 32 bits is used for destination port number if the host acts as a sender, or is used for SPI if the host is a router or a receiver. The last two fields hold the key pair for the RSA algorithm.

6.7.2 GUI Multicast Sender and Receiver Programs.

A GUI multicast user space program was developed to make testing easy. This program was written in script language, Python [42], so it can be easily modified to meet various requirements during testing. Another advantage of this program is that it is platform independent, so it can run on the Linux or Windows without any change.

The program can work as a sender or a receiver based on the command argument, "*-s*" for starting as a sender and "*-r*" for starting as a receiver. If there is no argument, selection window appears to allow the user to choose. The sender sends out specified size of packet to a multicast group at a defined interval. To avoid overloading the network, the packet generation rate is set to one packet per second by default. Parameters that can be specified from the user interface are group address, port number, TTL, packet size, and packet generation interarrival time. As shown in Figure 6.14, the default values for group address, port number, TTL, packet size, and interarrival time are 224.0.8.115, 2006, 3, 128 and 1 respectively. Information of the sending packet is displayed in the window, including the source address, the destination group addresses, the first 16 bytes of the packet and the packet size.

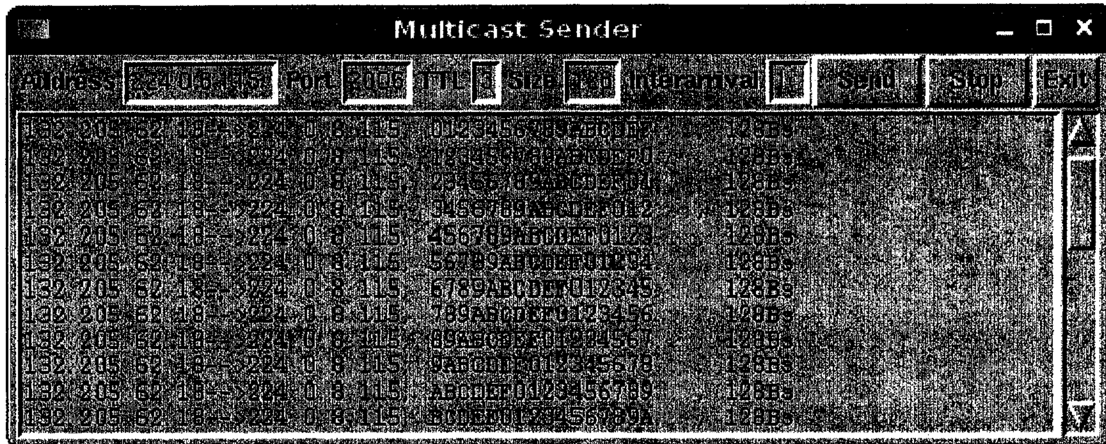


Figure 6.14 Multicast Sender

In case that user wants to generate packets quickly, there will be a lot of output information. To avoid decreasing the performance by printing out too much information, only one "*S*" is printed out for each packet. The sender and receiver agree that the destination port numbers that less are than 2000 are used for this purpose. When a user specifies an interarrival time less than 0.2 second for a group with a port number larger than 2000, the program changes the port number to 1999 automatically.

At the Receiver end, users can specify the group address and port number of the multicast traffic they want to receive. When the receiver receives a packet, it prints the information of the packet.

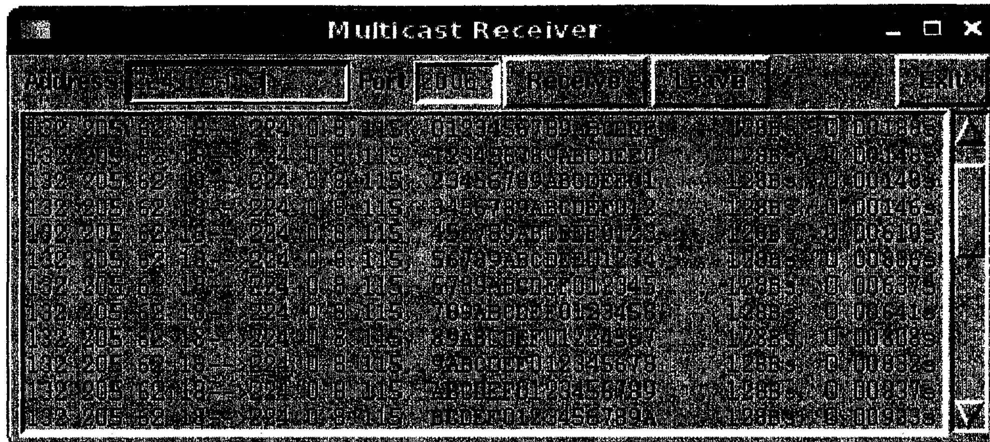


Figure 6.15 Multicast Receiver

As shown in Figure 6.15, the first two fields indicate the source address and the group address. Following that are the first 16 bytes of the payload of the packet, the size of the received packet and the delay of it. When a packet with a destination port number less than 2000 arrives, only one "R" is printed out. A dot in the window indicates that a time-out (2 seconds by default) occurs.

Chapter 7

Testing the Multicast Data Security Module

In section 7.1, the development and testing environment in the HSPL is described. Validation and performance measurement are given in section 7.2.

7.1 Testing Environment

The implementation in this thesis was developed in the HSPL. Three hosts and two switches are used for development and testing. The three hosts, *Alder*, *Oak*, and *Forest*, have identical basic hardware and software configuration: Intel Pentium II 350 MHz CPU, 64Mb Memory, SMC 10/100 Ethernet Network Card, Redhat Linux 7.3 Operating System with Linux-2.4.32 Kernel, and Gcc-2.96 C/C++ compiler.

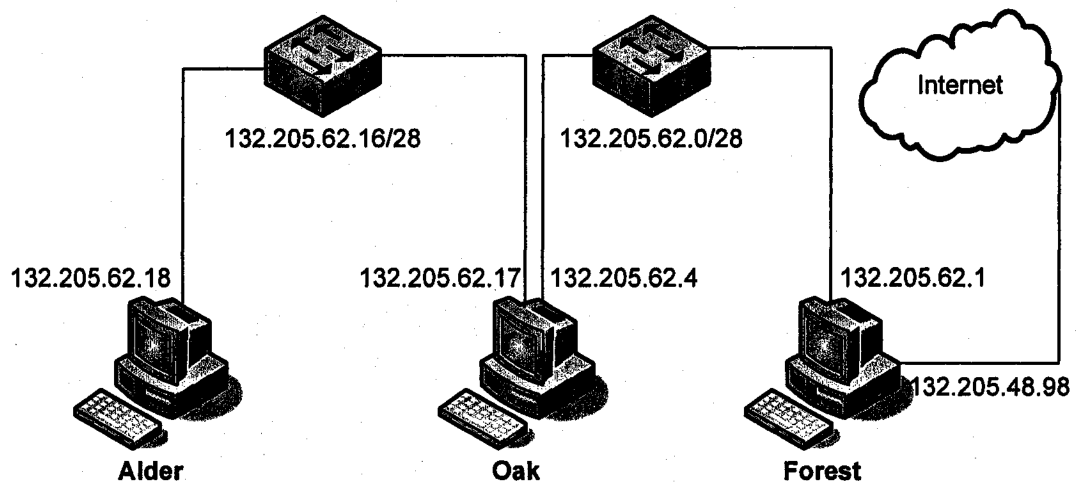


Figure 7.1 Network Configuration

The network configuration is shown in Figure 7.1. *Alder* and *Oak* are connected in the same sub-net 132.205.62.16/28; *Oak* and *Forest* are in another network, 132.205.62.0/28. *Oak* works as a router in between. The development networks have access to the outside through *Forest*. The logical topology of the network is shown in Figure 7.2.

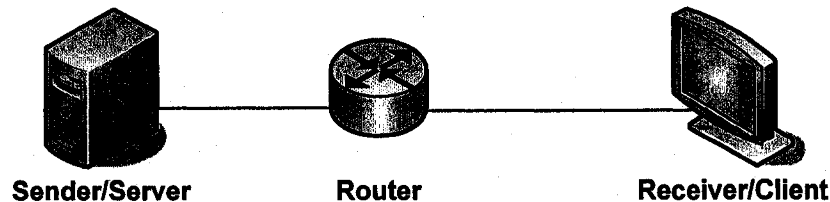


Figure 7.2 Logical Network Topology

7.2 Testing and Results

The testing procedure is described as the follows:

Without any transformation/encryption, the receiver could receive correct multicast data from the sender via the multicast enabled router, which runs XORP. That means that the sender, router, and receiver work well in the traditional way. Using a packet sniffer provided by Linux, Ethereal, to capture the packets in the network, it can be observed that the multicast packets for the given group are UDP packets.

After enabling the security modules and installing proper keys in the sender and receiver, the receiver could receive correct data. The packets captured by Ethereal were ESP packets. When the key was changed in the sender or receiver, the data could not be received. The reason is that the UDP packet, which was sent by the sender's application, including UDP header and payload, is encrypted using RSA algorithm. If it is decrypted with an incorrect key, the port numbers in the UDP header will be changed to an

unexpected value. Moreover the UDP checksum is incorrect, and the packet will be dropped silently. These mean that the RSA algorithm and ESP protocol work in the module. To further validate the implementation, the module in the sender was disabled; then the receiver could get the correct data. The reason is that the sender sent out UDP packets without the security module enabled, and the module in the receiver was configured to manipulate ESP packets, so these UDP packets that arrived at the receiver did not satisfy any rule and were forwarded to upper-layer application. The result obtained from the packet sniffer confirmed that the packets for the given group are UDP packets during that period. When only the module in the receiver was disabled, the Receiver application could not receive any packet for the given group. The reason is that, without the security module, the receiver could not understand the ESP packets that were sent by the sender.

After enabling the security module in the router, installing a proper proxy transformation key in the router, and updating the decryption key for the receiver, the receiver can receive correct data. The packets captured by Ethereal were ESP packets. If the key was changed in the sender, router or receiver, the receiver could not receive any multicast data from the given group. The reason is the same as the previous case. If the module in the router was disabled, the receiver could not receive any multicast data from the given group. The reason is similar to using an incorrect decryption key in the receiver. These mean that the Proxy Encryption works in the security module.

The implementation demonstrated that the Proxy Encryption works with multicast and it is transparent to traditional multicast applications.

To test the performance of the implementation, packet delay is recorded. Before testing was started, the sender's and receiver's clock were synchronized using Network Time Protocol (NTP), which can achieve an accuracy of microsecond level. The packet generation interarrival time was set to 0.5 second. The relation between packet size and delay is shown in Figure 7.3.

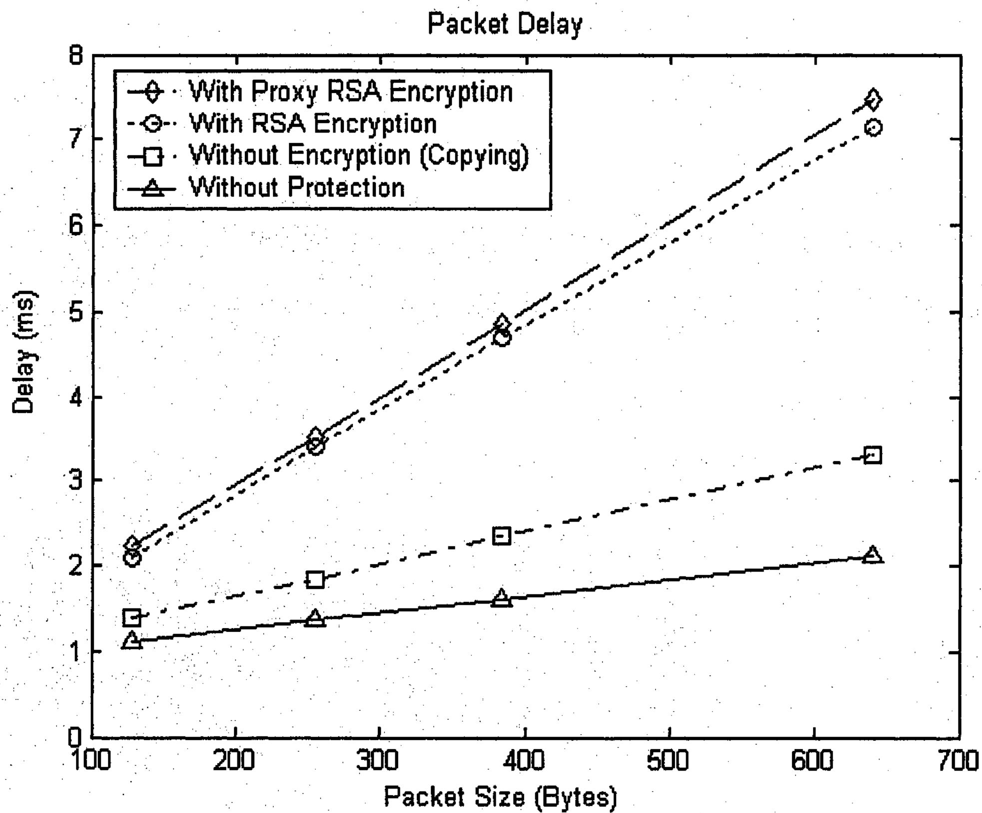


Figure 7.3 Packet Delay

The solid line shows the relation in the scenario without any data protection. The dash-dot line represents the scenario without RSA encryption. It shows how much delay can be introduced by encapsulation and de-capsulation. The dotted line represents the scenario where the sender and receiver performed RSA algorithm on data. The dashed line shows the scenario where the sender, router and receiver joined the transformation.

The delay increases with the growth of packet size in all scenarios, because bigger packets take more time to be propagated and processed. In the last two scenarios, encryption and encapsulation introduced additional delay. The increments of delay in these two scenarios also increase with packet size. In practice, the delay differences from scenario 1 and scenario 3 are proportional to the packet size: 1.0 ms (from 1.1 ms to 2.1 ms) for 128-byte packets, 2.02 ms (from 1.38 ms to 3.40 ms) for 256-byte packets, and 5.03 ms (from 2.12 ms to 7.15 ms) for 640-byte packets. The increments of delay in the last two scenarios are very close, around 0.2 ms difference. That means that the participation of the intermediary router does not affect the performance too much. The proxy security module achieves a satisfactory performance.

Chapter 8

Scalable Key Management

In practice, asymmetric key encryptions, such as RSA, are not used to protect data packets; however, symmetric algorithms, such as DES, are applied for this purpose, because they are several orders of magnitude faster than RSA in software implementations [43]. Keys for asymmetric algorithms must be longer for equivalent resistance to attack than for symmetric algorithms. As claimed by RSA Security, 1020-bit RSA keys are equivalent in strength to 96-bit symmetric keys [44]. It is too expensive to apply asymmetric key encryptions to protect multicast data on-the-fly.

Since proxy encryption can transform ciphertext for one key into ciphertext for another key, we can apply proxy encryptions for key distribution, so that only valid group members in a sub-tree can reveal new decryption keys. The algorithms used to protect multicast data can be symmetric ones. The following steps describe the procedure.

1. Initially, all group members share the same encryption/decryption key. The Data SAs, in senders and receivers, are the same, and they use the same symmetric algorithm and key *d*. All receivers and Service Nodes share the same Re-key SA. In this Re-key SA, there are two keys. One asymmetric key *r* is used to protect Re-key messages, and it works for proxy encryption

algorithm. Another key u is used to protect the update for the previous key. It can be an asymmetric key or a symmetric key.

2. When group membership changes, a Key Management server sends a unicast Key Update message to a Service Node at a minimum sub-tree that can cover this change. The message contains key r_1 split from r, and is protected by u. The Service Node updates key r to r_1. Then, the Key server sends a unicast Key Update message protected by key u to each valid receiver in that sub-tree. This key message holds the proxy decryption key r_2 split from r. Therefore, the Service Node holds key r_1, receivers in the sub-tree know key r_2, and the remaining receivers have key r.

3. Then the key server sends a multicast Re-key message that should be decrypted by the key r to all group members. All members, except members in that sub-tree, receive the message and update the encryption/decryption key d. Upon receiving the Re-key message, the Service Node transforms the message for key r to message for key r_2, and distributes the message to the sub-tree. All receivers in the sub-tree, including invalid ones, can receive the Re-key message, but only the valid receivers know key r_2, and can reveal the updated key in this message.

All group members, including senders, valid receivers and invalid receivers, can receive the multicast Re-key message. However, only the members who have a valid Re-key SA can reveal the new key.

This key distribution schema can guarantee the perfect forward secrecy and perfect backward secrecy with a light traffic. It isolates group changes, which are introduced by joining and leaving, in a sub-tree, and apply twice key updates. The first update is the unicast key update that happens only in the changed sub-tree, and can be received only by valid receivers. It is used to update the key to the second update. The second update is sent in multicast manner, and all members can receive it. Due to the transformation in a Service Node, the second update received by these receivers in the sub-tree should be decrypted by the key in the first update message. Therefore, in a large group, key messages could be fewer than sending unicast key messages to all valid group members.

Take an example to illustrate the efficiency. In a traditional scenario, DES algorithm is used to protect multicast data and Re-key messages, and the key length is 96-bit. The size of a Re-key message is about 300 bits, including packet headers and control information. The number of group members is m. When a change occurs, a key server would send $200m$ bits.

In a scenario with our key distribution scheme, the DES algorithm is used to protect multicast data and the first key update messages, and the key length is 96 bits. The message size is about 350 bits. Re-key messages, the second key updates, are protected by the proxy RSA algorithm. To achieve an equivalent key strength, 1020-bit RSA key is used. The Re-key message size is about 1300-bit. The group has m members, and the average sub-tree size is g. When a change occurs, joining or leaving, the key server sends $g+1$ RSA key updates to the sub-tree members, including the Service Node and valid receivers ($g + 2$ for joining and g for leaving, and we assume that they have an equal

chance to happen). After that, it sends one multicast DES key update to all members. Therefore, the message that the key server should send is

$$1300(g + 1) + 300 = (1300g + 1600) \text{ bits.}$$

When apply our key distribution scheme in a large group with 2,000 group members, and the average sub-tree size is 50, the amount of data would be send for each key update is 1300 x 50 + 1600 = 66,600 bits. In a traditional case, the amount of data should be sent is 200 x 2000 = 400,000 bits. Therefore, 400,000 – 66,600 = 333,400 bits, about 83% of traffic, could be saved. The amount of key update traffic in our scheme depends on the size of sub-tree, and does not depend on the size of the whole group, so our scheme can be applied in large-scale multicast groups.

If apply proxy encryption only in key distribution, the system conform to the Data SA requirements in and RFC 3740 and RFC 4046. All group members, including senders and receivers, hold the same Data SA and share the same encryption / decryption key. However, the Re-key SAs should be different in a group, and key messages could be unicast and multicast.

The group address for multicast Re-key messages could be the same as the multicast data, and use a different destination port number. Our implemented Multicast Data Security module can be used to capture these Re-key packets, and perform proxy transformation on them.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

Multicast is an efficient way to distribute data to multiple receivers simultaneously. The security and scalability issues prevent multicast from being applied widely in the Internet. In this thesis, we presented a Scalable Secure Multicast System to explore such an area.

The design explored a scalable and secure multicast system, which integrated two major achievements in our research group, the Hierarchical Topology for Multicasting and the Scalable Infrastructure for Multicast Key Management. The system guarantees the confidence of the multicast data, and provides the capability to be expanded for a large-scale application.

The investigation in XORP allows us to integrate our future development into a real router platform. The XRL interfaces provided by XORP allow new modules to communicate with the existing components efficiently. The extension proposed in this thesis enable other modules to interact with PIM-SM.

The implementation demonstrated that the proxy encryption for multicast works well. The security module works in the Linux kernel, and uses Iptables' packet matching mechanism. The kernel module has a user-space interface, which allows the module to be

configured dynamically. The module can achieve a satisfactory performance in a common platform. It proofed the concept of proxy encryption with multicast in software. In hardware implementation, technologies, such as Ternary Content Addressable Memory (TCAM) [45] and cryptographic accelerators can be applied to achieve high packet processing rates. Additional programs and tools help validation and testing, and can be applied in future projects.

The new key management approach, applying proxy encryption in key distribution, isolates key updates in a sub-tree, and improves the scalability of multicast key management. It can guarantee perfect forward secrecy and perfect backward secrecy with very light traffic.

The Scalable Secure Multicast System made great efforts toward providing a scalable and secure multicast system for the real world. The implementation of our modules proved the feasibility of proxy encryption with multicast.

9.2 Future Work

The efforts in this thesis are just the start of providing a secure multicast system in the real world. The design of the system and the proposed key distribution scheme needs to be further refined. In this thesis, only the Multicast Data Security module was implemented. Other modules, Group Controller, Key Management, and Tree Management, should be implemented and integrated into a whole system.

The testing and demonstration were done only in HSPL. To demonstrate and test in a large network, the Tree management module should be implemented, so that the

distribution tree can automatically be built; the Key Management module should also be implemented, so that keys can automatically be generated, split, and distributed to group members upon the changes to networks. However, it is difficult to fully test the system in an actual big network, because it is impractical to have thousands of group members and hundreds of routers run Linux and the modules that we developed.

The modules were designed assuming that proxy encryption would be used for protecting multicast data. If proxy encryption is only used for protecting Re-keying messages, and then the sender and receiver modules need to be able to perform symmetric encryption and decryption on the data in addition to asymmetric encryption and decryption. The routers only need to do proxy transformation on Re-keying messages. It is not much work to add this generalization.

The current Multicast Data Security module implementation is based on the Linux kernel 2.4.x. The Click Modular Router can be used as part of the Data Security module implementation, so that the system can be built on most of versions of Linux and FreeBSD.

Bibliography

[1] S. Deering, "*Host Extensions for IP Multicasting,*" RFC 1112, Internet Engineering Task Force, August 1989.

[2] C. Diot, B. N. Levine, B. Lyles, H. Kassem and D. Balensiefen, "*Deployment Issues for the IP Multicast Service and Architecture,*" IEEE Network, January/February 2000.

[3] T. Wang and J. W. Atwood, "*Hierarchical Topology for Multicasting,*" In Proceedings of IASTED Conference on Computer Systems and Applications (CSA 2004), Banff, Canada, 2004.

[4] R. Mukherjee and J. W. Atwood, "*Proxy Encryptions for Secure Multicast Key Management,*" In Proceedings of the 28th IEEE Conference on Local Computer Network (LCN 2003), Bonn / Koenigswinter, Germany, October 2003.

[5] T. Hardjono and B. Weis, "*The Multicast Group Security Architecture,*" RFC 3740, Internet Engineering Task Force, March 2004.

[6] R. L. Rivest, A. Shamir, and L. M. Adleman. "*A Method for Obtaining Digital Signatures and Public-key Cryptosystems,*" Communications of the ACM, pp. 120-126, 1978.

[7] "*XORP Design Overview,*" XORP technical document, http://www.xorp.org/.

[8] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma and L. Wei, "*Protocol Independent Multicast-Sparse Mode*

(PIM-SM): Protocol Specification," RFC 2362, Internet Engineering Task Force, June 1998

[9] C. Diot, W. Dabbous, and J. Crowcroft, "*Multipoint Communication: A Survey of Protocols, Functions, and Mechanisms,*" IEEE Journal in Selected Areas in Communications, vol. 15, no. 3, April 1997.

[10] S. Casner and S. Deering, "*First IETF Internet Audiocast,*" ACM Comp. Commun. Rev., pp 92-97, July 1992.

[11] K. C. Almeroth, "*The Evolution of Multicast: From the MBone to Interdomain Multicast to Internet2 Deployment,*" IEEE Network, January/February 2000.

[12] S. Paul, "*Multicasting: Empowering the Next-Generation Internet,*" IEEE Network, January/February 2000.

[13] W. Fenner, "*Internet Group Management Protocol, Version 2,*" RFC 2236, Internet Engineering Task Force, November 1997

[14] B. Cain, S. Deering, I. Kouvelas, B. Fenner and A. Thyagarajan, "*Internet Group Management Protocol, Version 3,*" RFC 3376, Internet Engineering Task Force, October 2002

[15] D. M. Chiu, S. J. Koh, M. Kadansky, B. Whetten and G. Taskale, "*Tree Auto-Configuration Building Block for Reliable Multicast Transport,*" Internet-Draft (Expired), Internet Engineering Task Force, December 2003.

[16] S. Kent and K. Seo, "*Security Architecture for the Internet Protocol,*" RFC 4301, Internet Engineering Task Force, December 2005.

[17] M. Baugher, R. Canetti, L. Dondeti, and F. Lindholm, "*Multicast Security (MSEC) Group Key Management Architecture,*" REF 4046, Internet Engineering Task Force, April 2005.

[18] M. Blaze and M. Strauss, "*Atomic Proxy Cryptography,*" Eurocrypt, 1998.

[19] R. Mukherjee and J. W. Atwood, "*SIM-KM: Scalable Infrastructure for Multicast Key Management,*" In Proceedings of the 29th IEEE Conference on Local Computer Networks (LCN 2004), Tampa, FL, November 2004.

[20] K. Y. Fung, "*Network Security Technologies Second Edition*", ISBN: 0-8493-3027-0, Auerbach Publications, pp. 133-136, 2005.

[21] S. Kent, "*IP Authentication Header (AH),*" RFC 4302, Internet Engineering Task Force, December 2005.

[22] S. Kent, "*IP Encapsulating Security Payload (ESP),*" RFC 4303, Internet Engineering Task Force, December 2005.

[23] A. Ivan and Y. Dodis, "*Proxy Cryptography Revisited,*" Network and Distributed System Security Symposium (NDSS), February 2003.

[24] M. Handley, O. Hodson and E. Kohler, "XORP: an Open Platform for Network Research," First Workshop on Hot Topics in Networks, October 2002.

[25] R. W. Smith, "*Advanced Linux Networking,*" ISBN: 0201774232, Addison Wesley Professional, June 2002.

[26] "*XRL Interfaces: Specification and Tools,*" XORP technical document, http://www.xorp.org/.

[27] "*Netfilter Hacking HOWTO,*" Netfilter technical document, http://www.netfilter.org.

[28] "*XORP Router Manager Process (rtrmgr)*, " XORP technical document, http://www.xorp.org/.

[29] "*XORP Routing Information Base (RIB) Process,*" XORP technical document, http://www.xorp.org/.

[30] "*XORP Forwarding Engine Abstraction,*" XORP technical document, http://www.xorp.org/.

[31] The Click Modular Router Project, http://www.read.cs.ucla.edu/click/.

[32] "*An Introduction to Writing a XORP Process,*" XORP technical document, http://www.xorp.org/.

[33] "*XORP MLD/IGMP Daemon,*" XORP technical document, http://www.xorp.org/.

[34] "*XORP PIM-SM Routing Daemon,*" XORP technical document, http://www.xorp.org/.

[35] J. Corbet, A. Rubini and G. Kroah-Hartman, "*Linux Device Drivers, Third Edition,*" ISBN: 0-596-00590-3, O'Reilly, February 2005.

[36] K. Wehrle, F. Pählke, H. Ritter, D. Müller and M. Bechler, "*The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel,*" ISBN: 0-13-177720-3, Prentice Hall, August 2004.

[37] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, "*Handbook of Applied Cryptography*" ISBN: 0-8493-8523-7, CRC Press, October 1996.

[38] R. Moskowitz and P. Nikander, "*Host Identity Protocol Architecture,*" Internet Draft (Work in Progress), Internet Engineering Task Force, March 2006.

[39] J. W. Atwood, "*An Architecture for Secure Multicasting*," Submitted, 2006.

[40] "*Netfilter Extensions HOWTO*," Netfilter technical document, http://www.netfilter.org.

[41] "*Modular Exponentiation*," http://www.answers.com.

[42] D. Ascher and M. Lutz, "*Learning Python, 2nd Edition*," ISBN: 0-596-00281-5, O'Reilly, December 2003.

[43] L. L. Peterson and B. S. Davie, "*Computer Networks: A System Approach, Third Edition,*" ISBN: 1-55860-832-X, Elsevier Inc., 2003.

[44] RSA Labs, "*A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths,*" RSA Labs Bulletin, http://www.rsasecurity.com/rsalabs/node.asp?id=2088

[45] D. E. Taylor, "*Survey & Taxonomy of Packet Classification Techniques*," Tech. Rep. WUCSE-2004-24, Department of Computer Science & Engineering, Washington University in Saint Louis, May 2004.