An IPSec–based Host Architecture for Secure Internet Multicast

Ran Canetti, Pau–Chen Cheng, Frederique Giraud, Dimitrios Pendarakis
Josyula R. Rao and Pankaj Rohatgi
IBM Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
email:{canetti,pau,dimitris}@watson.ibm.com, {giraud, jrrao, rohatgi}@us.ibm.com

Debanjan Saha
Bell Labs, Lucent Technologies
101 Crawford Corner Road
Holmdel, NJ 07733
email:{debanjan@dnrc.bell-labs.com}

Abstract

We propose a host architecture for secure IP multicast. We identify the basic components of the architecture, describe their functionalities and how they interact with one another. The fundamental design tenets of the proposed architecture are simplicity, modularity, and compatibility with existing protocols and systems. More specifically, we try to re-use existing IPSec mechanisms as far as possible, and extend them when necessary. We also discuss our experiences with implementing the proposed architecture on Linux.

1 Introduction

The Internet today supports a basic form of multicast service where a multicast group is identified by a Class D IP address [H95]. A receiver can join and leave the group by sending IGMP (Internet Group Management Protocol) [RFC1112, D91] messages to their local routers. To send datagrams to a multicast group, a sender need not be a member of the group. It can simply address the datagrams to the group address. It is the responsibility of the multicast capable routers to communicate with each other using multicast routing protocols and deliver the datagrams to all receivers who are members of the group. The multicast group is an open group and senders do not necessarily know the identities of the receivers in the group. Likewise, receivers do not have any mechanisms available to authenticate the identity of the senders or to verify the integrity of the received data.

Support for group membership control and for sender and data authentication is essential for many multicast applications. Maintaining confidentiality of the transmitted data is also required by some applications. Examples range from one-to-many scenarios such as news and data feeds (say, quotes of stock prices), audio and video broadcasts, or file and software updates, to more interactive scenarios such as electronic lectures, town-hall meetings and conferences. (See [Q98, CP99] for surveys of multicast applications and their security concerns.)

A lot of work has been done in designing *secure multicast* protocols (see the survey in [CP99]). Recently, the Internet Architecture Board (IAB) has chartered a working group within the Internet Research Task Force (IRTF) to study and develop standardizable protocols for secure multicast [SMuG].

Existing work on securing group communication concentrates on the task of group management and access control (see, for instance, [HM97a, HM97b, M97, STW98, WHA97, WGL98, HCD98, HCM98, BMS99]). More specifically, the focus is on distributing and maintaining a *group key* that is known to all legitimate members, but remains unknown to non-members. This group key is distributed by one or more *group controllers*, according to some pre-specified policy, and is then used to encrypt the group communication. Less attention is given to the internal design of a group member.

The focus of this paper is on the host architecture of a member of a secure multicast group. The member can be either a sender, or a receiver, or both. In the rest of the paper, we identify basic components of the architecture, outline their functionalities, and describe the interaction among them. The architecture is based on the IPSec protocol suite [KA98] and re-uses IPSec components (such as IKE and ESP). We have extended the IPSec architecture in several ways to accommodate the requirements of secure group communication.

Some of the main features of the proposed architecture are: simplicity, modularity and compatibility with existing systems and protocols. We identify the key components of the architecture and define the interfaces between them. This modular approach allows the development of alternative implementations of various components. In addition, our architecture can accommodate known proposals for group and key management. It is compatible with the secure IP multicast framework that is being developed by the Secure Multicast working group (SMuG) of the Internet Research Task Force [SMuG]. It is simple to incorporate within IPSec-compliant systems, with either small or no changes to the operating system kernels.

We are implementing the proposed architecture within the Linux kernel using the FreeSwan IPSec distribution [FSWAN] as the base. We describe this effort and our experience.

A preliminary version of this architecture appeared as an Internet Draft, draft-irtf-smug-sec-mcast-arch-00.txt at the Secure Multicast Users Group IRTF. This paper describes further internal details of the architecture and experiences with implementing its components.

Organization The rest of the paper is organized as follows. Section 2 reviews some salient security requirements of multicast applications, and describes how our work fits within the larger context of a comprehensive secure multicast solution. Section 3 presents our main design tenets. Sections 4, 5 and 6 describe the architecture. Section 7 discusses compatibility issues between secure multicast and IPSec and Section 8 describes experiments to validate the basic design decision to use IPSec and results of preliminary performance tests done as part of an ongoing implementation of the architecture. We conclude with some remarks in Section 9.

2 Background

2.1 Security requirements of multicast communication

We outline the salient security requirements of multicast communication. A more detailed discussion appears in [CP99, CG$^+$99].

- *Group membership control and confidentiality*: The goal is to ensure that the group communication is accessible only to legitimate group members. Group membership control is usually implemented by having a group key shared by all group members. All group communication is encrypted via symmetric encryption using this key. The architecture proposed here follows this approach.

  For many applications, group membership is likely to vary over time. It is often required that members leaving a group lose access to future group communication, and that members joining a group do not gain access to group communication that occurred before they joined.

  Note that in order to implement access control it must be possible to authenticate the identity of potential group members. This can be done using public-key certificates of potential members. A decision on whether to accept requests to join a group is usually made according to some predefined *policy*.

- *Group authentication*: This refers to the ability of a group member to verify that the received group communication originated from a source within the group. Note that any group member can impersonate the sender or maliciously modify the transmitted data. To achieve this, we follow the usual approach of implementing group data authentication using a dedicated key shared among all group members. All the communicated data is authenticated via Message Authentication Codes (MACs) using the shared key.

- *Individual source and data authentication*: This refers to the ability of group members to verify the authenticity of the data, and identity of the sender of data, even if the other group members are not trusted. This problem is inherently different from group authentication and from authentication of point-to-point connections, since single MAC based solutions (such as those used in IPSec [KBC97]) are not applicable here.

Other security concerns, like non-repudiability and anonymity, may also be relevant to some applications.

2.2 On the structure of secure multicast solutions

In order to place our work in context, we outline the general structure of secure multicast solutions.

Very roughly, a secure multicast solution involves the following entities. In addition to the group members (and would-be members) there are one or more group control entities. These consist of the group initiator and owner, and one or more policy and key servers. A solution can be viewed as consisting of two main parts:

- *Group control*: This part deals with the task of group management and access control, in particular, with distributing and updating the *group key* and other data that is relevant to securing group communication. Typically this part includes only communication between the control entities of the group and potential group members (and possibly some communication among the control entities).
- *Data handling*: This part deals with the communicated data itself. This includes the processing of data at the end hosts, data packet routing, and possibly en-route transformations. Typically the group control entities do not participate in this part at all.

This work handles both parts, from the point of view of a *group member.* It does not address the design of the group control entities; it is compliant with any such design.

3 Design Tenets

We describe the main design principles set forth in this work.

- *Independence from multicast routing*: The architecture does not interfere with the routing mechanism of data packets. Data packets may be routed via any multicast or unicast routing. For sake of simplicity and modularity of design, we recommend that the key management mechanism assume reliable communication. Yet, no specific mechanism for obtaining reliability is specified. Any reliable multicast or unicast mechanism (e.g., TCP) can be used.
- *Mimic IPSec architecture*: As in IPSec, we separate the modules that handle data from those that handle key management. Functions such as the generation, distribution and update of cryptographic keys are encapsulated in a key management module, that is placed in the "application layer" of the communication, and outside the OS kernel. This facilitates application specific operations, such as multiplexing of data for different users and certificate verification. The data handling part lies mostly in the IP layer, using IPSec modules. Yet, we introduce an additional level of data handling, in the interest of source authentication.
- *Use existing components*: We use existing components wherever possible. Specifically, we use IPSec's Encapsulating Security Payload (ESP) protocol for encrypting and authenticating data. The ESP protocol [KA98] is used for exchanging encrypted and authenticated multicast data. If confidentiality is not an issue or if additional authentication beyond what is provided by the ESP authentication mechanism is required (such as authentication of the IP header) then the AH protocol can also be used.

  Since IPSec was mostly designed for unicast, there are a few issues that arise when the ESP/AH protocols are used for multicast data. We discuss some of these issues in Section 7. Multicast-specific packet transformations may be introduced in the future.

  We also recommend using Internet Key Exchange protocol (IKE) as a component within the multicast key management module, for securing point–to–point communication between group members and the control entities.
- *Minimize modifications of the OS kernel*: The architecture is structured so that the new multicast security specific components are kept in the application space and the IPSec components that are currently in the OS kernel can be used for multicast security without modification. It is expected that with time, some of the multicast specific components may be moved to the kernel and IPSec components already in the kernel would be modified to better support multicast.
- *Flexibility in the choice of cryptographic algorithms*: By re–using the IPSec design for data transport, we retain the flexibility of using any data encryption and authentication algorithm which can be supported by IPSec.

  The new components introduced in this document, namely the Multicast Internet Key Exchange Module (MIKE) and the Source Authentication Module (SAM) provide frameworks which are flexible enough to support most group key management and source authentication schemes.

4 System Architecture

4.1 Motivation

We present a bird's eye view of the architecture, its components, and the data flow. We begin by motivating the decision to base the architecture on the IPSec design.

The IPSec architecture. In a nutshell, the IPSec protocol suite consists of the following components: The *Internet Key Exchange (IKE)* protocol is an application layer protocol for agreeing on a *security association*

(SA) between the communicating peers. Among other data, the SA includes shared session keys for authentication and encryption of data. The *Authentication Header (AH)* and *Encapsulating Security Payload (ESP)* are two protocols for authenticating and encrypting data packets, using the information in the SA for the relevant session. Both AH and ESP are IP–layer protocols and operate on a packet by packet basis. The communication between the application–layer IKE and the IP–layer AH and ESP is done via a *security association database.*

Why IPSec–based design? In contrast to previous designs of secure multicast host architectures (e.g., [CE$^+$99]), that remained exclusively in the application layer of the communication, we have based our design on the IPSec protocol suite.

One main reason for using IPSec is that it provides a set of protocols that will shortly be available on practically every security–conscious host on the Internet. We see great benefit in using existing protocols for secure multicast, both from the point of view of the system designer (who does not need to design such protocols from scratch) and from the point of view of the user (who does not need to have duplicate code and can have a unified standard security mechanism for unicast and multicast). Another reason for using IPSec is its enhanced efficiency in processing data, being an IP–layer protocol.

On the down side, using IPSec ties the design to existing protocols and forces us to deal with compatibility problems with existing implementations. We elaborate on such problems in Section 7. However, even in light of these problems, we regard the IPSec–based design as a viable and preferable option.

4.2 Architecture

4.2.1 Overview We present a high level overview of the architecture. As in IPSec, we partition the "control plane" functions (i.e., key management and policy mechanisms) from the "data–plane" transformations. The "control plane" functions are placed in the application layer. The main component here is called *Multicast Internet Key Exchange (MIKE).* This is a generic name for the component that is responsible for communicating with the group controller(s), both on joining and leaving the group, and for periodic updates of the group security data (such as key updates). The MIKE component generates and updates a *multicast security association (MSA)* database. This database contains the necessary data for members in the relevant group. In particular, for multi–user hosts the GSA specifies which users are members of the group.

In contrast with IPSec, where the data transformations are done exclusively in the IP layer, we partition the data–handling mechanism into two separate components. The first component, called *Source Authentication Module (SAM),* sits in the application layer and deals with transformations for authenticating the source and contents of the data. (Recall that the AH or ESP transforms do not guarantee individual source and data authentication for group communication.) One benefit of placing SAM in the application layer is avoiding the need to modify the operating–system kernel. In addition, transforms for source authentication may benefit form the ability to process the data in larger frames, *before* it is partitioned into IP packets by the sender, and after the frames have been re–assembled by the receiver. (This additional option may be most viable when *reliable* transmission is guaranteed.)

The second stage for data processing is the ESP transform at the IP layer. Here data may be encrypted/decrypted using the group key. Possibly the data is also authenticated using the group key; this guarantees group authentication.

Both SAM and ESP take the information needed for handling the data from the MSA database. (We remark that a similar notion of security association for multicast is proposed in [HM99]. The two notions may indeed be unified.)

4.2.2 Architectural Details We now describe the architecture of a member of a secure multicast group. The member could either be a data recipient or a data sender or both. From the application's perspective, the secure multicast framework provides a simple API for using secure multicast. This API is logically partitioned into the Data API which deals with sending and receiving multicast data securely and the Control API which deals with the process of joining and leaving a secure multicast group and the associated access control and key update functions. The block diagram of a secure multicast framework on a host is shown in Figure 1.

- *MIKE – Multicast Internet Key Exchange*: This module is responsible for key management and implements the Control API which permits applications to join and leave secure multicast groups. This module resides in the application layer, outside the OS kernel. It interacts with the MIKE modules of the group controller(s), and generates and maintains a Multicast Security Association (MSA) that contains:
  - Group keys for encryption/decryption and authentication of data (via the AH/ESP modules).

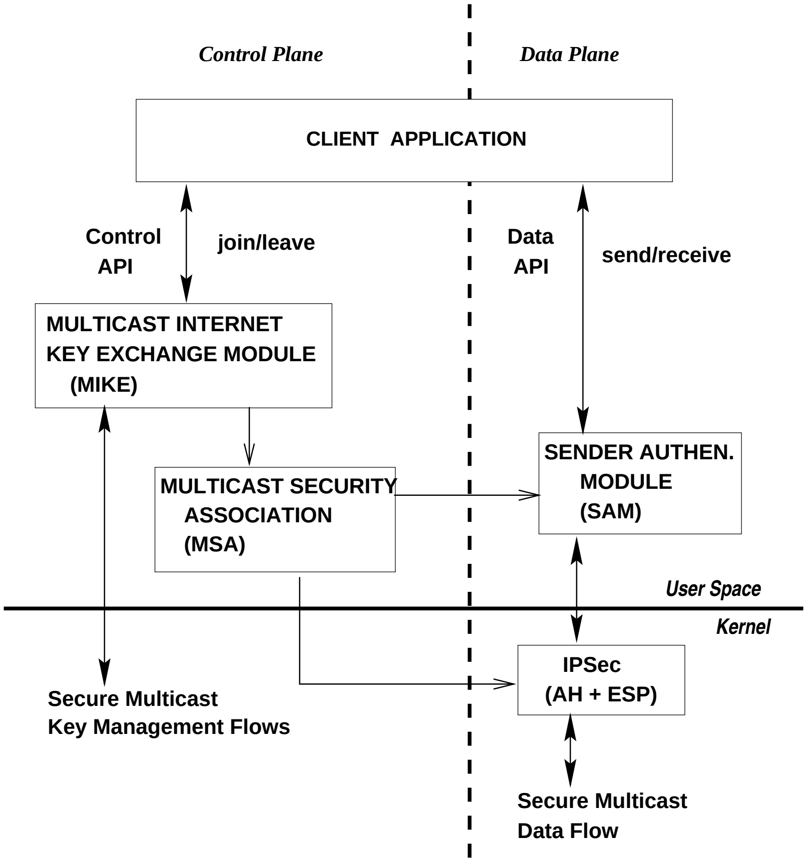
Control Plane
Data Plane
CLIENT APPLICATION
Control API
join/leave
Data API
send/receive
MULTICAST INTERNET KEY EXCHANGE MODULE (MIKE)
MULTICAST SECURITY ASSOCIATION (MSA)
SENDER AUTHEN. MODULE (SAM)
User Space
Kernel
IPSec (AH + ESP)
Secure Multicast Key Management Flows
Secure Multicast Data Flow

Figure 1. Block diagram of secure multicast host architecture.

- Signing/Verification keys for source authentication of data by the SAM module, described below.
- Other information regarding the connection, as in an IPSec SA.
- A list of applications that are members of the group (relevant for multi–user hosts only).

In order to make a MIKE specification complete, the MIKE modules within the group controller(s) need to be specified. See more discussion on the design of MIKE in section 5.

An additional functionality of MIKE is periodic updates of the MSA, whenever group keys or keys used by SAM are changed.

- *IPSec modules: AH/ESP*:

These are the IPSec modules that reside in the OS kernel and deal with encryption/decryption and authentication of data packets. These modules provide encryption/decryption and group authentication of incoming or outgoing multicast data. Data is encrypted with the group key by the sender(s) and decrypted using the same key by receivers. The ESP header remains as defined in the unicast case; the protocol header preceding the ESP header will contain the value 50 in its Protocol (IPv4) or Next Header (IPv6, Extension) field and its destination IP address will be the IP multicast group address, a class D address. Thus, the packet will be forwarded to all members of the group by routers supporting multicast delivery. ESP can be used in conjunction with the ESP authentication option. In principle, ESP for multicast traffic can be used either in transport or tunnel mode, although transport mode is clearly more appropriate in an environment where most participating members are end hosts.

For multicast data authentication different techniques can be used depending on whether group or individual sender authentication is desired. For group authentication, the protocol designed for unicast IP security, namely the IP Authentication Header (AH) [KA98] and/or the authentication option within ESP are sufficient. All members share a common, symmetric authentication key which is administered by the group controller and which is used to generate the message authentication code (MAC). The AH header remains as defined in the unicast case; the protocol header preceding the ESP header will contain the value 51 in its Protocol (IPv4) or Next Header (IPv6, Extension) field and its destination IP address will be the IP multicast group address, a class D address. The SPI value is selected, as for ESP, by the group controller. (See more discussion on the assignment of the SPI in Section 7.)

- *SAM – Source Authentication Module*: This module is responsible for the transformations that enable authenticating the source of received data and possibly for replay protection. Scalable source authentication may involve operations that span more than a single packet, both for outgoing and incoming data. Since UDP frames are good candidates for a basic unit for authentication, it seems reasonable to place this component in a higher layer in the protocol stack, i.e., above UDP. It could either be part of the kernel or kept in application space. If the kernel were to be modified, then this module should be part of a "Secure Multicast UDP" layer which could replace UDP. This "Secure Multicast UDP" layer could also implement the mult–user access control functionality which we will discuss later in Section 7.1. Apart from the disadvantage of requiring kernel modifications, placing SAM in the kernel will make it inflexible in the choice of authentication schemes and will also result in the kernel being entrusted with the user's signature keys. On the other hand, if the SAM module is kept in the application space then source authentication will not be transparent to applications and existing multicast applications will need to be re–written to invoke the relevant functions of the SAM module. But if a SAM module is well designed, then this burden should be comparable to moving from regular sockets to SSL in the case of unicast, and such a change should be equally acceptable to the application developer community. Therefore it is advantageous to keep the SAM module in the user space for now.

  The internal structure of the SAM depends to a large extent on the source authentication mechanism used. Several source authentication mechanisms exist, some are based on public key signatures which may be applied on a group of packets via a variety of mechanisms, and others are based on symmetric authentication with multiple keys. (See the survey of [CP99].) In particular the source authentication mechanism proposed in [M99], based on a scheme of [WL98] or a scheme described in [R99] are options for realizing SAM.

  An additional potential functionality of SAM is to provide replay protection for data, in case that the IPSEC replay protection mechanism is turned off because of multiple sender problems.

  We shall discuss the SAM module in greater detail in Section 6.

4.3 Data and Control Flows

Typical operation of the system is as follows.[1] The member initiates a secure multicast session by invoking the join operation in the Control API which is implemented by MIKE. This enables the member to register in the group as a sender or a receiver or both. Subsequently, the member is able to send and receive datagrams securely to the multicast group using the send/receive functions of the Data API. All group key management, data encryption/decryption and group/source authentication functions are managed by the secure multicast framework and are transparent to the member. If at some later point in time the member decides to leave the secure multicast group then this is done by invoking the leave operation in the Control API. This action eventually results in the termination of the member's ability to securely send/receive messages to the group.

We now outline the data and control flows in more detail.

- *Control Flows*:
  - *Client Join*: The application invokes MIKE to join a multicast group. At a minimum, the application must identify the group that it wishes to join and provide information as to the authentication required (e.g., whether or not source authentication is required and if so, the sources it will trust).

    MIKE then performs the process of registering with the group controller(s), sets up a Multicast Security Association (MSA), invokes a standard registration mechanism for the underlying IP multicast group and enables the ESP/AH and SAM modules to start processing data. (In multi–user hosts it may be that an MSA for this group already exists,

[1] Here we assume that the data is sent/received via either reliable or unreliable IP multicast. It is noted, however, that the security mechanism described here can be used even when the data is being routed via point–to–point connections, such as TCP.

with another application listening to it. In this case the MSA is updated to include the joining application.)

The registration process will inevitably include communication with the group controller(s) and this communication will require mechanisms for authentication of the parties as well as confidentiality of the information exchanged. This communication, its authentication and encryption mechanisms should be dealt with within the MIKE module.

At the end of the join process, the Multicast Security Association (MSA) database needs to be updated. The relevant information from the MSA is then pulled by the ESP/AH and SAM modules.

- *Key update*: Key updates messages are internal MIKE messages and are not part of the high–level architecture. These messages are authenticated and encrypted separately, as specified in MIKE. A special class of key update messages consist of member expulsion messages in which the controller expels the member from the group. The expulsion process is dependent on the specifics of the Key Management Protocol, but should result in the member being cryptographically incapable of sending/receiving messages from the secure multicast group. In this case MIKE module should treat an expulsion message like a member leave request without the need to contact the controller(s).
- *Client Leave*: First MIKE is invoked to de–register with the group controller(s). Next the Multicast Security Association database is modified to reflect the fact that the client leaves. (If there are no more applications that are listening on the same group then the entry may be deleted or marked stale). Finally, the host executes the standard procedure for leaving the underlying IP multicast group.

- *Data Flows*:
  - *Sending of data*: If source authentication is not needed, then data is transmitted directly via UDP (or a reliable multicast layer) and the IPSec module in the IP layer, with the IP multicast group in the destination address.

    If source authentication is needed then the data is first transferred to the SAM. There the data is processed for source verification. (The keys for performing these operations are obtained from the MSA.) Next, the data is directed to the AH/ESP transformations in the kernel. These transformations are executed with the group keys that appear in the MSA. Finally the data packets are sent on the channel in the standard way.
  - *Receipt of data*: Incoming data packets are first processed by IPSec's AH/ESP transformations in the kernel. There the data is decrypted and group authentication is verified. Next, the data stream is processed by the SAM and source identity is authenticated, if needed. Finally, the data is handed to the calling application. (In a multi–user host it should also be decided, based on information in the MSA, whether the application is eligible to receive the comunication of the relevant group.)

5 MIKE: Design Guidelines

Although the design of MIKE deserves a separate document, we suggest some requirements for MIKE, describe architectural principles that will allow MIKE to meet these requirements and yet be flexible enough to accommodate a variety of group key management techniques. This provides an interface for plugging in different group key management modules into MIKE.

It is stressed that the discussion below does not attempt to address the internal design of the group control entities. Instead, it is focused on the requirements from the point of view of a group member.

5.1 Functionality of MIKE

We list salient requirements for the MIKE module.

1. MIKE should support the simple scenario where there is only a single group controller that communicates with all group members. More complex environments where intermediate servers facilitate the communication may also be supported.

2. MIKE should support maintaining a set of keys (for symmetric encryption and authentication) shared among all group members. In addition, MIKE will help in forwarding the public verification keys of the group controller and of senders in the group, to support source authentication by group members. (Note that these verification keys may be different from the long–term certificates of these parties.) MIKE could obtain these keys directly from the group controller(s) or from some other repository using a protocol such as LDAP [LDAP].

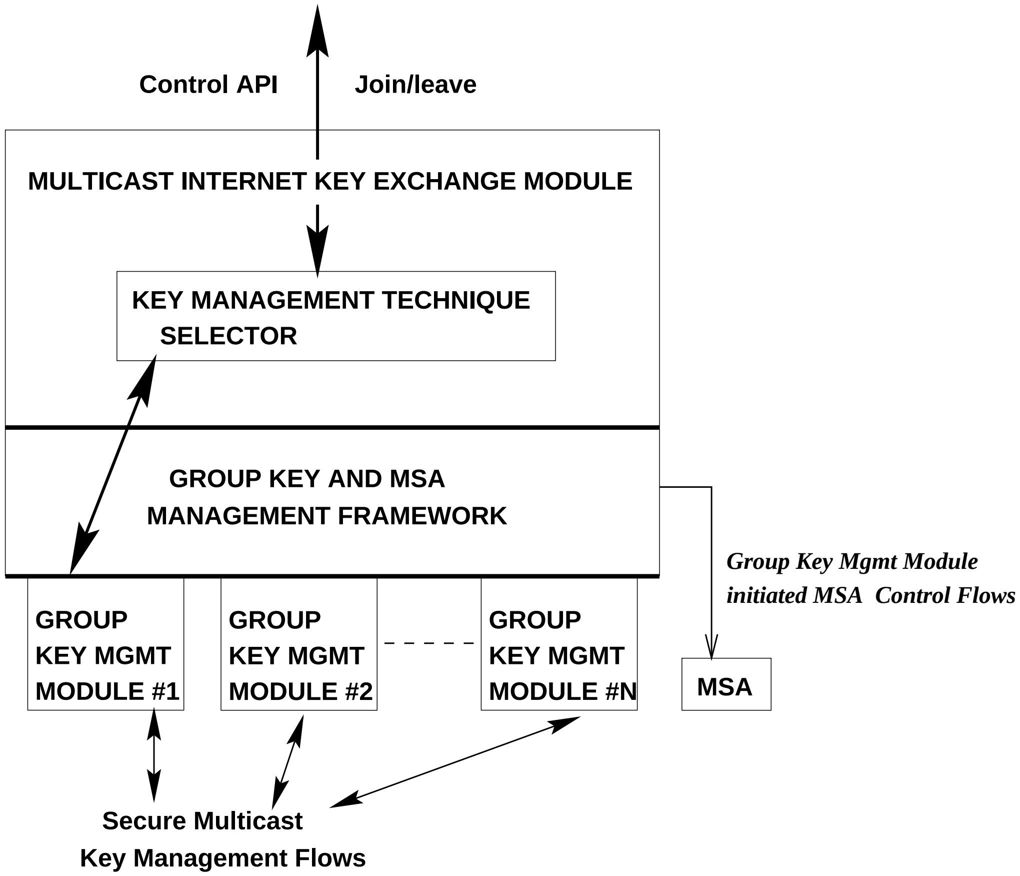
Control API
Join/leave
MULTICAST INTERNET KEY EXCHANGE MODULE
KEY MANAGEMENT TECHNIQUE SELECTOR
GROUP KEY AND MSA MANAGEMENT FRAMEWORK
Group Key Mgmt Module initiated MSA Control Flows
GROUP KEY MGMT MODULE #1
GROUP KEY MGMT MODULE #2
GROUP KEY MGMT MODULE #N
MSA
Secure Multicast Key Management Flows

Figure 2. Block diagram of MIKE.

3. MIKE will be placed in the application layer of the communication, and outside the OS kernel.

4. MIKE should be flexible enough to accommodate any reasonable multicast group key management solution.

5. MIKE should be able to deal with multi–user hosts. In particular, the MSA will contain information on which users are members of each secure multicast group.

An architectural block diagram of MIKE is shown in Figure 2.

5.2 Some design guidelines for MIKE

This section suggests some design guidelines for a group key management module to be plugged into MIKE. We stress simplicity and re–use of existing protocols and standards such as IPSec and IKE where possible. In particular, we propose using IPSec to set–up secure channels for all point–to–point communication between the host and the controller(s). Of course, other solutions that provide secure channels (e.g., SSL or proprietary communication protocols) can be used instead.

1. Point–to–point communication between a group member and the controller (say, for group registration and de–registration) will be secured via a standard IPSec connection established by IKE. This connection will provide confidentiality as well as authentication of the information exchanged. In particular, the group keys and additional information will be transmitted as *data* in the secure connection. The IPSec SA between a group member and a controller will be short–lived, and will generally not last throughout the lifetime of the multicast group.

2. Key update messages will be transmitted from the group controller to the members using an abstract transportation mechanism, called ”Reliable Multicast Shim” (RMS). This abstract mechanism provides reliable multicast, in the sense that any message transmitted via the RMS is guaranteed to reach all group members.

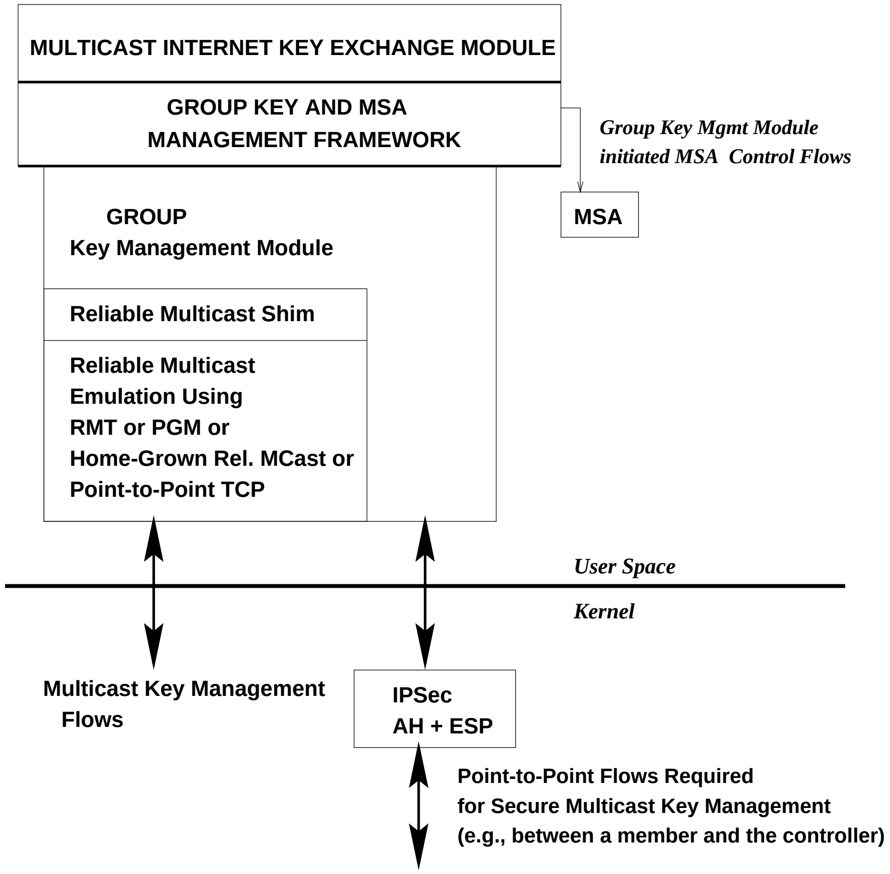
MULTICAST INTERNET KEY EXCHANGE MODULE
GROUP KEY AND MSA MANAGEMENT FRAMEWORK
Group Key Mgmt Module initiated MSA Control Flows
MSA
GROUP Key Management Module
Reliable Multicast Shim
Reliable Multicast Emulation Using RMT or PGM or Home-Grown Rel. MCast or Point-to-Point TCP
User Space
Kernel
Multicast Key Management Flows
IPSec AH + ESP
Point-to-Point Flows Required for Secure Multicast Key Management (e.g., between a member and the controller)

Figure 3. Suggested design of MIKE.

The RMS abstraction can then be implemented via any available reliable multicast mechanism (such as those developed in [RM]), or alternatively via point–to–point reliable communication (TCP).

3. Key update messages sent by the controller will have a special format. In particular, they will be authenticated using public–key signatures that are verifiable using a public key that is handed to the members at registration time. MIKE will implement its own signature verification mechanism.

The suggested design guidelines for MIKE are sketched in Figure 3.

5.3 An example

In the preceeding sections we outlined some requirements and guidelines for the MIKE module, without explicitly describing how existing group key management techniques and implementations could be modified to fit within the MIKE framework in conformance with these requirements and guidelines. In this section we give an example of one possible multicast key management module which conforms to the MIKE requirements.

The example module is based on a few modifications to the multicast key management module in IBM's Toolkit for Secure Internet Multicast [CE$^+$99]. The original toolkit employed a scheme by Wallner et al, [WHA97] for group key management and SSL for unicast connections between the client's key management module and the group controller. In the example module, the SSL interactions between the client's key management module (MIKE) and the group controller would be replaced by IPSec connections. In the original toolkit, signed key update messages were multicast by the group controller over a reliable multicast channel. In the example, these signed messages would be transported over the "Reliable Multicast Shim". These together with other minor changes would be sufficient to make the toolkit comply with the MIKE guidelines.

6 Interfaces of SAM

In this section, we briefly describe the interfaces of SAM, to the application and to the communication layer. We do not describe an implementation of SAM; implementations may vary widely and are to a large extent independent of the overall architecture. We believe that these interfaces are general enough to accomodate a variety of mulitcast authentication schemes and applications, and thus could serve as a basis for the standardization. From Figure 1, it is clear that the interface to SAM consists of a SAM Data API that is presented to secure multicast applications and a specification of the format of data blocks which are sent or received by the SAM layer to the communication layers below.

6.1 SAM Data Format

Data flowing between the SAM module and the underlying communication modules is organized into blocks or frames. The basic idea is that on the sending end, outgoing data supplied by the application is packaged together with authentication data into one or several SAM frames which are passed to the communication layers below. On the receiving end, SAM frames are delivered by the communication layers to the SAM module which verifies the authentication information contained within the SAM frames and provides the raw authenticated data back to the receiving application, together with the identity of the source.

Each SAM frame consists of several fields as shown in Figure 4. These include the Session Identifier, the Source Identifier, the Sequence Number, the Data Payload and the Authenticator fields. We now describe the functionality of each of these fields in more detail.

6.1.1 Session Identifier Field The Session Identifier is a numeric field which uniquely identifies the secure multicast session. The purpose of this field is to bind the Data Payload with a particular secure multicast session. It is expected that for some applications, senders may use the same long term signature key for signing packets for several different multicast sessions and this binding is necessary to prevent out–of–context substitution attacks. Our protoype currently implements this as 32–bit numeric field chosen by the group controller, but a more robust structure (possibly including a combination of group controller id and a group–controller assigned session id) would be required for larger scale deployment and for standardization.

6.1.2 Source Identifier In scenarios with multiple senders, the source identifier field provides a succinct way to identify the purported sender of the packet, so that the authentication mechanism corresponding to the purported sender is used to verify the source of the packet. In our prototype this is currently implemented as a source IP–address/port–number tuple. Again, for standardization purposes, this needs to be generalized to handle other types of Source Identifiers.

6.1.3 Frame Sequence Number The frame sequence number field is a numeric field containing the sequence number of the SAM frame with respect to the flow of SAM frames sent out by the sender (as identified by the source identifier) for the current session (as identified by the session identifier). Each sender maintains the sequence number for its own own flow by picking a starting sequence number for its frame and then subsequently incrementing the sequence number on each successive outgoing frame. The SAM module on the receiver side can optionally use these sequence numbers to implement replay protection and re–ordering of received SAM frames (if required by the source authentication scheme) by employing well known sliding window based techniques.

6.1.4 Data Payload This field holds the actual data payload that the sender needs to send in an authenticated manner.

6.1.5 Authentication Information This field holds the authentication information which is required by the receiver to authenticate the source and contents of the Data Payload, Session Identifier, Source Authentifier and Frame Sequence Number fields in SAM frames within the flow from the sender. To allow maximum flexibility in the choice of authentication schemes, there is no requirement that the authentication information carried within a SAM frame be either sufficient to or restricted to authenticate the contents of that frame, it could even carry authentication information related to other frames. All that is required is that SAM layer should have authenticatated each of the required fields within a SAM frame before passing the Data Payload to the application layer. Therefore, in cases where authentication information carried within a SAM frame and all previous frames is not sufficient to authenticate the frame, then depending on the authentication scheme, the SAM module may need to buffer this frame till the authentication information arrives or drop the frame as the frame is no longer within the sequence number window.

6.2 SAM Data API

The SAM Data API draws heavily from Netscape's SSL API [SSLRef]. This is because in the networking stack, the placement of SAM is quite similar to the placement of SSL. The Netscape's API was selected as a model for the SAM Data API because of its clean modular design that abstracts away most networking and memory management details. Using a similar design for SAM permits us to create a SAM module that focusses only on its designated task, i.e., source authentication, without having to handle extraneous issues such as networking, heap management etc. Another benefit of this

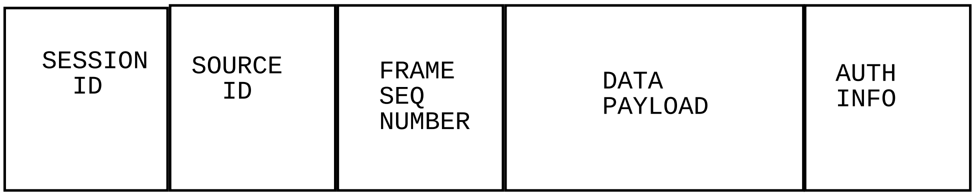

Figure 4. SAM Data Format

appraoch is that the same SAM module can work with a variety of different underlying transport mechanisms.

At the core of the SAM Data API is the concept of a SAM context, a data structure which holds all information revelant to source authentication (including current state) for a particular secure multicast session. The SAM Data API provides APIs for intializing and maintaining components of the SAM context as well as an API for sending and receiving authenticated data from the secure multicast session identified by the SAM context. We now describe these two types of APIs in more detail.

6.2.1 SAM context component management For modularity, the SAM context consists of an I/O context, a memory context, a user context and a group context. The I/O context deals with the multicast data input and output functions, the memory context deals with the platform and/or application specific memory management functions, the user context deals with information specific to the application as a sender and/or receiver within the multicast group and the group context deals with information related to the entire group. We now describe each of these contexts and their management in greater detail.

- **SAM I/O context**: The SAM library provides prototypes for functions to read and write data blocks to the underlying communication layer. These functions are *read*, *readfrom* and *write*. The *read* function reads data from the communication layer into a supplied buffer. The *readfrom* function is similar but in addition to receiving data into a buffer, the *readfrom* call also provides a hint as to the identity of the sender (e.g., IP–address/port–number of sender). The *write* function sends a buffer of data down to the communication layer. SAM assumes that pointers to the *read* and *write* functions are blocking and internally SAM uses only these functions for communciations. *The actual read, readfrom and write functions are provided by the application and registered to the SAM context*. This modular approach allows the SAM layer to be oblivious to the specifics and internals of the communication mechanism being used for the group communication. Accordingly, the SAM context management API provides a set of functions to register the application supplied *read, readfrom and write* functions with a SAM context.

- **SAM memory context** As with the I/O context, the SAM library provides prototypes for standard memory mangement functions such as *alloc, free* and *realloc* and provides an API to register these application supplied functions with a SAM context.

- **SAM user context** The SAM user context data structure contains user specific information relative to the secure multicast group. This includes the user identifier, the current outgoing sequence number and the user's authentication information which includes a handle to the source authentication algorithm and key to be used for computing authentication information for outbound SAM frames. The user context API also provides for functions to intialize and maintain this information. It is expected that the MIKE module would invoke these functions to manage this information.

- **SAM group context** This data structure holds group specific information such as a session identifier and a list (or a hash table) of registered senders and their algorithms, replay windows and keys to be used to authenticate SAM frames from each of the registered senders. Again, the SAM group context API provides for functions to initialize and maintain this information. It is expected that the SAM group context will be managed by MIKE either via direct communication with the Group controllers or via some sort of LDAP [LDAP] access to a repository of authentication keys.

6.2.2 SAM context data flow API Once the SAM context components are initialized, a SAM context can be used by an application to send and receive authenticated data on the corresponding multicast session. The main data communication functions available to the application are

- **sam_readfrom**: This function is to be used by applications to receive source authenticated data from the secure multicast group. When provided with a valid SAM context, a data buffer and a source identifier placeholder, this function reads incoming data from the secure multicast group identified by the SAM context, authenticates it and places raw authenticated data into the supplied data buffer and the identity of the source into the source identifier placeholder. This function blocks till authentic data is received.

- **sam_write**: This function is to be used by applications to post authenticated data to the secure multicast group. When provided by a valid SAM context and a data buffer, this function posts the data contained in the data buffer together with user authentication information to the secure multicast group.

7 Compatibility with IPSec & IKE

This section discusses some compatibility issues of secure multicast with the design of the IPSec protocol suite.

7.1 Granularity of Access Control: Host vs. Application

By design, IPSec is ideally suited for securing traffic between two hosts. Securing traffic between applications requires hosts to implement additional control mechanisms to create and maintain SA's at the granularity of applications, i.e., to create and maintain proper associations between SA's and applications' <protocol, port> tuples [KA98, CGHK98]. Similarly, multicast over IPSec works best when the granularity of access control or group membership is at the level of hosts. However, if the granularity of access control is at the level of user applications, then IPSec by itself is not sufficient. This is not a problem for most hosts on the Internet which are single–user systems. However in multi–user systems where multiple users could belong to the same secure multicast group, additional mechanisms need to be implemented by the host to ensure that IPSec protected multicast traffic flows are initiated and delivered to only those applications which belong to the multicast group. This requires close cooperation between MIKE and the system.

An ideal solution involves making changes to the host kernel so that:

1. The multicast SA's for the multicast group are only associated with the specific UDP port used by the group.
2. Only applications which are current members of the multicast group can send/receive packets through the group's UDP port.

This implies a new control mechanism in the UDP layer which controls access to the port on the basis of multicast membership information received from MIKE.

A less intrusive, but inferior solution which does not require kernel modifications would be put the same control mechanism in a system daemon process. This process joins a secure multicast group once on behalf of user applications and perform multiplexing and access control on outgoing and inbound data. By binding to the group's UDP port exclusively the daemon can ensure that no other application can subvert the daemon's control on the data flow.

7.2 Identification of Multicast Security Associations and SPI assignment

In the Internet Protocol, a Security Association (SA) is uniquely identified by the combination of the destination address, Security Parameter Index (SPI) and the protocol used (e.g., AH, ESP). As stated in the IPSec architecture document [KA98], the destination address can be either unicast or multicast; the definition of an SA remains the same.

In unicast SAs, in order to avoid potential conflicts of SPI values, receivers are responsible for assignment of the SPI. Since in the multicast case there are multiple destinations, all within the same multicast destination address, such an approach is impractical since it would require coordination by all receivers. Selection by the sender would also be problematic, especially in the case of multiple group senders.

Within our framework, a reasonable solution to the problem is to utilize the benefits of the centralized controller by requiring that the group controller selects the SPI for each multicast group and communicates it to members, senders and receivers, during registration. Selection by the controller guarantees that the SA is uniquely identified by the combination of the SPI value, the multicast group address and the protocol. (A similar solution is suggested in [HM99].)

As stated in [KA98], multiple senders to a multicast group MAY use a single Security Association (and hence Security Parameter Index) for all traffic to that group. In that case, the receiver only knows that the message came from a system knowing the security association data for that multicast group. Multicast traffic MAY also use a separate Security Association (and hence SPI) for each sender. The assignment of SA's to senders can be done by the group controller.

7.3 Sequence Number Handling and Replay–Prevention

Both ESP and AH headers contain a mandatory, monotonically increasing, sequence number field intended to provide anti–replay protection. Processing of the sequence number is at the discretion of the receiver, but the sender MUST always transmit it. The sender's and receiver's counters have to be initialized to 0 when the SA is established and the first packet of that SA will have a sequence number of 1.

In the case of multiple senders using the same security association (and hence the same SPI value) consistency and monotonicity of the sequence number cannot be guaranteed. Hence, anti–replay service SHOULD NOT be used in a multi–sender environment that employs a single SA. Multicast security implementations should thus ensure that receivers do not perform sequence number processing and verification.

We see two possible solutions to provide anti–replay protection:

(1) Using multiple SAs, one for each sender. (All these SAs may be part of a single MSA.) This can provide a weak form of replay protection (against outsiders).

(2) Putting anti–replay protection in some higher level module such as SAM. This solution requires application–layer framing of multicast messages.

These alternative solutions may better suit different applications.

7.4 Allowing IPSec processing of multicast packets

Some current implementations of the IP protocol stack will discard any IP packet with a class D destination address and a "protocol" field that is not UDP. Such implementations need to be changed to support IP–multicast packets protected by IPSec.

8 Validation of Architecture

A prototype of the proposed secure IP multicast architecture is currently under development. Components on the data path such as SAM and UDP over IPSec have been implemented and tested. Components along the control path such as MIKE are still being developed.

Even at this early stage of development we were able to test the feasibility of using IPSec to secure multicast traffic. We also were able to evaluate the performance impact of adding source authentication, group authentication and confidentiality to our architecture.

In this section, we describe these feasibility and performance tests and present preliminary results.

8.1 Feasibility Tests

Our test bed consists of IBM PCs running Red Hat Linux 5.1 with kernel version 2.0.35 and with Freeswan version 0.91 implementation of IPSec. Freeswan consists of two daemons, klips and pluto, which are started at boot time.

Klips, in the kernel, encrypts/encapsulates outgoing packets and decrypts/decapsulates incoming packets. It is implemented as a *virtual* network interface and is configured as any other network interface. This virtual interface is *attached* to a physical interface which handles the traffic flow to/from the network. In the network stack, the IPSec packet handler is piggy–backed onto the associated physical interface packet handler. Klips also provides an API to set up the security associations that map destination addresses with the proper IPSec security association.

The other daemon, pluto, runs in user space and manages the keys and their updates. Refer to the Freeswan documentation [FSWAN] for additional details. Pluto was *not* used in our test since we configured the IPSec SAs manually.

8.1.1 System set up and configuration Figure 5 shows the system setup for testing the Linux *FreeSwan* IPSec package.

We first tested the proper configuration of IPSec, by testing the 2 modes of IPSec encapsulation, transport and tunnel, as described in the klips installation files.

We connected two security gateways A and B, running Linux with Freeswan, via an Ethernet link, configured as network interface *eth0* on both machines, and representing the public Internet. Each gateway was also connected via a token ring network to a LAN representing a private Intranet, designated as C and D, respectively. A snooping box E running Solaris 5.1 observed all Ethernet traffic.

The IPSec virtual interface *ipsec0* was attached to eth0 and configured So that all network traffic between C to D would be secured across the public Ethernet. We ran the test scenarios of the Freeswan sample files and verified that packets between C to D were properly encrypted by A and B.

8.1.2 Test Setup We then proceeded to test the multicast support of Freeswan. For this set of tests we did not change our previous configuration and ipsec0 was still bound to eth0 and traffic through the Ethernet was still encrypted. But we disconnected the 2 subnets, which were not needed. We installed, on gateways A and B, a simple multicast client/server application that opened a UDP socket, joined a given multicast group on the eth0 interface and then looped on sending multicast

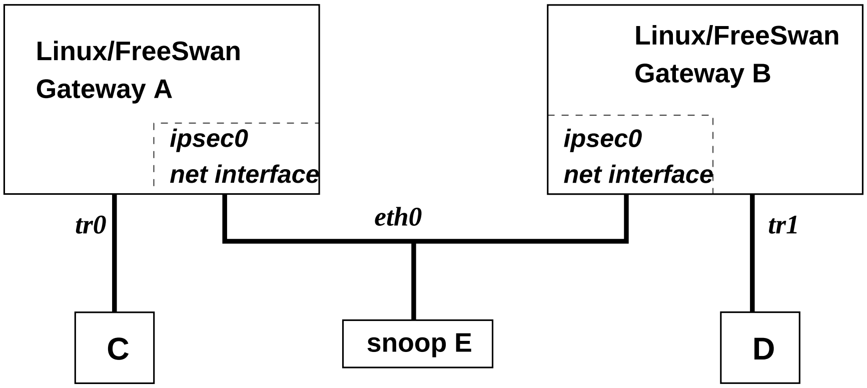

Figure 5. System Setup for testing FreeSwan/IPSec on Linux.

messages and listening for messages. Again, a snooping box on the Ethernet eavesdropped on the traffic across the wire.

8.1.3 Feasibility Test Results We made the following observations:

- *Handling of multicast addresses by Linux* Multicast addresses are Class D addresses identified by first 4 bits equal to 1110 with the remaining 28 bits specifying a multicast group id. This translates to addresses in the range from 224.0.0.0 to 239.255.255.255.

  In the process of configuring the Linux routes to run our tests, we noticed that Linux treats multicast addresses in the 224.0.0.0 range differently from others. In theory, all multicast address routes should use the same network mask of 240.0.0.0, since they are not real network addresses. But in practice, this is only true of the 224.0.0.0 range. Other multicast ranges, such as 235.0.0.0, cannot use the netmask 240.0.0.0 when the routes are set, or else the “route add” command fails; we used the netmask 255.0.0.0.

- *Handling of received multicast IPSec packets by Linux* Linux successfully processed and delivered received multicast IPSec packets intended for the all–host multicast group at address 224.0.0.1. But it dropped packets intended for other multicast groups.

  The flow–of–control in Linux and FreeSwan code is as follows:

  1. $ip\_rcv()$ in $ip\_input.c$: This function retrieves an IP packet from the system input queue; the IP packet is in a kernel buffer (skb) containing a *device id* representing the physical network interface on which the packet was received; in our test the id is eth0. The function massages the skb and passes it to the next protocol layer for further processing. If the packet is an IPSec packet, it is passed to the $ipsec\_rcv()$ function, which will call $esp\_rcv()$ in $ipsec\_esp.c$ (or $ah\_rcv()$ in $ipsec\_ah.c$.).
  2. $esp\_rcv()$ in $ipsec\_esp.c$: This function processes the skb further, and replaces the device id (eth0) in the skb buffer with the virtual interface identifier (ipsec0). The skb packet is then decapsulated and decrypted as needed. Finally, the resulting skb is placed back on the system input queue and control is returned to $ip\_rcv()$ in $ip\_input.c$
  3. $ip\_rcv()$ in $ip\_input.c$: This function retrieves the skb from the system input queue again; note that the device id in the skb now is ipsec0. The function examines the destination address of the packet. If the address belongs to a multicast group, the function checks if the network interface identified by the device id in the skb has registered interest in that group (via a multicast join on that interface). If not, the packet is dropped. Otherwise, processing continues and the the resulting skb packet is passed to the next protocol layer, in this case UDP, for further processing and delivery to the application.

  One problem that we encountered here is that in the Linux socket API for IPv4, multicast join requests can only be associated with physical interfaces, such as eth0, but not with virtual interfaces, such as ipsec0. This causes $ip\_rcv()$ to drop the decapsulated ipsec packet since the packet is associated with an interface (ipsec0) which is not registered with the multicast group.

We developed a temporary work–around for this problem:

- in the $esp\_rcv()$ function of $ipsec\_esp.c$: We patched the klips code to simply restore the skb device field id in the skb to the physical device id before placing the skb back on the system input queue.
- in $ah\_rcv()$ function of $ipsec\_ah.c$: The same patch (as above) can be used, since the $ah\_rcv()$ processing is nearly identical to that of $esp\_rcv()$.

- **Sharing of UDP ports** UDP delivers decrypted multicast packets to any process listening on the same interface and port as the registered multicast application, even if the process has not joined the multicast group. This happens when the multicast application sets the socket **SO_REUSEADDR** option before binding to the port. This is routinely done in application code. Since multicast only works with UDP sockets, care must be taken to control access to the port. This is an instance of the access control granularity problem discussed in section 7.1.

  A temporary work–around is to have the secure multicast group $join$ operation not set the **SO_REUSEADDR** socket option. This way after the first group member joins, no other applications will be able to bind to the multicast group's UDP port. This limits the granularity of multicast group membership to one application per host. Better solutions which do not have this restriction are described in section 7.1 but have not yet been implemented.

- **Packets in the clear** If a plain UDP packet is sent to the multicast address, it should be dropped by the receiver. The FreeSwan–0.91 code does not drop the packet. This is a known bug to the FreeSwan developers and it is mentioned in the FreeSwan distribution.

  In general, an implementation of IPSec must check and drop any received packet that is not properly protected [CGHK98].

- **Conclusion from Feasibility Test Results** Freeswan 0.91 was cumbersome to install, configure and test. Documentation was basic but adequate. So far, tests of secure IP multicast with our patched Freeswan–0.91 seem to work within the limits of the previous remarks.

8.2 Performance Tests

We performed a series of tests to evaluate the performance cost of adding security to group multicast in our architecture. Since only the data flow side of our design is implemented, we focussed on the cost of adding group authentication, source authentication and encryption to the multicast data flows within our architecture. We did not address performance issues relating to key management. We performed our tests on a 400 Mhz Pentium II machine running Red Hat Linux 5.1 with kernel version 2.0.35 compiled with the Freeswan–0.91 package [FSWAN] IPSec. Our SAM user library was implemented as described in Section 6 using the Hybrid Signature scheme for source authentication as described in [R99]. The test machines were connected via a 10Mbps ethernet LAN. In the absence of a MIKE module, all keys used for these tests were entered manually.

8.3 Description of Tests

- *Baseline Test: Multicast over UDP*: In order to establish a performance baseline, we first tested the speed of plain multicast over UDP, i.e., the packet processing time required to send a UDP packet down to the communication hardware. This was done for two different packet lengths both of which are less than the MTU size.

- *UDP–Multicast over IPSEC*: In this test, we measured packet processing time for UDP–Multicast packets over IPSEC for two packet sizes, using the ESP protocol with the Triple DES and HMAC–MD5–96 transforms.

- *SAM over UDP–Multicast over IPSEC*: In this test we measured the packet processing time for packets sent through our SAM interface to the UDP–Multicast/IPSEC layer. The IPSEC layer used the ESP protocol with the Triple DES and HMAC–MD5–96 transforms and the SAM layer used the Hybrid Signature scheme as described in [R99]. This was again done for two different packet sizes.

8.4 Performance Results

We first tried to measure the time to send a packet by measuring the time taken by an application to send several packets and dividing by the number of packets. This approach produced some strange results: we observed that an application executing a loop to repeatedly send a multicast packet over UDP could take longer than a application trying to send the same number of multicast UDP packets over IPSEC! We were able to trace this anomaly to the networking device's buffer getting filled up by the high data input rate when plain multicast

UDP Payload Size	Ethernet Frame Size	Time (μs)
464	506	15.5
1262	1304	19.4

Table 1. Baseline Test Results

UDP Payload Size	Ethernet Frame Size	Time (μs)
464	542	349
1262	1334	823

Table 2. Multicast over UDP/IPSEC

packets were sent. This would cause a large OS overhead since the process would have to be swapped out and placed in the wait queue and restarted only when networking device buffer was less than half–full. This overhead could significantly skew the average time calculation. For example, when a loop to send out 1262 byte messages over UDP was executed, the first 41 iterations of the loop took 796 μs, or roughly 19.4 μs per packet but the first 42 iterations took 26468 μs or an average of 630 μs per packet! So we decided to measure only the best timing we obtained when sending few packets, which in this case would be 19.4 μs per packet.

- **Baseline Test: Multicast over UDP**
  Table 1 summarizes the results.

- **Multicast over UDP/IPSEC**
  Table 2 summarizes the results.

- **SAM/UDP/IPSEC**

  When measuring the time to send only a few packets (100 in this case), with the SAM module using the hybrid signature scheme of [R99]i, we observed that the packet processing overhead was not substantially worse than in the case without source authentication. This is because of the off–line/on–inline nature of the hybrid signature scheme. When sending only a small number of packets, the average time calculation includes only the on–line costs of the hybrid scheme. The results are summarized in Table 3 below.

  However, when averages were computed by sending much larger number of packets (1000 in this case), the average calculations included a portion of the off–line cost as shown in Table 4

  Eventually, we expect that the full off–line cost of around 1500 μs per packet would show up when calculating the averages based on sending a very large number of packets.

UDP Payload Size	Ethernet Frame Size	Time (μs)
464	542	392
1262	1334	933

Table 3. Avg per–packet time (100 pkts)

UDP Payload Size	Ethernet Frame Size	Time (μs)
464	542	576
1262	1334	1522

Table 4. Avg per–packet time (1000 pkts)

9 Conclusion

We have presented a host architecture for a member in a secure multicast group. The architecture is based on the IPSec host architecture for secure point-to-point communication, and re-uses the IPSec components (ESP, AH, IKE). In addition, the architecture identifies new modules: MIKE for key exchange, SAM for source and data authentication, and the MSA for bridging the control path (MIKE) and the data path (ESP/AH and SAM). We have discussed compatibility issues with IPSec, and described an on-going effort to validate the architecture via implementation.

The proposed architecture complements existing proposals for secure multicast key and policy management, that concentrate on the design of the group control entities. We hope it will become an integral part of a comprehensive secure multicast solution.

References

[BMS99] D. Balenson, D. McGrew, A. Sherman, “Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization”, Internet Draft draft-balenson-groupkeymgmt-oft-00.txt, February 1999.

[CG$^+$99] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas, “Multicast Security: A Taxonomy and Efficient Authentication”, INFOCOM ’99.

[CP99] R. Canetti, B. Pinkas, “A taxonomy of multicast security issues”, Internet Draft draft-irtf-smug-taxonomy-01.txt, April 1999.

[CE$^+$99] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, D. Saha, “A Toolkit for Secure Internet Multicast”, INFOCOM 99.

[CGHK98] P. C. Cheng, J. A. Garay, A. Herzberg, H. Krawczyk, “A Security Architecture for the Internet Protocol”, IBM System Journal, Vol. 37, No. 1, Feb. 1998.

[D91] Steve E. Deering, “Multicast Routing in Datagram Internetworks”, Ph.D. Thesis, Stanford University, December 1991.

[RFC1112] S. Deering, “Host Extensions for IP Multicasting”, IETF Request for Comments No. 1112, August 1989.

[FSWAN] Code available at http://www.xs4all.nl/ freeswan/download.html.

[HCD98] T. Hardjono, B. Cain, N. Doraswamy, “A Framework for Group Key Management for Multicast Security”, Internet Draft draft-ietf-ipsec-gkmframework-00.txt, July 1998.

[HCM98] T. Hardjono, B. Cain, N. Monga, “Intra-Domain Group Key Management Protocol”, Internet Draft, draft-ietf-ipsec-intragkm-00.txt, November 1998.

[HM99] T. Hardjono, N. Monga, “Group Security Association (GSA) Definition for IP Multicast”, Internet Draft draft-irtf-smug-gsadef-00.txt, February 1999.

[HM97a] H. Harney, C. Muckenhirn, “Group Key Management Protocol (GKMP) Specification”. IETF Request for Comments 2093, July 1997.

[HM97b] H. Harney, C. Muckenhirn, “Group Key Management Protocol (GKMP) Architecture”, IETF Request for Comments 2094, July 1997.

[H95] C. Huitema, “Routing in the Internet”, Prentice Hall, 1995.

[KA98] Stephen Kent, Randall Atkinson, “Security Architecture for the Internet Protocol”, IETF Request for Comments 2401, 1998.

[KBC97] H. Krawczyk, M. Bellare, R. Canetti, “HMAC: Keyed-Hashing for Message Authentication”, IETF Request for Comments 2104, February 1997.

[LDAP] ”Lightweight Directory Access Protocol (v3)”, IETF Request for Comments 2251, 1997.

[M99] L. Mccarthy, “RTP Profile for Source Authentication and Non-Repudiation”, Internet Draft raft-mccarthy-smug-rtp-profile-src-auth-00.txt, May 1999.

[M97] S. Mittra, “Iolus: A Framework for Scalable Secure Multicast”. In Proceedings of ACM SIGCOMM ’97, Cannes, France, September 1997.

[Q98] Bob Quinn, ”IP Multicast Applications: Challenges and Solutions”, draft-quinn-multicast-apps-00.txt, Nov 1998.

[RM] The Reliable Multicast working group at the Internet Research Task Force, http://www.irtf.org/charters/reliable-multicast.htm.

[R99] Pankaj Rohatgi, ”A Compact and Fast Hybrid Signature Scheme for Multicast Packet Authentication”, To appear in the Proceedings of 6th ACM Computer and Communications Security Conference, 1999.

[SMuG] The Secure Multicast working group at the Internet Research Task Force, http://www.irtf.org/charters/secure-multicast.htm and http://www.ipmulticast.com/community/smug.

[SSLRef] SSLRef 3.0: Available from http://www.netscape.com.

[STW98] M. Steiner, G. Tsudik, M. Waidner, “CLIQUES: A new approach to group key agreement”, IEEE ICDCS’98, May 1998.

[WHA97] D.M. Wallner, E. J. Harder, R. C. Agee, “Key Management for Multicast: Issues and Architectures”, Internet Draft draft-wallner-key-arch-01.txt, September 1998. (Preliminary version in July 97.)

[WGL98] C. K. Wong, M. Gouda, S. S. Lam, “Secure Group Communication Using Key Graphs”, SIGCOMM ’98. Also, University of Texas at Austin, Computer Science Technical report TR 97-23.

[WL98] C.K. Wong, S.S. Lam, “Digital Signatures for Flows and Multicasts”, IEEE ICNP ’98. See also University of Texas at Austin, Computer Science Technical report TR 98-15.