The University of Kansas

Technical Report

A Prototype Implementation for Dynamically Configuring Node-Node Security Associations using a Keying Server and the Internet Key Exchange

Suresh Krishnaswamy,
Joseph B. Evans, and Gary J. Minden

February 2003

ITTC-FY2003-19740-08

Project Sponsor:

The Defense Advanced Research Project Agency
and the United States Air Force Research Laboratory
under contract F30602-99-2-0516

Abstract

Realizing large-scale active networks is heavily contingent upon addressing security concerns at the outset. Various approaches have been taken toward integrating security within an active node, each defining the mechanisms required to be in place within the NodeOS or the EE in order to provide security guarantees within the system. An acceptable short-term solution to security in deploying a practical testbed such as the ABONE is to divide security concerns into two classes viz. hop-by-hop and end-to-end. This paper describes one approach toward setting up hop-by-hop packet authentication and integrity, similar to the ABone Hop-by-Hop message authentication and integrity framework, but usable in a more general context. It answers most of these requirements using existing protocols in network security and is flexible enough to be used in any scenario requiring mediated node-node security associations.

List of Figures

1 Introduction

Traditional networks are constrained in the way they process packets. There exists a strict distinction between what the user does in terms of executing the code and what the underlying network provides the user. Processing within the network is limited to simple forwarding and/or routing decisions with some level of congestion control and Quality of Service. The traditional "passive" networks have several limitations - the difficulty in integrating new technologies and standards into the shared network infrastructure, poor performance due to redundant operations at several protocol layers, and difficulty in accommodating new services within the existing architectural model. Active Networks were born out of the need felt to replace the numerous ad hoc approaches to network-based computation, thus providing a generic capability that allowed the users to program the network itself. It is now possible to think of new protocols and innovative cost-effective technologies easily deployed at intermediate nodes.

Simply having any active packet modify the behavior of the intermediate network-nodes has potential implications on security. An active packet containing incorrect code, or more importantly, malicious code, can easily compromise any active node if proper precautions are not taken. The topic of security in active networks is complex in itself and like every big problem, this one too is better approached, by breaking it into parts. In this paper we attempt to answer security issues between two communicating peers, using a keying server and predefined protocols in the area of network security.

1.1 Introducing the security problem in Active Networks

Given the flexibility and power that active packets provide, it is only obvious that security concerns in active networks are amplified many-fold. Numerous threats arise due to the various interactions between the NodeOS, the EE, the channels and the ANEP[1] packets. A threat model (Figure 1) can be defined by identifying the various cases where security may be compromised.

The principals that need to be considered in this threat model are the active packet, the NodeOS, the EE(s) and any installed active service. Channels may be created either by the EE or by the installed active code by specifying the filtering identifier to the NodeOS.

[1] Active Network Encapsulation Protocol

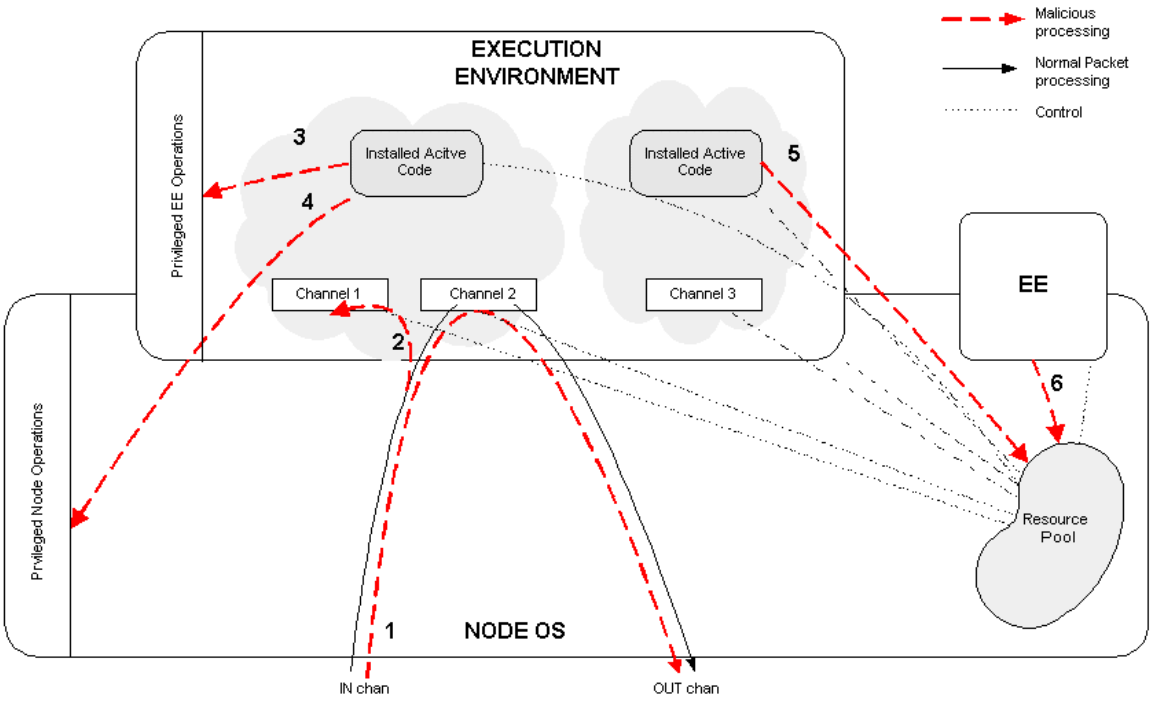


Figure 1Threat model in Active Networks

We begin by considering a bogus packet entering the active node (1). An active node can be flooded by such bogus packets with the intent of causing some Denial of Service. The EE sees a threat from this bogus packet because processing such packets translates into some direct resource consumption by itself. EEs are granted only limited resources when they start up and so care must be taken to see that its resources are not consumed indiscriminately. The same applies when these bogus packets consume resources allocated to other installed active services within the Execution Environment.

Malicious active code installed in an Active Node may try and access (or modify) certain privileged information within either the NodeOS or the EE (3, 4). Privileged functionality can either be some protected operation, or just any sensitive information that needs to be kept private within the node. Additionally, active code installed within the EE may try and steal packets from a flow (2) by setting up filters that manage to capture someone else's packets. This means that certain characteristics required by the sender of the packet can now be manipulated to exhibit different behavior. Active code may also directly try and access resources allocated to other active services or EEs (5), causing the latter to exceed their resource bounds and be terminated or revoked.

A malicious EE can try and consume all of the node resources or feed off resources allocated to other EEs (6). There is no way for active code to ensure by itself that the EE in which it is executing will perform its operation reliably. A faulty or malicious EE can leak private information contained within the packet and do practically anything it wants with the active code. In the same way, a malicious active node may do anything it wants with an installed EE or active code.

The Security Architecture for Active Networks [8] uses the above threat model as a reference to describe the security architecture components of an active node. It also describes the recommended primitives available to an active application for interacting with the security architecture. This architecture aims primarily at answering the following questions:

1. Which principal wants to perform this action?
2. Does this action have any security implications?
3. Is this principal authorized to perform this action?

The first question relates to authentication. Knowing who wants to perform the said action is a necessary precondition to any form of secured policing. Authorization is very often combined with the list of authorized activities a principal can perform to form what is known as a credential. Since attempting to describe the security policy in terms of each individual principal's authority to access each individual object is not considered scalable, credentials combine the description of the identity of the principal and the aggregation of certain attributes associated with that principal using roles, groups, etc.

Questions 2 and 3 relate to node policy. Policy enforcement cannot be built directly into the NodeOS or the EE enforcement mechanism since it is not possible to know a priori who will or will not be granted access to an active code. Furthermore, it is not possible to know what actions the active code would consider security relevant. Thus every node, EE and active service exports a set of interfaces to operations and resources within itself. Access to these resources can now be mediated by treating access to these interfaces as specific events within the policy framework.

Compliance checks are made at appropriate places in order to ensure that a particular operation abides by the policy. Authorization may also be delegated, at which time credentials describing the originating principal may also need to be passed on. In some active network architectures, there may be no mechanism to pass credentials identifying the requesting principal to the node. Here the node must allocate bulk resources and trust the EE to perform reliably. In case the EE is not that trusted, the node may simply limit its risk by allocating only limited resources to the EE.

1.2 What problem we are attempting to solve

In order to form a realistic testbed for Active Networking experiments, the Active Network Backbone, or the ABone [1] has been set up. The Abone forms a virtual network infrastructure on which a growing set of active network components can be tested and experimentally deployed. While security in active networks is an ongoing effort in the research community, there still remains the need to provide some acceptable short term solution so that nodes can be reasonably trusted for the near term usage of the ABone.

Secure services for active networks can be roughly separable into two classes viz. hop-by-hop and end-to-end. Hop-by-hop protection is intended to prevent "outsiders", or non-neighbors from inserting packets into the stream. This in turn ensures that any signaling between the neighbors is also protected. End to end security mechanisms on the other hand, bind the originating entity to the packet so that the necessary access control checks can be performed.

Hop-by-hop security is general enough to warrant its presence in the NodeOS itself – different EEs can then use this as a common service. It still remains unclear whether end-to-end security should reside in the EE or have part of it implemented in the NodeOS. Currently, the ABone effort is directed toward placing an end-to-end security mechanism into each EE and requires that the NodeOS trust the EEs in this regard.

This paper describes our effort toward setting up hop-by-hop packet authentication and integrity as specified in the ABone Network Security Architecture [2], but useable in a more general context. It answers most of the requirements of hop-by-hop packet security by using existing protocols in network layer security as opposed to the ANEP layer as described in [6]. This approach is general enough to be used in any networking scenario requiring mediated node-node security associations. Our approach uses a keying server to dynamically set up Security Associations between two peers that want to have hop-by-hop security between them. A secure topology can be defined at the keying server using a simple configuration file. Hosts that are a part of this secure topology, simply establish an IKE Security Association with the keying server, and are handed all the pertinent keying information in the form of encrypted IKE notification payloads.

Security Associations between peers are implemented in exactly the same way as is done for IPSec and hence all the features of IPSec such as replay protection, message authentication and message integrity are automatically adopted by this framework. This framework also integrates multicast security associations into the IPSec framework and implements the hierarchical key distribution mechanism, facilitating easy revocation of group members.

2 Background

2.1 Hop-by-Hop Security at what level?

Data sent between the two nodes can be secured at various layers of the OSI model. The layer where security actually needs to be implemented depends on the user and the more importantly the application requirements.

In the case of hop-by-hop security, performing secure transforms at higher layers might lead to duplication of effort either on a per-application or a per-transport basis. It thus makes sense to have this feature lower down in the active network protocol stack, either in the ANEP or the network layer. We use the network level rather than the ANEP layer in order to provide hop-by-hop security services. Doing so allows us to use this framework even in non-active scenarios. We summarize a few differences between our approach and the ABONE hop-by-hop framework in Table 1.

Defining security services in the network layer also allows us certain component choices:

- Network layer entities such as IP addresses can be easily authenticated using the DNSSEC framework
- Packet authentication and integrity can be achieved using the IP Security Framework (IPSec) as defined by the IETF.

The Keying Server Prototype	ABONE Hop-by-Hop framework
Defines Security Associations in the Network Layer	Defines Security Associations in the ANEP layer
Identifies the SA using the receiver IP address and the SPI	Identifies the SA using the sending interface and the key identifier
Uses existing sequence numbering and replay protection within the IPSec implementation	Needs to define this separately since replay protection is done at the ANEP layer
Do not have a clear picture of the active packet's message boundaries	Clearly knows the message boundaries, hence can perform services such as non-repudiation more effectively

Need to secure every packet sent from the source to the destination irrespective of whether it is an ANEP packet or no	Can be more efficient in terms of speed because ANEP packets as opposed to individual IP packets are being secured
Defines a key management protocol to distribute authentication keys among peering nodes	No such protocol has been defined in this framework as yet although key management interfaces have been defined

Table 1Comparison between the Keying Server and the ABONE Hop-by-Hop security frameworks

2.2 The IP Security framework

IP packets themselves have no inherent security. They only provide us with a simple level of safety where malformed packets and packets not destined to any particular host are discarded or dropped. There is no means to prevent different hosts from reading packets passing through the network, from modifying IP data and their related checksums, and from spoofing packets in general. IPSec provides us the mechanism to protect IP datagrams by defining a set of protocols to be used along with IP, thus providing features such as source authentication, data integrity, data confidentiality, and anti-replay protection.

IPSec provides a standard, robust, and extensible mechanism to provide security to IP and upper-layer protocols. A default suite of algorithms is defined to assure interoperability between different implementations while also keeping it relatively straightforward to add new algorithms without breaking interoperability. IPSec achieves its security using two protocols - Authentication Header (AH) and Encapsulating Security Payload (ESP). A "Security Association" (SA) is a term used to refer the particular set of security services afforded to IP packets sent from a particular source to a particular destination. Packets are mapped to a particular SA based on some security policy.

IPSec provides all of its security services using secret key cryptography. A mechanism to manually add keys for these services is mandatory to implement thus ensuring interoperability of the base IPSec protocols. Since this method of manual keys scales very poorly and is difficult to manage, a dynamic key management protocol called the Internet Key Exchange (IKE) is also defined.

2.2.1 The Internet Key Exchange

RFC 2409 describes the IKE framework for obtaining authenticated keying material for use in Security Associations such as AH and ESP. The IKE protocol is a hybrid of three different protocols, each contributing to different features:

- The Internet Security Association Key Management Protocol (ISAKMP) defines a framework for authentication and key exchange but does not define them specifically.
- The Oakley protocol describes a series of key exchanges and details the services provided by each
- SKEME describes a versatile key exchange technique, which provides anonymity, repudiability and quick key refresh.

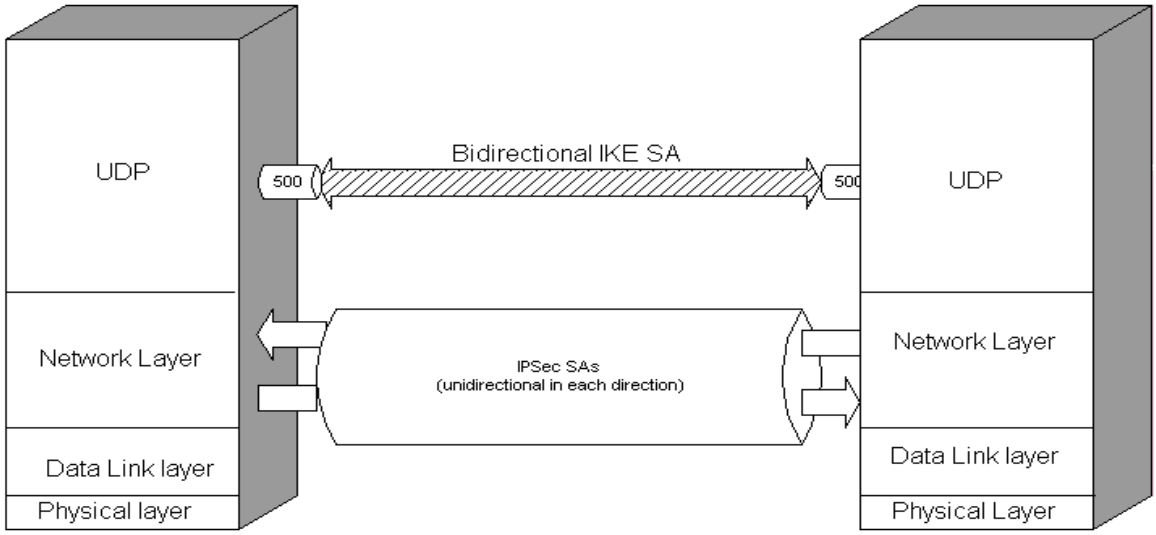


Figure 2 IKE and IPSec SAs

Using a part of each of these component protocols, IKE is able to provide a rich variety of services within the IPSec Domain of Interpretation (DOI) as well as general-purpose policy and key negotiations for other protocols such as SNMPv3, OSPFv2, etc.

1.

The IKE protocol also shares the concept of a Security Association. However in the IKE context, the SAs are bi-directional. Nodes wanting to be IPSec peers must also be IKE peers. The IKE SA defines the way in which the two peers communicate – the

algorithm to use to encrypt the traffic and the means to authenticate the other end. The IKE SA once formed can then be used to create any number of IPSEC SAs between the peers (Figure 2).

2.3 DNSSEC Authentication

The DNSSEC framework (RFC 2535) describes the security extensions to the basic Domain Name System. These extensions provide enhancements to Key Distribution, Data Origin Authentication and Transaction and Request Authentication that allow it to be used as infrastructure for public key distribution.

The DNS Security Extensions define a new Resource Record (RR) type called the KEY record that can be used to associate public keys with DNS names. Authentication of records is possible by a security aware resolver by checking the digital signatures sent as SIG records as part of a query response. Commonly all the zone records in the DNS server are signed at the same time using a single private key but there can also be multiple keys for different algorithms, signers, etc. The private key for the DNS zone can be kept offline for most of the time and is not in any way related to the server that maintains the DNS records. Retrieving signed keys for a zone's sub-zones can be done by descending through the tree starting with one or more trusted keys for the parent zone.

2.4 Multicast Key Distribution

Figure 3 shows the Logical Key Distribution architecture [11], which provides for secure removal of registered members providing transmission and storage efficiency.

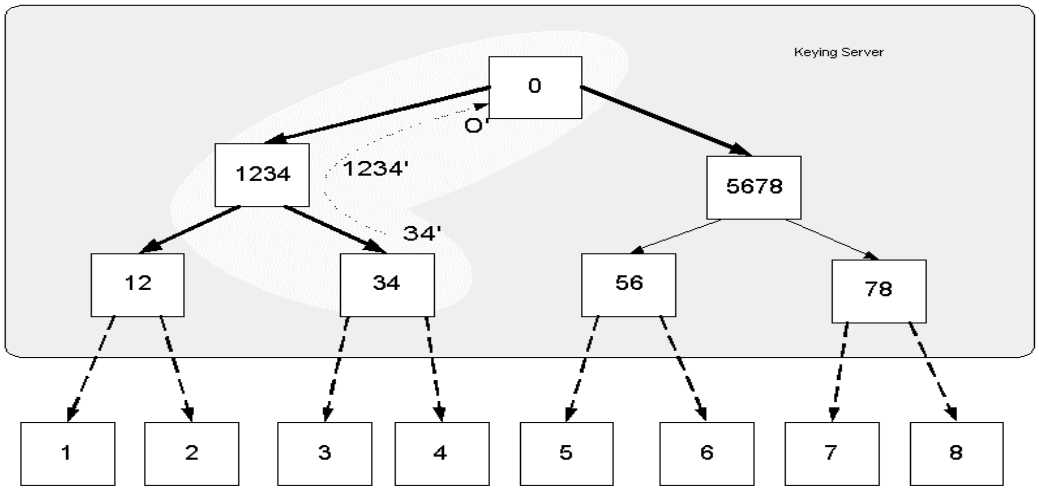


Figure 3 The Logical Key Distribution Framework

Although this scheme is hierarchical, there is no actual requirement for multiple servers. Nodes wanting to be a part of the multicast group establish a shared secret with the server. The server additionally generates a number of intermediate keys for each remaining node in the tree using some robust key generation process. Clients remain ignorant of these intermediate keys generated by the server. Thus in the sample hierarchical tree shown in the Figure 2.5, keys 1 through 8 are keys shared by the server independently with the eight different nodes, while keys 12, 34, 56, 78, 1234, 5678, and 0 are maintained as internal nodes in the keying server's hierarchical tree.

Starting above the leaf node of which a client is a member and proceeding toward the root, the server sends all ancestor node-keys within the tree, such that every key sent is encrypted by its corresponding child key. Thus for the node 3, the server sends the key 34 encrypted using key 3, key 1234 encrypted using key 34 and key 0 encrypted using 1234

Key messages may also be combined into one message at the expense of larger message size. Thus keys 0, 34, and 1234 may be encrypted once using key 3 and then sent to the client corresponding to this particular key. At the end of this process, all the nodes that have registered with the server now have the list of keys from the root to its leaf in the hierarchical tree. Key 0, which forms the root of the hierarchical tree is shared between all the clients and is used as the common multicast key.

To see the strength of the LKH approach let us consider that client 3 needs to be revoked form the multicast group. In order to do so, the keying server simply changes the ancestor keys corresponding to client-3 i.e. keys 0, 34, and 1234. Only nodes that use one of these affected keys need to be made aware of this change. The keying server thus sends out only the following key-update messages (shown lightly shaded in Figure 3): Key-34' encrypted using Key-4, Key-1234' encrypted using Key-34', Key-1234' encrypted using Key-12, Key-0' encrypted using Key-1234' and Key-0' encrypted using Key-5678. Key-0' now forms the new group key.

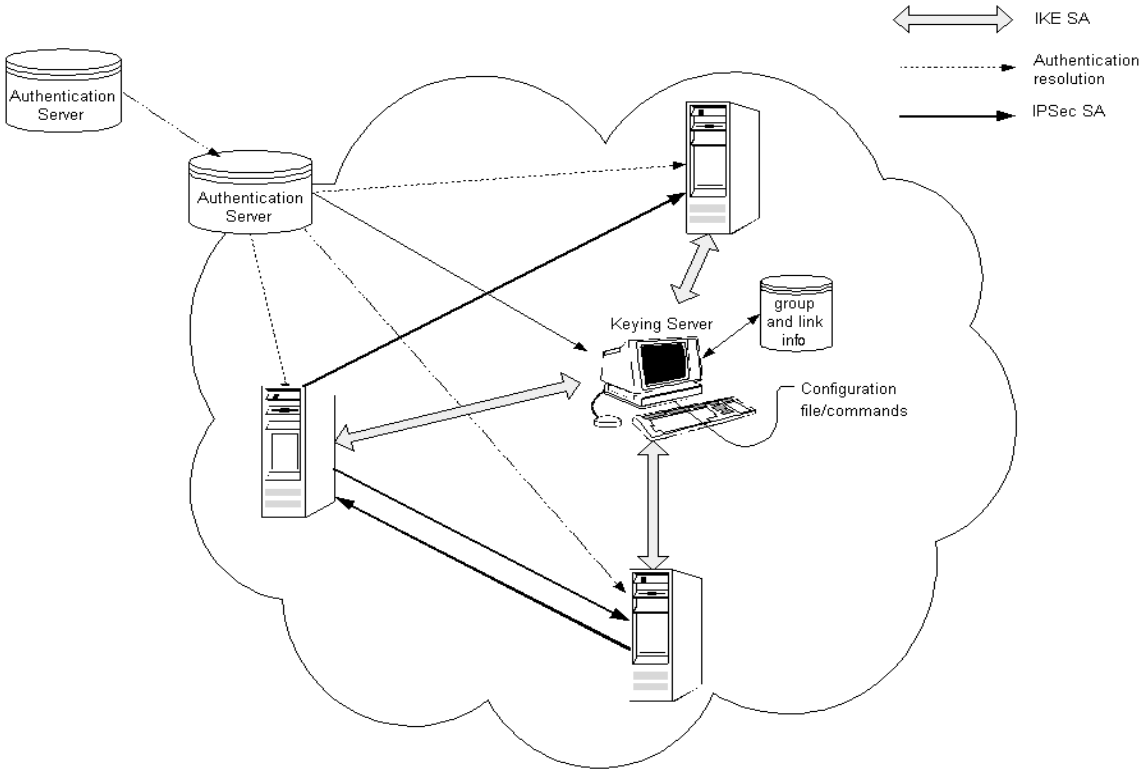
The LKH approach has a number of benefits. Firstly, the costs of user-storage and re-key transmissions are balanced and scale well as the number of clients increase. Also, every intermediate key is capable of being used as a subgroup key protecting those nodes below it. Finally, this approach is also resistant to collusion among a group of users.

3 Designing the Keying framework

3.1 Overview

Our keying infrastructure consists of three main components as shown in Figure 4 - the Keying Server (KSV), a Key Management Module (KMM) within every node, and an Authentication Server (ASV). The keying server forms the central entity in our framework. Rather than defining a separate module for the KMM we decide to extend the services provided by IKE itself so as to seamlessly integrate IPSec services into our framework. Finally, the ASV itself can be simply treated as a DNSSEC server.

Figure 4 Design overview for the keying framework



The secure topology is specified in the form of links and groups, where the links correspond to unidirectional Security Associations and the groups, the different

multicast groups. This topology is configured into the KSV either statically, when the server comes up or dynamically using configuration commands. We define the "trusted set" in a KSV as the collection of all those nodes that are either configured as one end of a secure link or a member of a secure group inside a particular KSV.

The KMM runs on every node that wants to be part of the KSV's trusted set including the KSV itself. In order to set up the relevant security associations, a node needs to register itself with the KSV. Either the KSV or the trusted node can initiate this registering process. If initiated by the node, we can think of this being done automatically when the node boots up. A node can register with multiple servers; we do not however define how to resolve any conflicts that may arise out of the same link being defined at two different servers. Errors in configuration are flagged to the appropriate servers when a node detects that such a mismatch in configuration has occurred. Policy can additionally be used to define which KSV a node is allowed to register with in the first place.

During the node registration process and any time the configuration at the KSV affecting this particular node has changed, the KMM at the KSV sends all the pertinent information for both links as well as groups to this node. This link and group information is then used to set up the security associations within the node.

Information exchanged between the KSV and the trusted nodes are authenticated so as to discourage spoofing of messages, sent either with the intent of breaking security or simply in order to perform some Denial of Service attack at the KSV. The Authentication Server provides this service in our framework. The ASV maintains authentication information for all those nodes that it considers as being a part of its domain of control. ASVs are organized hierarchically and perform some trust-chaining mechanism to authenticate nodes not lying directly in its domain of control.

In the following sections we describe the basic design of the various sub-components, the capabilities if any, that we need additionally from them, and finally how we actually provide these capabilities.

3.2 Integrating IKE

3.2.1 Node Registration

Nodes that want to be a part of the secure topology maintained at the KSV need to register with it. Once an IKE SA between the KSV and the node has been set up,

keying information is sent from the former to the latter. Since the registering node needs to set up IPSec-SAs without any direct IKE negotiation with the peer, the KSV needs to send all such information required by the registering node in order to describe the connection and SA between the latter and its peer accurately. We investigate the different fields that need to be included in the message sent by the KSV in order to achieve this.

In order to set up a Security Association, the first thing that needs to be known is the type of security service (transformation type) – AH or ESP being offered by this SA. We specify a list of policy groups in the KSV configuration file. Every link or group specified in the configuration file is associated with a particular policy. Each policy group contains the following information:

- The transformation type : e.g. (TRANS-TYPE = AH or ESP)
- The lifetime in seconds for this SA: e.g. LIFE-SEC = 100
- The lifetime in Kbytes for this SA: e.g. LIFE-KBYTE = 10000

Another value that we need to add artificially is the SPI value. The SPI is used to identify a particular SA at the receiving end. Since this value is receiver specific, the KSV creates an SPI for every node that it maintains as part of its trusted set. By ensuring that the SPI generation process does not create duplicate values, the KSV can make sure that no two hosts use the same SPI to send data to the same peers. In order to conserve SPI space, the <SPI, sender> tuple can be used instead of just the SPI to identify the SA uniquely at the receiver. However in our application, we use the SPI value directly.

Security Associations can be either inbound or outbound. Each of these SAs can exist independent of each other and can also have separate policies. Links at the KSV are specified with some directionality. In our configuration file we specify outbound links using the symbol " =>". In order to define inbound and outbound links with symmetrical policy, we also specify a bi-directional link using the "⇔" symbol. Examples of configuring links with such directionality are given in section 4.

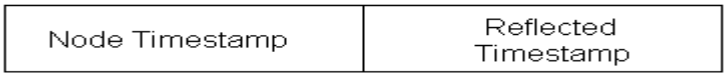


Figure 5 Fields in the ACK Message

The KSV can specify some additional operations to be performed as soon as the given set of SAs is installed at the receiver. An indication of which operation is to be performed is passed as flags in the register message.

1. REFRESH: This set of SAs is treated as a complete set and not an incremental update. Any older SA that was created as part of an earlier registration or update process has to be deleted.

2. ACK_REQD: The KSV has asked the node to send back a confirmation about having received and installed the SAs. Since IKE messages are sent using the unreliable UDP protocol, this is particularly useful in some instances where we want to guarantee that a particular SA was successfully installed. A timestamp returned by the node can be used by the KSV to determine the order in which SAs were installed at the node if required. In order to identify which out of a number of possible states an ACK value corresponds to, the client also reflects back the server's timestamp value in its ACK message.

The format of the ACK message is specified in Figure 5. We summarize the information contained in a registration message in Figure 6 below. A few of the fields defined in Figure 6 are used in the context of multicast groups and are explained in section 3.6. Nodes that do not belong to any multicast group have the "ID len" and "Number of Keys" fields set to zero.

Once the information necessary to form an SA is received by the node, it goes ahead and instantiates the same. No negotiation between the two nodes or costly DNS lookups is required because KSV has already done this during the registration process.

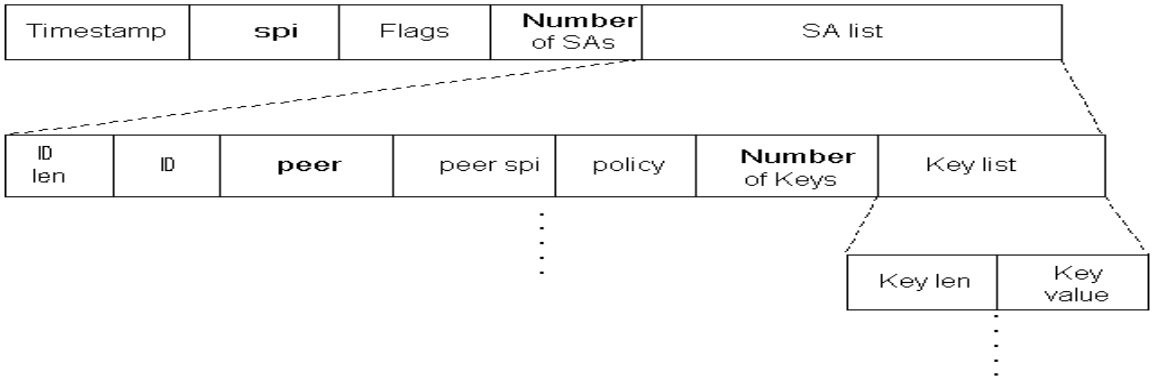


Figure 6 Fields in the Registration Message

3.2.2 SA Deletion

The process of deleting an SA is very similar to that of creating one using the keying server. However, we only need to send enough information to identify the SA or more

specifically, the peer and the SA directionality. Figure 7 specifies the format of fields required in the delete message. We also specify the flag and timestamps fields as specified in the previous section

Figure 7 Fields in the Delete Message

When a node receives a "DELETE" message from the keying server for a particular SA, it identifies and deletes all the states associated with the same. Deletion of these states automatically cleans up any connection that may have been created "on the fly".

3.2.3 Alarm Indication

Messages that are sent by the Keying Server do not fail silently if they are not able to perform their expected behavior. The affected node sends back some error indication, which the server can simply log or use directly in order to rectify the problem. We identify the following types for alarm indications:

- SA_EXPIRED
- MALFORMED_PACKET
- SERVER_CONFLICT
- SIGN_FAILED

Figure 8 Fields in the Alarm Message

Security associations have a limited lifetime. We need to re-key a particular SA before the latest one expires so that the link security is never compromised. In case a new key is not generated before the latest one expires, the node continues to use the existing SA, renewing its lifetime to the default value. The alarm "SA_EXPIRED" is however sent to the keying server to indicate that such an event has taken place.

A "MALFORMED_PACKET" indicates that the node did not understand some packet sent by the server. This is different from the malformed packet at the network layer, which we handle by retransmission mechanisms at the TCP level or drop entirely in the case of UDP. Reception of such an alarm at the KSV is usually an indication of either someone attempting (unsuccessfully) to spoof or replay messages

from the server or simply a loss of synchronization between the keying server and this node.

Some configuration errors such as two keying servers independently specifying the same node to be a part of the same multicast group can be detected at the node. If so, the node sends the "SERVER_CONFLICT" message to both the servers it has registered with.

The SIGN_FAILED alarm is used in the context of secure multicast groups, and is described in section 3.6.

3.3 Information Packaging

IKE uses the ISAKMP protocol (RFC 2408) to specify the message formats sent between the two peers during various exchanges. Messages exchanged in an ISAKMP-based key management protocol are constructed by chaining together ISKMP payloads to an ISAKMP header. There are thirteen distinct payloads that all begin with the same generic header. Payloads are chained together in a message using the "next payload" field in the generic header. The ISAKMP header describes the first payload following the header and each payload describes which payload comes next. In our framework, we use the ISAKMP notification payload in an "Informational Exchange" to send keying information from the KSV to the node.

We define the following notification message types for use within the keying server framework with the values for these types taken from the private section of the status message codes:

- KEYEXCHANGE_ACK = 32768
- KEYEXCHANGE_REGISTER = 32769
- KEYEXCHANGE_DELETE = 32770
- KEYEXCHANGE_ALARM = 32771

The values within the notification payload are as specified in RFC 2408 except for the notification data field, which is decided by notification message type of this payload. For the notification types defined in our system - KEYEXCHANGE_ACK through KEYEXCHANGE_ALARM we define the notification data exactly as shown in Figure 5 through Figure 8.

The Informational Exchange messages are protected using the IKE SA that has already been created. Thus only the IKE peers are able to read or modify the notification message contents.

3.4 Installing IPSec Security Associations

The Freeswan KerneL IPsec Support (KLIPS) implementation provides us with a mechanism for installing and maintaining Security Associations within a node.

KLIPS is implemented as follows. Every physical network interface is associated with one virtual interface (ipsec0, ipsec1, etc). Whenever a new SA is to be installed either manually or using the pluto IKE daemon, a new route referencing this virtual interface is installed in the kernel routing table.

The SA corresponding this connection is itself added to the SADB, which is maintained as a hash table in kernel memory. IP packets belonging to a particular SA now pass through the ipsec virtual interface and are applied IPSec transforms based on the security association retrieved from the SADB.

In our prototype, we use the KLIPS implementation for unicast SAs, as is.

3.5 Extending the IPSec security associations for multicast data

IPSec is inherently a point-to-point protocol. One side encrypts and the other side decrypts using some shared key either statically configured into it or dynamically generated using IKE. Securing multicast data is a totally different paradigm since there are multiple recipients of a single packet and often, many senders to the same multicast address.

Some aspects of IKE fail when viewed in a multicast context. It is not possible for SPI values to be unique at all destinations within a multicast group without making the negotiation process too complex. Even if this were possible, we now have the problem of the entire SPI space being shared by all multicast nodes rather than each of them maintaining their own individual SPI space. A better approach and the one that matches our prototype well, is for the multicast server (the KSV in our prototype) to define and distribute the SPIs for each multicast group itself.

Another aspect that fails in a multicast scenario is that of source authentication and replay protection. There is no way to synchronize the anti-replay counters when all nodes sharing the same secret can send data on the same multicast address. In our implementation we simply turn off replay protection and limit source authentication to lighter forms of “one within the group” rather that stricter definitions.

In order to have multicast support within KLIPS, we need to make a few modifications.

- KLIPS is modified to treat any multicast packet as destined to itself. Multicast packets are only dropped if no application has been registered on this node to read multicast packets sent on this address.
- As of this writing KLIPS is incapable of handling IP packets with a header size greater than 24 bytes as in the case of IGMP. We modify this behavior to allow the latter to go through
- We also note that it is not possible to have an inbound and outbound SA for a given multicast address. This is because for both cases, the destination address and the SPI remain the same. We do not know the sender beforehand and so cannot remove the SA ambiguity using the source address. Hence we install only an outbound SA. The incoming packets use this same SA in order to decrypt packets sent to this multicast address

In order to allow any side to be able to find the SA for a multicast packet uniquely, each node now uses the common group SPI as both the sending as well as the receiving SPI. Multicast SAs defined in this manner are always bi-directional which, intuitively, seems perfectly reasonable.

3.6 Integrating the hierarchical keying framework for multicast groups

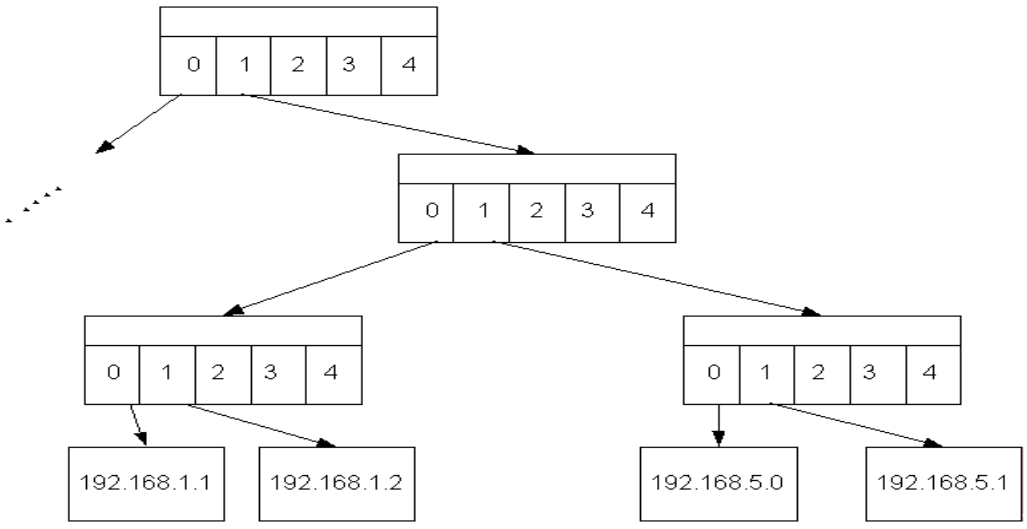


Figure 9 Generating Group Member IDs

We use the LKH approach described in section 2.4 in our prototype implementation for multicast group member revocation.

The LKH tree is constructed as a B+ tree. Nodes are added by comparing an "index" with the "current" node-index and traversing the tree based on the comparison result. The index would be a value that uniquely defines the host in a particular trusted set.

The KSV defines a multicast group, which it uses to send key update messages to all members defined within one of its groups. This address is either known beforehand to each of the clients or is sent as part of the first register message sent by the KSV. In our implementation we have this value well known by all the clients. Clients that lie in the trusted set of a particular KSV need to bind itself to this group in order to receive the key update messages sent by its KSV.

In order to send the key-update messages, there needs to be some way for the server to inform the clients about the key it is using to encrypt the current key-update message. A level indicator can identify this value – however there can be multiple nodes at the same level each maintaining the same level-indicator. For example in Figure 9, both keys 12 and 34 have a level-indicator as 2 (considering the root as level 0). In order to remove this ambiguity, we define a new method for identifying the encrypting key.

We define a new ID for each member within a group by encoding the branch indices as an integer value, taken, in order to traverse the hierarchical tree from the root to this member. Hence the ID corresponding to node 192.168.1.1 in Figure 9 is "X100" while the ID for node 192.168.5.0 is "X110". The leading "X" identifies the particular group in the list of all the groups maintained by this KSV. Intermediate key IDs are simply subsets of the leaf IDs – hence to denote the parent node of indexes 192.168.1.1 and 192.168.1.2 we simply specify its ID as "X10". These ID values and their associated lengths are sent in the node registration message (Figure 6) in order to identify a group member completely.

Every message that is sent by the KSV needs to be signed. We see that this is true because using shared key approaches, it is always possible for a node containing the shared key to spoof messages from the KSV and hence confuse any revocation attempt by the KSV. Update messages that do not contain a valid signature are ignored and a SIGN_FAILED alarm (Section 3.2.3) is sent back to the KSV. In order for the group update process to work reliably we need some guarantees from the multicast channel itself. The algorithm will work incorrectly if packets are lost in the multicast channel. We however treat this issue as one inherent of multicast communication and beyond the scope of our problem.

Adding a new member to a multicast group can be a very costly process in this approach. This is because, in order to keep the B+ tree balanced, ID values of

different nodes may change when a new member is added. New ID values for all affected nodes then need to be sent separately using register messages.

3.7 Integrating the packet filter

This security framework defined in its present form yet has a serious problem – spoofing is still very easily possible. To see how this is possible we again allude to the Freeswan IPSec implementation.

The sending end of a Security Association ensures that packets that are sent are afforded all the necessary transforms. At the receiver end, packets with the correct transforms, using the correct keys are accepted, while those failing this step are considered to be bad. There is nothing however, which checks if the sender has simply abstained from IPSec processing altogether. The sender or any third party can change the source address on the packet without adding any IPSec headers and the receiver would just go ahead and treat this as a normal packet. This problem arises basically due to the lack of an inbound policy check in the Freeswan implementation.

We provide an external policy check using a simple packet filter such as IPChains. IPChains can be integrated using the existing "updown" script within the FreeSwan implementation, to add and remove the filter rules as different connections are setup and destroyed.

4 Evaluation

4.1 Security Associations

While Node-Node security associations function as expected, multicasting at the time of this writing suffers from a curious problem. We tried testing multicast using a simple reader-writer application on a multicast address. While security associations were found to be set up correctly, packets were still not detected at the readers. One observation we made was that the IGMP join messages themselves were not being sent out of the readers, which led us to believe that there was some filtering happening at the Ethernet layer. [3] Suggested that routing packets through an "eroute" is what causes this behavior. We could not confirm this fact however. We supply a temporary fix at this stage by modifying the Ethernet driver code to accept all multicast packets seen on the Ethernet bus. The readers and writers now function as expected.

4.2 Timing Evaluation

DNSSEC processing typically takes about 2.5ms on average. In order to investigate the latency imposed by IPSec encapsulation, we run a simple application-level "ping" and examine the round-trip overhead. Expectedly, IPSec processing introduces a significant overhead. While the average RT-time between two nodes without a Security Association between them is 0.58 ms, the same, using either IPSec ESP or AH processing doubles to about 1.2 ms.

A more interesting problem, however, is comparing the time taken for explicit keying between every KSV-client pair and hierarchical keying using the LKH mechanism whenever a member is revoked. We perform this test as follows:

1. Configure a multicast group containing different testnodes
2. Register every node with the KSV so as to maximize the number of keys we may have to send during a re-key
3. Remove every nodes from the multicast group incrementally, so as to trigger any keying updates on our multicast channel
4. Perform the same operation this time using individual KSV-client re-keys for every node currently registered at the KSV
5. Repeat steps 1 through 4 for different orders of the LKH tree

Table 2 and Figure 10 summarize our comparison results for explicit keying (EK) and hierarchical keying using the LKH approach.

Revoke this testnode =>		10	9	7	6	5	4	3	2	1
Order = 1	EK	4.4	3.8	3.3	2.8	2.3	1.7	1.2	.6	-
	LKH	236.2	235.3	158.1	157.0	89.6	166.1	78.5	78.6	-
Order = 2	EK	4.4	3.8	3.2	2.7	2.2	1.7	1.1	.6	-
	LKH	336.5	235.6	157.0	235.3	156.9	78.5	156.8	85.3	-
Order = 3	EK	4.4	3.9	3.2	2.7	2.2	1.7	1.1	.6	-
	LKH	414.5	314.0	235.2	163.1	90.0	244.9	164.0	78.5	-

Table 2Time in ms taken for Explicit (EK) and LKH keying for different Orders of the LKH tree

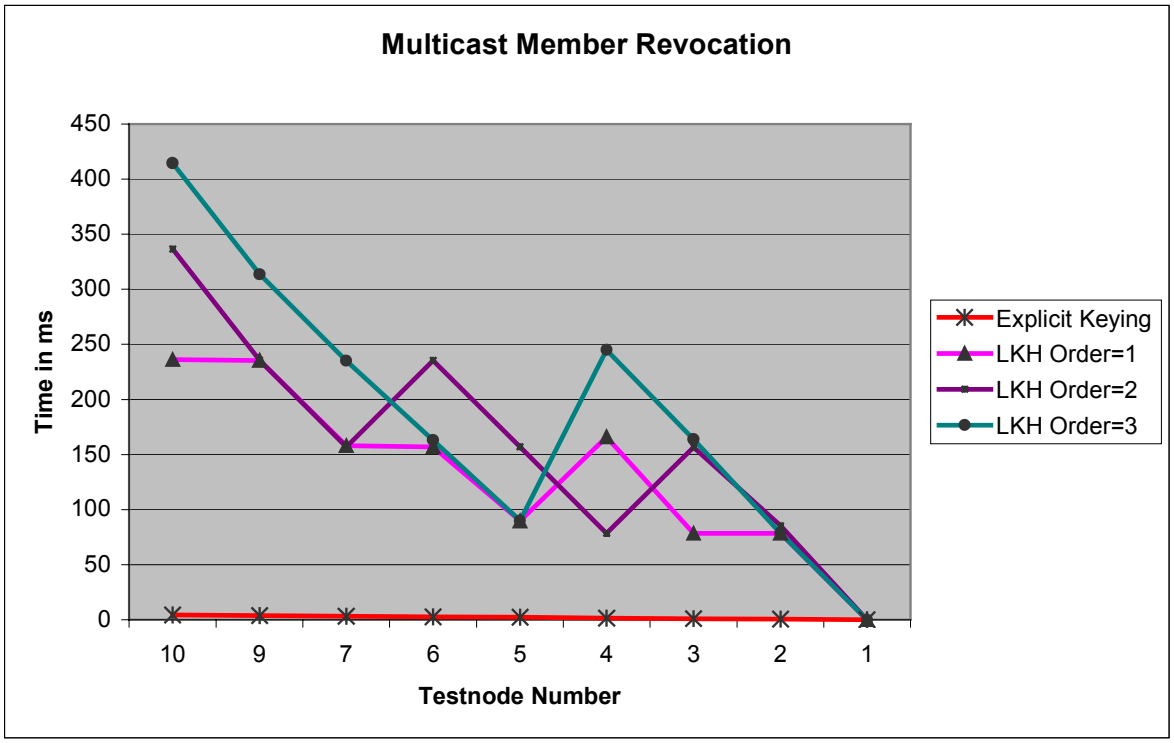


Figure 10 Comparison between Explicit Keying and LKH for multicast member revocation

To investigate the behavior of the LKH approach, we need to see how the LKH trees are created in the first place. Figure 11 depicts this information.

We observe that the timing behavior for the LKH trees directly depend on the number of public-key operations required to be performed during the revocation process. Thus for the LKH tree with order=1, the number of public-key operations required when testnode10 is revoked is 3 – one for every sibling node - (5,6) and (7,9), and one for the adjacent branch. Revocation of testnode9 also has the same number of public-key operations – one for testnode7, one for its sibling node – (5,6) and finally one for the adjacent branch.

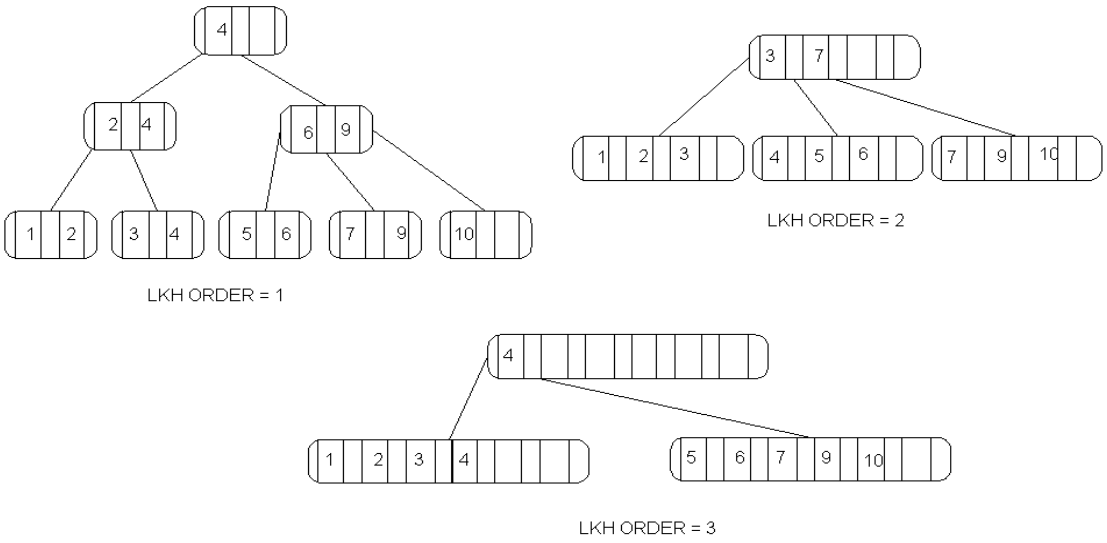


Figure 11 LKH trees for Tree Orders of 1, 2, and 3

This result is interesting because it tells us that simply increasing the order of the LKH tree does not necessarily improve the key-update time. On an average, the update time increases because there are more number of key transmissions required for larger orders. Even though a broader LKH tree requires a lesser number of keying updates *across* branches for propagating key revocations, the delay overhead caused by sending an update for every sibling in the current node is significant.

Finally, we observe that the LKH mechanism is more than an order of magnitude slower than its explicit key-exchange counterpart due to the expensive public key operations required in its implementation.

In summary, it is clear that for small number of nodes the advantage of the LKH algorithm is amortized by the overhead of the public key operations required in its implementation. Explicit keying for every peer also does not scale well and becomes

comparable to the LKH mechanism as the number of nodes increase. An interesting future enhancement could be combining Explicit Keying with the LKH approach in order to reduce the overall latency during group member revocation.

5 Conclusions

This paper describes our effort at building a prototype framework for dynamically setting up node-node Security Associations. In this prototype we have successfully built in and/or integrated support for the following:

- Source-Authentication using DNSSEC
- Peer-Authentication, Integrity and Confidentiality services using the IPSec infrastructure
- A Keying mechanism using IKE and the Logical Key Hierarchy for group member revocation.

We suggest some possible areas for future work:

1. Configuration of secure topologies currently lacks a GUI. Also, the parameters used in the security association can be extended to include specific algorithms and IPSec modes.
2. The multicast member revocation currently uses the LKH mechanism. The Enhanced LKH+ mechanism suggests some optimizations to this basic protocol. Other approaches for optimizing key updates should also be evaluated
3. We assume the Keying Server to have a single public-private key-pair. While this assumption itself does not compromise security, servers capable of connecting to clients on different interfaces may want to use different public keys for each interface
4. Our implementation does not handle inter-domain KSV management. While clients can register with multiple servers, there currently exists no mechanism for key management and SA arbitration between nodes belonging to two different domains.

References

[1] Berson, S., et al, "Introduction to the Abone", white paper: http://www.isi.edu/abone/DOCUMENTS/ABoneIntro.ps, June 15, 2000

[2] Braden, B., et al, "A Proposed ABone Network Security Architecture", ABONE-DRAFT, November, 1999

[3] Canetti, R., et al, "An IPSec--based Host Architecture for Secure Internet Multicast", Proceedings of NDSS '2000, 2000

[4] Doraswamy and Harkins, " IPSEc The New Security Standard for the Internet, Intranets, and Virtual Private Networks", Prentice-Hall, Inc., 1999

[5] Kent, S., et al, "Security Architecture for the Internet Protocol", IETF Network Working Group, RFC 2401

[6] Lindell, B., "Active Networks Protocol Specification for Hop-By-Hop Message Authentication and Integrity", ABONE-DRAFT, December 1999

[7] Maughan, D., et al, "Internet Security Association and Key Management Protocol (ISAKMP)", IETF Network Working Group, RFC 2408

[8] Murphy, S., "Security Architecture for Active Nets", AN Security Working Group, July 15, 1998

[9] Piper, D., "The Internet IP Security Domain of interpretation for ISAKMP", IETF Network Working Group, RFC 2407

[10] Schneier, B., "Applied Cryptography", Second Edition, John Wiley & Sons, Inc, 1996

[11] Wallner, D., et al, "Key Management for Multicast: Issues and Architectures", IETF Network Working Group, RFC 2627