Proceedings of the 9th USENIX Security Symposium

Denver, Colorado, USA
August 14–17, 2000

A Multi-Layer IPsec Protocol

Yongguang Zhang Bikramjit Singh
HRL Laboratories, LLC
{ygz,bsingh}@hrl.com

Abstract

IPsec [KA98c] is a suite of standard protocols that provides security services for Internet communications. It protects the entire IP datagram in an "end-to-end" fashion; no intermediate network node in the public Internet can access or modify any information above the IP layer in an IPsec-protected packet. However, recent advances in internet technology introduce a rich new set of services and applications, like traffic engineering, TCP performance enhancements, or transparent proxying and caching, all of which require intermediate network nodes to access a certain part of an IP datagram, usually the upper layer protocol information, to perform flow classification, constraint-based routing, or other customized processing. This is in direct conflict with the IPsec mechanisms. In this research, we propose a multi-layer security protection scheme for IPsec, which uses a finer-grain access control to allow trusted intermediate routers to read and write selected portions of IP datagrams (usually the headers) in a secure and controlled manner.

1 Introduction

The Internet community has developed a mechanism called *IPsec* for providing secure communications over the public Internet. IPsec can provide data integrity, origin authentication, data confidentiality, access control, partial sequence integrity, and limited traffic flow confidentiality services for communications between any two networks or hosts [KA98c]. By addressing the security issues at the IP layer and rendering the security services in a transparent manner, IPsec attempts to relieve software developers from the need to implement security mechanisms at different layers or for different Internet applications. Arguably, IPsec is the best available mechanism for Virtual Private Networks (VPN) and secure remote accesses.

1.1 The Protection Model in IPsec

The fundamental concept behind the IPsec technology is as follows. The path between an IP datagram's source and destination is divided into three segments (see Figure 1) — the protected and trustworthy local network at the source (e.g., a company's private LAN), the untrustworthy public Internet segment, and the protected and trustworthy local network at the destination. The IPsec architecture places a security gateway (here G_1 and G_2) at each boundary between a trustworthy and an untrustworthy network. Initially, G_1 at the source establishes a security association with G_2 on the destination side, which is a security relationship that involves negotiation of security services and shared secrets. Before an IP datagram (from S to D) is sent to the untrustworthy Internet, the security gateway (G_1) encrypts and/or signs the datagram using an IPsec protocol. When it reaches the security gateway at the destination side (G_2), the datagram is decrypted and/or checked for authentication, before it is forwarded to the destination (D). In some cases, the trustworthy local network on either side can be omitted, and the source or destination host can perform encryption, authentication and other security-gateway functions itself.

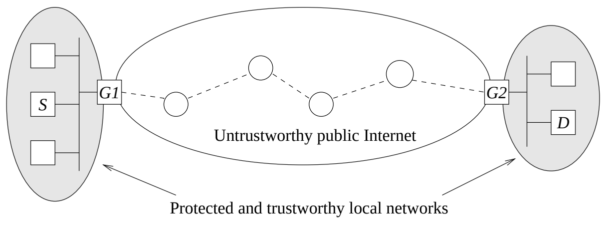
G1
S
G2
D
Untrustworthy public Internet
Protected and trustworthy local networks

Figure 1: System Model

The IPsec architecture uses two protocols to provide traffic security – AH (Authentication Header) [KA98a] and ESP (Encapsulating Security Payload) [KA98b]. AH provides integrity and authentication without confidentiality; ESP provides confidentiality, with optional integrity and authentication. Each protocol supports two modes of use: *transport mode* and *tunnel mode*. Transport mode provides protection primarily for upper layer protocols, while in tunnel mode the protection applies to the entire IP datagram.

The granularity of security protection in the IPsec architecture is at the datagram level. It treats everything in an IP datagram after the IP header as one integral unit. Usually, an IP datagram has three consecutive parts – the IP header (for routing purposes only), the upper layer protocol headers (for example, the TCP header), and the user data (for example, the TCP data). In transport mode, an IPsec protocol header (AH or ESP) is inserted in after the IP header and before the upper layer protocol header to protect the upper layer protocols and user data. In tunnel mode, the entire IP datagram is encapsulated in a new IPsec packet (a new IP header followed by an AH or ESP header). In either case, the upper layer protocol headers and data in an IP datagram are protected as one indivisible unit (see Figure 2).

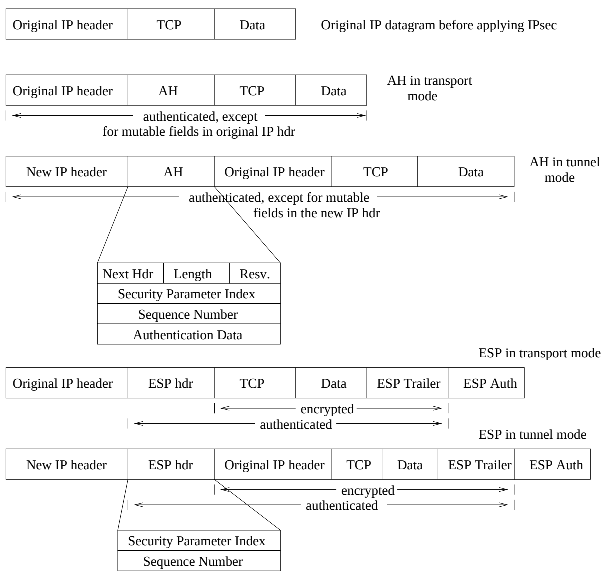

Figure 2: The Protocol Formats of IPsec-protected IPv4 Packets (assuming TCP)

The keys used in encryption and authentication are shared only by the sender-side and receiver-side security gateways. All other nodes in the public Internet, whether they are legitimate routers or malicious eavesdroppers, see only the IP header and will not be able to decrypt the content, nor can they tamper with it without being detected. Traditionally, the intermediate routers do only one thing – forward packets based on the IP header (mainly the destination address field); IPsec's "end-to-end" model is well-suited to this layering paradigm.

1.2 Limitations of End-to-End Security

However, this protection model and its strict layering principle are unsuitable for an emerging class of new networking services and applications for the next generation Internet. Unlike in the traditional minimalist Internet, intermediate routers begin to play more and more active roles. They often rely on some information about the IP datagram payload, such as certain upper layer protocol header fields, to make sophisticated routing decisions. In other words, routers can now participate in a layer above the IP. Examples of such active networking techniques are:

- Internet traffic engineering. The Internet is moving towards active traffic engineering to meet increasing demand for bandwidth and rich services. Routers/switches will support per-flow and class-based queueing to give fair bandwidth access to all users. A QoS guarantee will be provided to traffic flows generated by paying customers. Router-based congestion control mechanisms, such as Random Early Detection (RED) [FJ93] with penalty box [FF99], also require intermediate nodes to discriminate between traffic flows. Depending on the granularity used in defining a "flow," certain nodes in the middle of the network may need access to information in the upper layer protocols, such as TCP/UDP port numbers, to classify packets into flows before applying discriminating operations.

- Transport-aware link layer mechanisms. The global Internet has accommodated a very wide range of link technologies, but certain transport protocols like TCP have not achieved optimal performance when operated over a path that includes lossy wireless links or long-delay satellite links. For example, in a recent paper [BPSK97], Balakrishnan proves that, to significantly improve the TCP's performance over a wireless link, the base station at the lossy link must be aware of the TCP state information in each passing flow, and deliberately delay or drop certain types of TCP packets. Such link-layer mechanisms for TCP performance improvement (often referred to as *TCP Performance Enhancing Proxies* or TCPPEP [BKGM00]) require intermediate nodes to access and sometimes modify the upper layer protocol headers.

- Application-layer proxies/agents. Some Internet routers can provide application-layer services for performance gains. For example, an intermediate router can become a transparent web proxy when it snoops through the TCP and HTTP header of a bypassing IP datagram to determine the URL request, and serves it with the web page from the local cache. It is transparent to end-users but boosts the responsiveness because the delivery paths for web requests and data between the intermediate router and the web site server are eliminated.

- Active networks. Going one step further, the active network architecture is a new networking paradigm in which the routers perform customized computation on the data flowing through them. A number of experimental active network systems have been developed and they can be run over the Internet. In this architecture, a single IP datagram carries not only upper-layer protocol headers and user data, but also a "method" – a set of executable instructions to be interpreted by the intermediate routers, for describing, provisioning, or tailoring network resources and services in order to achieve the delivery and management requirements. Obviously then, the "method" portion of the IP datagram ought not to be encrypted "end-to-end."

- Traffic Analysis. Many network operators actively monitor the traffic for accounting or for intrusion detection purposes. Usually, such monitoring requires logging of certain upper layer protocol information, like TCP/UDP ports. Many firewalls that protect local networks also depend on such information to deny unauthorized traffic.

All these mechanisms require intermediate network nodes to access information encoded in the IP datagram payload, but the current IPsec technology advocates end-to-end security and prevents such access. This fundamental conflict [NBB99] makes it a very difficult problem to provide both security and extensibility in one unified platform.

1.3 Problem Statement

The goal of this research is to develop a security scheme that supports the above new network services and applications under the IPsec framework. The new scheme should *grant trusted intermediate routers a secure, controlled, and limited access to a selected portion of certain IP datagram, while preserving the end-to-end security protection to user data.*

2 Approaches

We have investigated three ways to solve the problem – replacing IPsec with a transport-layer security mechanism, using a transport-friendly ESP format, and developing a multi-layer protection model for IPsec.

The first approach, replacing IPsec with a *transport-layer mechanism*, circumvents the problem of intermediate nodes not being able to access the encrypted TCP headers, yet introduces certain other difficulties. There are actually several transport-layer security mechanisms available today, including SSL (most notably used in Netscape and other WWW applications) or TLS (a proposed IETF standard [DA99]). Both SSL and TLS encrypt the TCP data while leaving the TCP header in unencrypted and unauthenticated form so that intermediate nodes can make use of the TCP state information encoded in the TCP header. However, letting the entire TCP header appear in clear text exposes several vulnerabilities of the TCP session to a variety of TCP protocol attacks (in particular traffic analysis), because the identity of sender and receiver are now visible without confidentiality protection.

Alternatively, it is possible to tunnel one security protocol within another, such as SSL/TLS inside an IPsec ESP – letting SSL/TLS protect the TCP data and ESP protect the TCP header. However, there is a problem here too because ESP encrypts both TCP header and TCP payload (SSL/TLS-protected data) as a whole. Thus, the encryption/authentication/decryption has to be done twice on the TCP data part, an unnecessary waste of resources. The intermediate router, for example, must decrypt the entire packet to access just the TCP header information.

The second approach is to develop a *transport-friendly ESP* (TF-ESP) protocol format for IPsec. Proposed by Steve Bellovin of AT&T Labs [Bel99], TF-ESP modifies the original ESP protocol to include limited TCP state information, such as flow identifications and sequence numbers, in a disclosure header outside the encryption scope (but authenticated). This approach will work well for some TCP PEP mechanisms such as TCPPEP for wireless network (e.g., TCP snooping), but it may not suite other mechanisms that need a write access, such as TCPPEP for satellite networks [ZDRD97, BKGM00]. To support TCPPEP for satellite networks, the TCP state information also needs to be placed outside the authentication scope. Without proper integrity protection, this can be dangerous. Further, the unencrypted TCP state information is made available universally, including to untrustworthy nodes, which creates vulnerability for possible attacks. In addition, TF-ESP is not flexible enough to support all upper-layer protocols.

Since the above two approaches both have limitations, we thus propose a third approach – to develop a *multi-layer security protection scheme* for IPsec. The idea is to divide the IP datagram into several parts and apply different forms of protection to different parts. For example, the TCP payload part can be protected between two end points while the TCP/IP header part can be protected but accessible to two end points plus certain routers in the network. The rest of this paper will describe the principle, the design and an implementation of this approach.

3 The Principle of Multi-Layer Security Protection

Our approach is called ML-IPsec (Multi-Layer IPsec). It uses a multi-layer protection model to replace the single end-to-end model. Unlike IPsec where the scope of encryption and authentication apply to the entire IP datagram payload (sometimes IP header as well), our scheme divides the IP datagram into zones. It applies different protection schemes to different zones. Each zone has its own sets of security associations, its own set of private keys (secrets) that are not shared with other zones, and its own sets of access control rules (defining which nodes in the network have access to the zone).

When ML-IPsec protects a traffic stream from its source to its destination, the first IPsec gateway (or source) will re-arrange the IP datagram into zones and apply cryptographic protections. When the ML-IPsec protected datagram flows through an authorized intermediate gateway, a certain part of the datagram may be decrypted and/or modified and re-encrypted, but the other parts will not be compromised. When the packet reaches the last IPsec gateway (or destination), ML-IPsec will be able to reconstruct the original datagram. ML-IPsec defines a complex security relationship that involves both the sender and the receiver of a security service, but also selected intermediate nodes along the traffic stream.

For example, a TCP flow that desires link-layer support from the network can divide the IP datagram payload into two zones: TCP header and TCP data. The TCP data part can use an end-to-end protection with keys shared only between the source and the destination (hosts or security gateways). The TCP header part can use a separate protection scheme with keys shared among the source, the destination, and certain trusted intermediate node. (See Figure 3.) This way, no one in the public Internet other than the source, the destination and the trusted intermediate nodes has access to TCP header or TCP data, and no one other than source and destination (not even the trusted intermediate node) has access to TCP data.

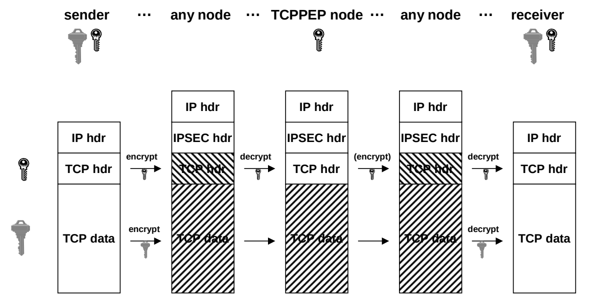

Figure 3: Multi-Layer Protection Model for TCP

This scheme in effect provides a finer-grain access control to the IP datagram. Since ML-IPsec allows network operators and service providers to grant intermediate nodes limited access to IP datagram contents parts (such as TCP header), such access must be granted in a secure and controllable way. The identity of the intermediate nodes must be authenticated (using an out-of-band mechanism such as a public-key infrastructure) to prevent any man-in-the-middle attack. After authentication, keys or shared secrets corresponding to the authorized IP datagram zones must be distributed to the intermediate nodes, also using out-of-band mechanisms like IKE [HC98].

4 ML-IPsec Design Details

The architecture of ML-IPsec embraces the notion of zones, a new type of security association, the new AH and ESP header formats, and the inbound/outbound processing of ML-IPsec protocol packets. It is designed to be fully compatible with the original IPsec in both protocol formats and processing software.

4.1 Zones

A zone is any portion of IP datagram under the same security protection scheme. The granularity of a zone is 1 octet. The entire IP datagram is covered by zones, except for the IP header in the transport modes, but zones cannot overlap. Using the same TCP example, the portion of the IP datagram that contains TCP header (21st to 40th octet) is Zone 1, and the TCP data portion (41st and above octet) is Zone 2 (assuming transport mode and no TCP options).

A zone need not be a continuous block in an IP datagram, but each continuous block is called a *subzone*. A *zone map* is a mapping relationship from octets of the IP datagram to the associated zones for each octet. Figure 4 below shows a sample zone map.

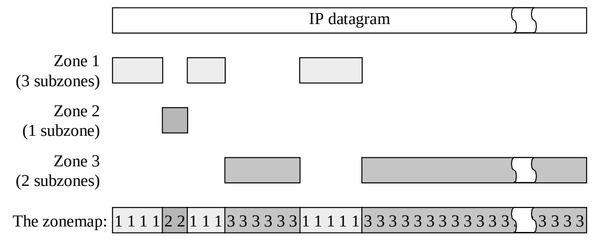

Figure 4: A Sample Zone Map

The zone map is a constant in a security relationship. That is, the zone boundaries in each IP datagram must remain fixed in the lifetime of the security association; otherwise, it will be extremely difficult to do zone-by-zone decryption and authentication. Since IP datagrams are variable in length, the zone that covers the last part of the datagram, usually the user data, should also be variable in size. Zone 3 of the above is an example. It is also possible, theoretically, to define a phantom zone that does not correspond to any byte in an IP datagram.

4.2 Security Association (SA)

4.2.1 Original SA for IPsec

Security Association (SA) is a key concept in the IPsec technology [KA98c]. It is a one-way relationship between a sender and a receiver that affords security services. Each SA defines a set of parameters including the sequence number and anti-replay window for anti-replay service, the protocol mode (transport or tunnel), the lifetime of the SA, the path MTU and other implementation details. For authentication services in AH or ESP, each SA also defines the choice of cryptographic algorithm, the crypto-keys, key lifetimes and related parameters. For encryption services in ESP, each SA further defines the choice of encryption algorithm, the encryption keys, the initial values, key lifetimes, etc. When an outbound IP datagram passes the security gateway, the IPsec module first compares the values of the appropriate fields in the IP datagram (the selector fields) against a set of predefined policies, called SA selectors, in the Security Policy Database (SPD). It then determines the SA for this datagram if any, and does the required security processing (e.g., encryption). When an inbound IPsec datagram passes the security gateway, the IPsec module uses the SPI (Security Parameter Index) field to determine the SA for this datagram and performs security processing (e.g., decryption). Figure 5 gives a simple illustration of how these pieces are connected together in the IPsec architecture.

4.2.2 Composite SA for ML-IPsec

SA in the original IPsec defines a simple security relationship from the sender to the receiver that affords the protection service. ML-IPsec however requires a much more complex security relationship to include sender and receiver, as well as the selected intermediate nodes. Since the security service is zone-by-zone, conceptually we can use an indi-

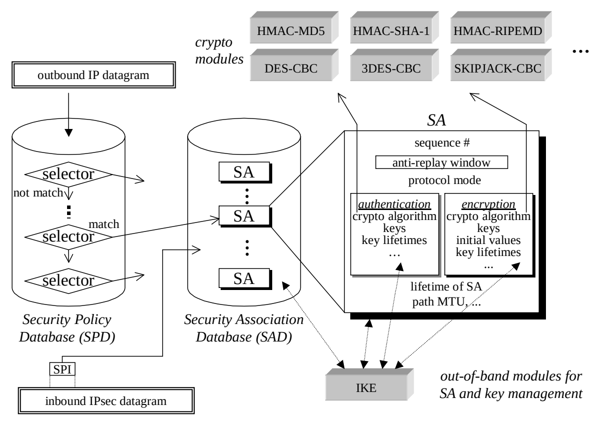

Figure 5: IPsec System Architecture

vidual security relationship to cover each zone, and then build a composite relationship to cover the entire IP datagram. Mapping this idea to the basic Security Association (SA) concept, ML-IPsec needs a new type of SA called *Composite SA* (CSA). CSA is a collection of SAs that collectively afford a multi-layer security protection for the traffic stream.

A CSA has two elements. The first element is a zone map. The zone map specifies the coverage of each zone in an IP datagram. The zone map must be consistent in all nodes involved in the same ML-IPsec relationship. The second element in a CSA is a *zone list*. A zone list is a list of SAs for all the zones. Each and every such SA is stored in the Security Association Database (SAD) [KA98c]. However, some of the fields are used differently in ML-IPsec than as defined in the original IPsec [KA98c]. The following SAD fields, for example, are applicable only on the corresponding zone of the SA.

- Lifetime of this Security Association
- AH Authentication algorithm, keys, etc.
- ESP Encryption algorithm, keys, IV mode, IV, etc.
- ESP Authentication algorithm, keys, etc.

The other SAD fields have no meanings on the zone level. With the exception of a *designated SA* in the zone list, the following SAD fields are not used in other zonal SAs, although they may be initialized during the SA creation process.

- Sequence Number Counter
- Sequence Counter Overflow
- Anti-Replay Window
- IPsec protocol mode
- Path MTU

The designated SA however operates on these fields as defined in the original IPsec. The designated SA is a special SA in the zone list, usually the first SA in the list. It is responsible for maintaining parameters for the IP datagram layer and "represents" the CSA in security processing.

The zone map and zone list can be stored with the designated SA as additional fields in the SAD, or, they can be stored in a separate CSA database. This is an implementation choice and it allows flexibility in adding ML-IPsec features to an existing IPsec implementation.

On inbound processing, if the traffic stream is under ML-IPsec protection, the destination IP address, the IPsec protocol type, and the SPI identifies an entry in the SAD, which points to the designated SA of the CSA for this traffic stream. Or, under alternative implementation, the triplet identifies an entry in the CSA database. By traversing CSA's zone list, ML-IPsec can further identify the SA entries for all the zones.

On outbound processing, the Security Policy Database (SPD) [KA98c] will have a pointer to the designated SA or an entry in the CSA database. Just as in the original IPsec, the selectors will direct the outbound traffic to the proper SPD entry.

4.2.3 Access Control in a CSA

A CSA involves the sender, receiver, and all the authorized intermediate nodes that collectively provide a multi-layer security protection for a traffic stream. Therefore, an instance of CSA must be created in each of these nodes before the ML-IPsec service can commence. It will have these features. First, the zone map must be distributed and remain the same for all nodes. Second, each CSA instance must have a designated SA, and the choice of designated SA must be consistent across all the nodes. Finally, because the designated SA is the one and only SA responsible for the integrity of the IPsec header (with fields like SPI and sequence number), all these nodes must be able to process this SA.

However, the zone list need not be the same for all nodes. In principle, each zonal SA independently determines the access list for that zone and not all nodes will have access to all zones. If some node does not have access to a zone, the corresponding zonal SA in the zone list will be null. For a particular zonal SA, an instance must be created in each authorized node and stored in its SAD as a step in CSA creation. By determining which zonal SA is to be created in which node, CSA enforces a multi-layer access control for an IP traffic stream.

Since the designated SA must be consistent across all nodes involved in a CSA, they should all have access to the corresponding zone. For convenience, we call this zone for which the designated SA is chosen the *designated zone*. The requirement that all nodes must have access to one common zone is very natural in most applications; the designated zone is usually the first zone in the list, containing the IP header plus certain upper protocol headers. In rare cases where the zones accessible by intermediate nodes are disjoint, we must define a phantom zone of zero size and make it the designated zone. We can now make an SA for this zone and use it as the designated zone. This however introduces extra overhead because the protocol needs to accommodate one more SA.

4.2.4 A TCP Example

Here is an example to illustrate the concept of CSA. It is a traffic flow from Sender (the ultimate source or the outbound IPsec gateway) to Receiver (the ultimate destination or the inbound IPsec gateway), passing through Gateway (an intermediate router providing diffserv or TCPPEP service). Let's assume the desired security service is ESP transport mode.

The corresponding CSA in Sender or Receiver will have the following elements:

- zone map
  - zone 1 = byte 1-20
  - zone 2 = byte 21-EOP (*end-of-packet*)
- zone list
  - SA1 (designated)
    - sequence number counter
    - sequence counter overflow
    - anti-replay window
    - protocol mode = TRANSPORT
    - path mtu
    - lifetime
    - ...
    - encryption algorithm = DES-CBC
    - encryption key = *key1*
    - authentication algorithm = HMAC-MD5-32
    - authentication key = *key2*
    - ...
  - SA2
    - ...
    - lifetime
    - ...
    - encryption algorithm = 3DES-CBC
    - encryption key = *key3*
    - authentication algorithm = HMAC-MD5-96
    - authentication key = *key4*
    - ...

The corresponding CSA in Gateway will have the following elements:

- zone map
  - zone 1 = byte 1-20
  - zone 2 = byte 21-EOP
- zone list
  - SA1 (designated)
    - sequence number counter
    - sequence counter overflow
    - anti-replay window
    - protocol mode = TRANSPORT
    - path mtu
    - lifetime
    - ...
    - encryption algorithm = DES-CBC
    - encryption key = *key1*
    - authentication algorithm = HMAC-MD5-32
    - authentication key = *key2*
    - ...
  - SA2 = NULL

Here HMAC-MD5-32 is a hypothetical keyed hash algorithm that produces a smaller 4-octet signature. Using smaller size authentication data on certain zones (usually the protocol headers) has the advantage of lower overhead. Otherwise, the standard HMAC-MD5-96 can be used.

ML-IPsec has an unintended benefit in this case – it is actually more secure than the original IPsec, because the chosen plaintext attacks [Bel96] become more difficult now. Under the original IPsec, chosen plaintext attacks can compromise user data carried in an IPsec packet by exploiting the fact that the values in many protocol header fields are predictable; that is, the encryption key can be discovered by comparing the plaintext and ciphertext versions of these fields. However, even so this will not compromise the user data under ML-IPsec, because it uses a different key from the headers.

4.3 AH and ESP Headers

The same security protocol formats, AH and ESP, are used in ML-IPsec. Both AH and ESP have transport mode or tunnel mode, as indicated by the "protocol mode" field of the designated SA. Figure 6 describes the format for both headers used in ML-IPsec.

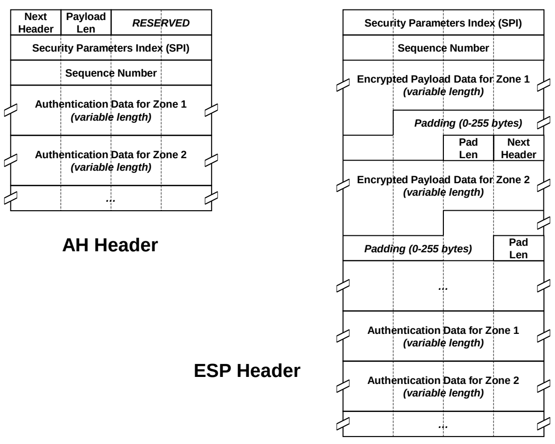
Next Header
Payload Len
RESERVED
Security Parameters Index (SPI)
Sequence Number
Authentication Data for Zone 1 (variable length)
Authentication Data for Zone 2 (variable length)
...
AH Header
Security Parameters Index (SPI)
Sequence Number
Encrypted Payload Data for Zone 1 (variable length)
Padding (0-255 bytes)
Pad Len
Next Header
Encrypted Payload Data for Zone 2 (variable length)
Padding (0-255 bytes)
Pad Len
...
Authentication Data for Zone 1 (variable length)
Authentication Data for Zone 2 (variable length)
...
ESP Header

Figure 6: ML-IPsec Protocol Header Format

The protocol header format for AH in ML-IPsec is almost identical to the original IPsec AH [KA98a], except that the Authentication Data section in AH is further subdivided into zones. The Authentication Data field is a variable-length field that contains several Integrity Check Values (ICVs) for this packet. The total length of this field is controlled by Payload Len. The size of each ICV is determined by the authentication algorithm used in each zonal SA, but must be an integral multiple of 32 bits. The boundaries of these zonal authentication data sections can be derived from the CSA.

ML-IPsec is perhaps more useful in ESP, where the IP datagram can be encrypted using different keys in different SAs. The ML-IPsec ESP header format follows the principle in IPsec ESP. But unlike IPsec ESP, the Payload Data field in ML-IPsec ESP is broken into pieces, one for each zone. The Payload Data for each zone, together with Padding, Padding Length, and Next Header field (only in the designated zone), are collectively referred to as the ciphertext block for the zone. The size of each ciphertext block can be determined by the CSA, since all zones except the last one are fixed in size.

Similar to ML-IPsec AH, the optional Authentication Data field in ESP is also variable in length and contains several Integrity Check Values (ICVs) for this packet. The size of each ICV is determined by the authentication algorithm used in each zonal SA, but must be an integral multiple of 32 bits. The boundaries of these zonal authentication data sections can be derived from the CSA.

4.4 Inbound and Outbound Processing in ML-IPsec

4.4.1 ICV Calculation and Verification

The AH ICV calculation in ML-IPsec is rather different from that in the original IPsec. For the designated zone, the ICV is computed over:

- IP header fields that are immutable in transit.
- The AH header, including Next Header, Payload Len, Reserved, SPI, Sequence Number, and the Authentication Data (which is set to zero for this computation), and the optional explicit padding bytes if any.
- All octets in the designated zone.

For other non-designated zones, the ICV is computed only over the octets of the zone.

The ICV verification during the inbound processing of an ML-IPsec datagram is also done zone-by-zone. A zone is authenticated only if the corresponding zonal SA is non-null. The ICVs are calculated in the same way as described above, and the values are then matched against the ICVs stored in the Authentication Data. In an intermediate node, a packet will go through inbound processing and then outbound processing. If changes are made to the packet in an authorized zone, the ICV is recomputed and stored in a proper place in the Authentication Data field. The ICVs of the unchanged zones are left untouched.

4.4.2 Zone-by-Zone Encryption

On outbound processing, the sender takes the following steps in packet encryption:

1. **Zone-wise Encapsulation.** For each zone, all octets of all sub-zones are concatenated (in the order they appear in a datagram) and then encapsulated into the ESP Payload Data field for the corresponding zone.
2. **Padding.** The sender adds any necessary padding to each zone's Payload Data field, to meet the encryption algorithm's block size requirement if any, and to align it on a 4-byte boundary according to the ML-IPsec ESP format.
3. **Encryption.** The sender then encrypts the resulting plaintext (Payload Data, Padding, Pad Length, and Next Header) using the key, the encryption algorithm, and the algorithm mode indicated by the zonal SA and cryptographic synchronization data (if any).

4.4.3 Outbound processing

The outbound processing of an IP datagram in an IPsec gateway is illustrated in Figure 7 through a 2-zone example. The plaintext from each zone is masked by the zone map, concatenated into a continuous block, and passed through the zone-by-zone encryption as described above in Section 4.4.2. If a zone is itself a continuous block before the masking, we must do optimization to avoid extra copying; the encryption should be operated directly at the IP datagram buffer. For ESP, the ciphertexts from all zones are concatenated and stored in the ESP payload data field of an ML-IPsec ESP packet. After packet encryption, for AH and ESP with an authentication option, the sender computes the ICV from the ciphertext of each zone according to Section 4.4.1. The ICVs are then concatenated and stored in the Authentication Data field of the final outgoing ML-IPsec AH or ESP packet.

4.4.4 Inbound Processing

The inbound processing of an IPsec packet is almost a simple reverse of the outbound processing (see the

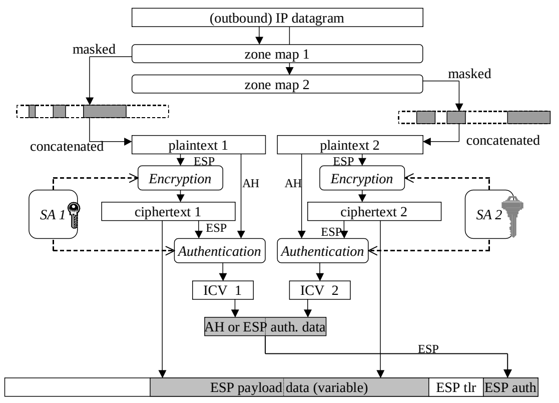

Figure 7: An Example of ML-IPsec Outbound Processing

same 2-zone example in Figure 8). If the SA structure indicated by the SPI value is a CSA type, the ML-IPsec processing mode is triggered. ML-IPsec first performs ICV verification if the protocol is AH, or if the protocol is ESP with an authentication option. The ICV check is done zone-by-zone according to Section 4.4.1. If the ICV check fails for any one zone, the entire datagram is discarded. The next step is the zone-by-zone decryption if the protocol is ESP. For a zone whose zonal SA is valid and non-null, the receiver decrypts the ESP Payload Data, Padding, Pad Length, and optional Next Header using the key, encryption algorithm, algorithm mode, and cryptographic synchronization data (if any), indicated by the zonal SA. After processing Padding, the receiver then reconstructs the original IP datagram from the original IP header (transport mode) or the tunnel IP header (tunnel mode), plus the IP payload stored in all the Payload fields. In the reverse procedure of encryption, the receiver takes Payload Data of a zone and restores the bytes back according to the zone map. If a zone has a null SA, the bytes corresponding to the zone map will be left zero.

4.4.5 Partial Datagram Processing at Intermediate Routers

In an intermediate node that is authorized to access at least one zone, a bypassing ML-IPsec datagram will go through inbound processing and then outbound processing if changes have been made (see Figure 9). The processing requires extra care because it may not have all the SAs to process the entire datagram. It must ignore zones for which

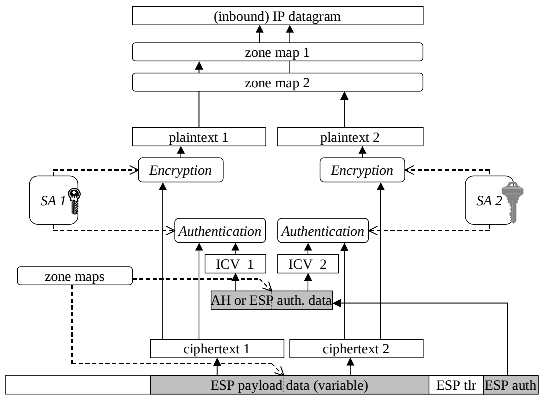

Figure 8: An Example of ML-IPsec Inbound Processing

keys are not granted. If the protocol is ESP, the decrypted plaintext may be incomplete for the original datagram, but it is likely to have the zone it needs (e.g., TCP header) for customized operations. If the intermediate node modifies the plaintext (e.g., in TCPPEP), it must redo authentication and/or encryption for that zone, and replace the corresponding ICV and/or Payload Data field in the bypassing IPsec datagram, before forwarding to the next hop.

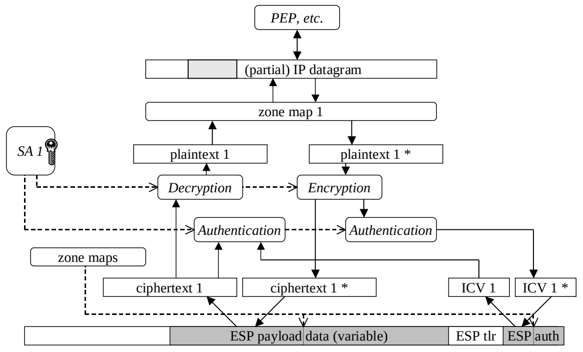

Figure 9: An Example of Partial Inbound/Outbound Processing

4.5 Bandwidth Overhead Analysis

The extra overhead introduced by the multi-layer protection model includes IPsec datagram size and processing load. The datagram size in a two-zone ML-IPsec is likely to increase when we do authentication or encryption as two separate plaintext blocks instead of one trunk in the original IPsec. For example, the concatenated ciphertext from two individually encrypted plaintexts might be larger than the single ciphertext of the concatenated plaintext, due to the synchronization data (such as an initialization vector in some encryption algorithm) and separate padding. The authentication data field is also bigger. For example, if HMAC-MD5-96 is used, the ICV is a fixed size of 12 bytes. If the CSA has two zones, the new IPsec datagram will increase by 12 bytes compared with the original IPsec one. To understand the increase in packet size caused by ML-IPsec, we conduct protocol analysis on TCP applications.

The datagram used in the analysis is a TCP datagram, with 20-byte IP header, 20-byte TCP header, no IP options, and no TCP options. For ML-IPsec, we assume the TCP datagram is divided into two zones, one for TCP/IP headers, and the other for the TCP payload. We calculate the overhead for both AH and ESP protocols (with authentication) and for both transport and tunnel mode. We analyze both ML-IPsec and IPsec for comparison purposes. In all the cases we assume use of the HMAC-MD5-96 algorithm for authentication and the 3DES-CBC algorithm for encryption.

We have analyzed the overhead for all eight cases (AH vs. ESP, Transport mode vs. Tunnel mode, and IPsec vs. ML-IPsec). Due to space limitations, we will not enumerate the calculation in detail. We only summarize the results in Tables 1 and 2. The variable n denotes the length of the TCP payload in the original IP datagram.

While the use of IPsec to protect IP datagrams adds an overhead ranging from 24 to 57 bytes, the new ML-IPsec scheme adds only an additional 12 bytes to the AH protocol and a maximum 20 bytes to the ESP protocol. Assuming an average IP datagram of 536 bytes, that is only a 2-3% increase. One way to further reduce the overhead increase is to use a "weaker" authentication algorithm for the "less important" field. For example, a 4-byte HMAC-MD5-32 ICV may be sufficient for the TCP header zone, instead of the 12-byte HMAC-MD5-96 ICV in the original IPsec. This saves 8 bytes for each ML-IPsec packet and brings the overhead down to the 1-2% range.

Table 1: Packet Length Comparison (bytes)

	Original IP	IPsec	ML-IPsec
AH Transport mode	$40 + n$	$64 + n$	$76 + n$
AH Tunnel mode	$40 + n$	$84 + n$	$96 + n$
ESP Transport mode	$40 + n$	$64 + \lceil(6 + n)/8\rceil * 8$	$92 + \lceil(1 + n)/8\rceil * 8$
ESP Tunnel mode	$40 + n$	$88 + \lceil(2 + n)/8\rceil * 8$	$116 + \lceil(1 + n)/8\rceil * 8$

Table 2: Packet Length Overhead (bytes)

	IP $\rightarrow$ IPsec	IP $\rightarrow$ ML-IPsec	IPsec $\rightarrow$ ML-IPsec
AH Transport mode	24	36	12
AH Tunnel mode	44	56	12
ESP Transport mode	[30,37]	[46,53]	12 or 20
ESP Tunnel mode	[50,57]	[70,77]	20

5 Implementation

5.1 Platform

This implementation of ML-IPsec is done on Linux FreeS/WAN version 1.1 on Linux kernel version 2.2.12. Linux FreeS/WAN (`www.freeswan.org`) is an implementation of IPsec available free (under GNU license term) to users and developers all around the world. The FreeS/WAN system has the following major parts.

- **KLIPS.** The **K**ernel **IP**sec **S**upport part includes the necessary elements in the Linux kernel for running IPsec protocols on the system. Most of the changes required for implementing ML-IPsec using FreeS/WAN are done in this part.
- **The Pluto Daemon.** This part implements the IKE protocol [HC98] – verifying identities, choosing security policies and negotiating keys for the KLIPS layer. Our current ML-IPsec implementation does not change this part, as we are using manual keying only. Our future work will involve modifying the Pluto daemon to facilitate automatic keying among all nodes and specifying ML-IPsec policies.
- **The `ipsec` Command.** This is a user command for controlling IPsec activities, such as setting up and tearing down IPsec tunnels, etc.
- **Linux FreeS/WAN Configuration File.** This file (usually `/etc/ipsec.conf`) contains configuration parameters for setting up IPsec tunnels in the system. We have modified the format of this file to accommodate ML-IPsec functions. The modifications will be listed in a later section.

5.2 Changes in Data Structure

The main changes that we have made are in the data structures for SAs (Security Associations). In original FreeS/WAN, the SAD (security association database) was implemented as a hash table of `struct tdb` nodes. Our ML-IPsec implementation does not change this storage organization, but it modifies the `tdb` structure to store extra fields for ML-IPsec specific data. In addition, we create new structures for “zonemap” and “subzone” as illustrated in the design. The case of a “normal” SA (relating to the the original IPsec) would be handled as a special case of an ML-IPsec SA with one zonemap extending from byte 1 to EOP (end-of-packet). Figure 10 contains a verbatim copy of the new data structure; our changes are annotated by the C preprocessor macro `ML_IPSEC`. The storage organization of these structures in the memory is shown in Figure 12.

Although the AH and ESP header formats in ML-IPsec are different from the original IPsec, the correpsonding programming constructs do not need modification because the fixed length fields remain the same. The variable length fields and the pointer targets are set up properly during the allocation stage for the socket buffers.

The data structure of the control interface (`encap_msghdr`) used for passing the requisite infor-

```
struct tdb                       /* tunnel descriptor block */
{

#ifdef ML_IPSEC
  struct zone         *csa_zonemap;                       /* pointer to the zonemap for the CSA */
  __u8                 desig_sa_flag;                     /* boolean value - designated SA or not*/
#endif

  struct tdb          *tdb_hnext;                         /* next in hash chain */
  struct tdb          *tdb_onext;                         /* next in output */
  struct tdb          *tdb_inext;                         /* next in input (prev!) */
  struct ifnet        *tdb_rcvif;                         /* related rcv encap interface */
  struct sa_id         tdb_said;                          /* SA ID */
  __u32                tdb_seq;                           /* seq num of msg that set this SA */
  __u32                tdb_pid;                           /* PID of process that set this SA */
  struct xformsw      *tdb_xform;                         /* transformation to use (host order)*/
  caddr_t              tdb_xdata;                         /* transformation data (opaque) */
  __u8                 tdb_authalg;                       /* auth algorithm for this SA */
  __u8                 tdb_encalg;                        /* enc algorithm for this SA */

  __u8                 tdb_replaywin;                     /* replay window size */
  __u8                 tdb_state;                         /* state of SA */
  __u32                tdb_replaywin_lastseq;             /* last pkt sequence num */
  __u64                tdb_replaywin_bitmap;              /* bitmap of received pkts */

  __u32                tdb_flags;                         /* generic xform flags */

  __u32                tdb_lifetime_allocations_c;        /* see rfc2367 */
  __u32                tdb_lifetime_allocations_s;
  __u32                tdb_lifetime_allocations_h;
  __u64                tdb_lifetime_bytes_c;
  __u64                tdb_lifetime_bytes_s;
  __u64                tdb_lifetime_bytes_h;
  __u64                tdb_lifetime_addtime_c;
  __u64                tdb_lifetime_addtime_s;
  __u64                tdb_lifetime_addtime_h;
  __u64                tdb_lifetime_usetime_c;
  __u64                tdb_lifetime_usetime_s;
  __u64                tdb_lifetime_usetime_h;
  struct sockaddr     *tdb_addr_s;                        /* src sockaddr */
  struct sockaddr     *tdb_addr_d;                        /* dst sockaddr */
  struct sockaddr     *tdb_addr_p;                        /* proxy sockaddr */
  __u16                tdb_addr_s_size;
  __u16                tdb_addr_d_size;
  __u16                tdb_addr_p_size;
  __u16                tdb_key_bits_a;                    /* size of authkey in bits */
  __u16                tdb_auth_bits;                     /* size of authenticator in bits */
  __u16                tdb_key_bits_e;                    /* size of enckey in bits */
  __u16                tdb_iv_bits;                       /* size of IV in bits */

  __u8                 tdb_iv_size;
  __u16                tdb_key_a_size;
  __u16                tdb_key_e_size;
  caddr_t              tdb_key_a;                         /* authentication key */
  caddr_t              tdb_key_e;                         /* encryption key */
  caddr_t              tdb_iv;                            /* Initialisation Vector */
  __u16                tdb_ident_type_s;                  /* src identity type */
  __u16                tdb_ident_type_d;                  /* dst identity type */
  __u64                tdb_ident_id_s;                    /* src identity id */
  __u64                tdb_ident_id_d;                    /* dst identity id */
  __u8                 tdb_ident_len_s;                   /* src identity type */
  __u8                 tdb_ident_len_d;                   /* dst identity type */
  caddr_t              tdb_ident_data_s;                  /* src identity data */
  caddr_t              tdb_ident_data_d;                  /* dst identity data */
};


#ifdef ML_IPSEC
struct sub_zone
{
  struct sub_zone     *next;                              /* next subzone pointer for the zone*/
  __u16 left;                                             /* left bound of the subzone*/
  __u16 right;                                            /* right bound of the subzone*/
};

struct zone
{
  struct sub_zone     *subzone;                           /* pointer to the first subzone for this zone*/
  struct tdb          *corres_sa;                         /* pointer to the corresponding SA for this zone */
  struct zone         *next;                              /* next zone pointer for the CSA*/
};
#endif
```

Figure 10: The New SA Data Structure in FreeS/WAN Code for ML-IPsec

mation read from the configuration file to the kernel storage of SAs is also changed to accomodate the zone boundaries in the Xfm sub-structure (see Figure 11).

```
struct
{
  struct sa_id Said;    /* SA ID */
  int If;               /* enc i/f for input */
  int Alg;              /* Algorithm to use */

#ifdef ML_IPSEC         /* zone bytes specifications */
  __u16 sbyte[MAX_SUBZONES];
  __u16 ebyte[MAX_SUBZONES];
#endif

  union {               /* Data */
    __u8 Dat[1];
    __u64 Datq[1];      /* maximal alignment (?) */
  } u;
} Xfm;
```

Figure 11: The New Xfm Structure

5.3 Changes in C Functions

A number of C functions in the FreeS/WAN system source files need to be altered to take care of the changes made to the tdb data structure. Most of these files belong to the KLIPS part of the FreeS/WAN system. Rest are the user level files that interact with the KLIPS through the device interface.

The functions like deltdb(), deltdbchain() and ipsec_tdbcleanup(), which deal with deletion of SAs from the SAD, require minor changes to take care of extra pointers for zones and subzones in the CSAs. Some other functions requiring minor changes are ipsec_spi_get_info() and ipsec_spigrp_get_info(), which are responsible for printing information in the /proc directory. Some extra information relating to CSAs (like zone lengths and subzones) needs to be displayed in this case.

The major changes are in the functions that deal with user level process interaction with the kernel in storing, extracting, and updating SAs in the SAD. ipsec_callback() deals with the interaction of the user level ipsec command with the ipsec device and takes care of adding, deleting, and updating SAs from the SAD in the kernel. It calls the functions mentioned above and also functions like tdb_init(), which facilitate updating the fields in the kernel SA using the information received from the user level process through the device interface. A major change in this function is the formation of the SAD structure as shown in Figure 12, which now includes chaining between CSA and Zonal SAs using zonemaps and subzones.

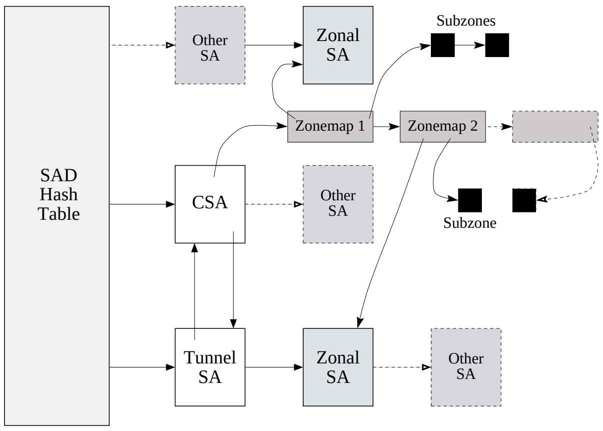

Figure 12: ML-IPsec storage structure in the SAD

The inbound and outbound processing of FreeS/WAN also requires major changes for ML-IPsec. The function ipsec_rcv() does the inbound processing for an IPsec packet received over the network. This function is responsible for getting the packet from the socket buffers, decapsulating it, extracting information from it to locate the SA in the kernel, doing authentication if necessary, doing decryption if necessary and forming the original IP Packet. It then pushes the packet up the stack to the IP layer for processing and/or forwarding onto the internal network. This function requires considerable changes to facilitate the processing in accordance with the new tdb structure and the new AH and ESP headers. The case of partial authentication/decryption (new in ML-IPsec, see Section 4.4.5) is also handled in this function.

The function ipsec_tunnel_start_xmit() is responsible for the outbound IPsec processing of an IP packet. The activities carried out by this function include: receiving packets from the IP layer, finding the corresponding SA and encapsulating route, tunneling the packet if necessary, and attaching the corresponding ESP and AH headers if necessary. The changes made to this function include differential calculation of the extra space required for putting zonal headers and trailers, buffer management on a per zone basis and authentication and encryption on a per zone basis. Compared to IPsec, this function has a much more complicated task because, instead of encapsulating and processing (encryption and/or authentication) on the whole payload, it has to take care of zone boundaries within the payload and provide differential processing according to the particular zonal SA. Futhermore, the encapsulation has to be exactly the same as in the IPsec case.

Finally, a number of changes are made in the shell scripts that interpret the configuration file and call the `ipsec` command with appropriate arguments for controlling the ML-IPsec functionality.

So far, we have omitted the PF_KEY interface [MMP98] part, which also operate on the `tdb` structures to facilitate communication between the user level utilities and the kernel level processes. The functions in this interface will ultimately need appropriate changes as part of our future work to add automatic keying support.

5.4 Changes in Configuration File

The `ipsec.conf` file specifies most configuration and control information for the FreeS/WAN IPsec subsystem. Its syntax needs to be modified for use in ML-IPsec implementation. This includes adding a new section called the ZONE section, and adding specification for six new parameters. Further, some parameters ought to be moved around from one section to another to match new SA structure. Most of these changes apply to the CONN section. The CONFIG section needs no change. Here we explain the modification in detail.

CONN Sections Two sets of new parameters have been added to this section and seven parameters have been moved to the new ZONE section. The semantics for the remaining parameters in this section stays the same as in original FreeS/WAN. The parameters that remain in this section are `type`, `auto`, `left`, `leftsubnet`, `leftnexthop`, `leftfirewall`, the corresponding "right" parameters, and the manual keying parameters – `spibase`, `espreplay_window`, and `ahreplay_window`. Other parameters that correspond to the automatic keying policy also remain unchanged, because the current ML-IPsec implementation only deals with manual keying.

The new parameters that we have added to this section are:

- *hop_number* This is an integer value which represents the number of intermediate nodes that will be doing ML-IPsec processing on the packets. This number does not include the source and the destination nodes.

- *hop[x]* This is the IP address of the intermediate node that will be doing ML-IPsec processing on the packet from left to right and vice-versa. There can be multiple instances of this parameter on separate lines in the file as long as x satisfies the condition $1 \leq x \leq n$ (where n is the value of the `hop_number` parameter) and is not reused. The value of each `hop[`x`]` parameter is also not to be reused. Also if `hop[`x`]` is present then `hop[`$x-1$`]` should also be present. The syntax of this value is the same as a `left` parameter value.

- *zones* This is a comma-separated list of `ZONE` section names that belong to this connection.

ZONE Sections This new section is designed to cohere with the ML-IPsec zone concept. The parameters that are moved from the CONN sections in original FreeS/WAN include `esp`, `espenckey`, `espauthkey`, `leftespspi`, `ah`, `ahkey` and `leftahspi`. The semantics and syntax of these parameters remain the same as before. The new parameters are:

- *hosts* This is a comma-separated list of the string "*hop[x]*" where x is as defined in the previous section. This defines the list of "hops" that have access to this zone of the IP packet. The IP addresses for the "hops" are defined in the above CONN section, but the list of "hops" that can access this zone are defined here. For example,

  hop[2], *hop[4]*, *hop[5]*

  means that the nodes whose IP addresses are defined by hop[2], hop[4] and hop[5] in the conn section have access to this zone of the packet.

- *zonebytes* This is a comma-separated list of integer ranges. It specifies the bytes of an IP packet that constitute a zone, like

  1-14, 20-26, 42-EOP

  (EOP denotes end-of-packet).

- *desig* This yes/no value signifies whether this section specifies a designated zone or not.

6 Conclusion

The end-to-end network security mechanisms such as IPsec and the rich network services such as those described in this paper are two fundamentally conflicting mechanisms. On the one hand, end-to-end security advocates the use of cryptography at the network layer to protect the payload over an untrustworthy internet. On the other hand, certain network services rely on intermediate nodes to perform "intelligent operations" based on the packet data type – the information encoded in a higher protocol layer. It is the need to find the right balance between the two mechanisms and to achieve the goals of both, that makes for a difficult engineering problem.

Our attempt to solve the problem is based on the layering architecture for network security protocols. The approach presented in this paper may already have the right mix to provide both security and extensibility in one unified platform. Certainly, we have shown that through protocol design and system implementation ML-IPsec can easily be added to an existing IPsec system and that its overhead is low. ML-IPsec has achieved the goal of granting trusted intermediate routers a secure, controlled, and limited access to selected portions of IP datagrams, while preserving the end-to-end security protection to user data. A similar system, which is based on the same layering principle described in this paper, has been implemented independently by University of Maryland [Kar99], although their implementation was based on an older version of FreeS/WAN and an older version of Linux kernel.

Our plan for future work includes an extension of IKE to support ML-IPsec. IKE is the key distribution protocol for IPsec, but we did not use it here because our current implementation uses manual keying only. It will be very important for ML-IPsec to be able to utilize automatic keying because it uses more keys, involves intermediate nodes, and requires a more complicated configuration than the original IPsec. The technical challenge will be finding the efficient mechanism needed for multi-party key distribution.

Web Site

The URL for the web site of this project is `http://www.wins.hrl.com/people/ygz/ml-ipsec` and it contains relevant documents, software releases (when they are ready), performance measure data for this implementation, and further developments for ML-IPsec.

Acknowledgments

The authors would like to thank the staff and students in the WINS group at HRL for many good suggestions and criticisms. In particular, we'd like to acknowledge the following students who have interned with us. Ilan Zohar (`ilanz@leland.stanford.edu`) of Stanford University, started looking into this problem in summer 1998. Praveen Kumar Gonugunta (`gonugunt@uiuc.edu`), a Ph.D. student at UIUC, started the first prototype implementation of ML-IPsec on an early version of FreeS/WAN (1.0, on Linux kernel 2.0.36), when he was an intern with us in summer 1999. Ashish Shah (`ashah@leland.stanford.edu`), a Ph.D. student at Stanford University who also interned with us in 1999, helped in the first prototype. In addition, the authors have also benefited from their e-mail discussions with participants in the IETF IPSEC working group and TF-ESP BOF.

References

[Bel96] S. Bellovin. Problem areas for the IP security protocols. In *Proceedings of the Sixth Usenix Unix Security Symposium*, San Jose, CA, July 1996.

[Bel99] S. Bellovin. Transport-friendly ESP (or layer violation for fun and profit). IETF-44 TF-ESP BOF, March 1999.

[BKGM00] J. Border, M. Kojo, J. Griner, and G. Montenegro. Performance enhancing proxies, March 2000. IETF draft, work in progress `draft-ietf-pilc-pep-02.txt`.

[BPSK97] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz. A comparison of mechanism for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, December 1997.

[DA99] T. Dierks and C. Allen. The TLS protocol, version 1.0, January 1999. IETF RFC 2246.

[FF99] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Transactions on Networking*, August 1999.

[FJ93] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, pages 397–413, August 1993.

[HC98] D. Harkins and D. Carrel. The Internet key exchange (IKE), November 1998. IETF RFC 2409.

[KA98a] S. Kent and R. Atkinson. IP authentication header, November 1998. IETF RFC 2402.

[KA98b] S. Kent and R. Atkinson. IP encapsulating security payload (ESP), November 1998. IETF RFC 2406.

[KA98c] S. Kent and R. Atkinson. Security architecture for the Internet protocol, November 1998. IETF RFC 2401.

[Kar99] M. Karir. IPSEC and the Internet, December 1999. Master Thesis, University of Maryland (Technical Report ISR-MS-99-14).

[MMP98] D. McDonald, C. Metz, and B. Phan. PF_KEY key management API, version 2, July 1998. IETF RFC 2367.

[NBB99] D. Nessett, B. Braden, and S. Bellovin. IPSEC: Friend or Foe. Panel discussion in Network and Distributed System Security Symposium (NDSS'99), February 1999.

[ZDRD97] Y. Zhang, D. DeLucia, B. Ryu, and S. Dao. Satellite communications in the global Internet: Issues, pitfalls, and potential. In *INET'97*, Kuala Lumpur, Malaysia, June 1997.