A Security Architecture for the Internet Protocol*

Pau–Chen Cheng† Juan A. Garay‡ Amir Herzberg§ Hugo Krawczyk¶

Abstract

In this paper we present the design, rationale and implementation of a security architecture for protecting the secrecy and integrity of Internet traffic at the IP layer. The design includes three components: (1) a security policy for determining when, where and how security measures are to be applied; (2) a key management protocol, called MKMP, for establishing shared secrets between communicating parties and meta-information prescribed by the security policy; and (3) the IP Security Protocol, as being standardized by the Internet Engineering Task Force, for applying security measures using information provided through the key management protocol.

Effectively, these three components together allow for the establishment of a secure channel between any two communicating systems over the Internet. This technology is a component of IBM's

*A preliminary version of this paper, "Design and Implementation of Modular Key Management Protocol and IP Secure Tunnel on AIX," was presented at the *5th USENIX UNIX Security Conference*, Salt Lake City, Utah, June 1995 [CGHK95].

†IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, U.S.A. E-mail: `pau@watson.ibm.com`.

‡IBM Thomas J. Watson Research Center. `garay@watson.ibm.com`.

§IBM Haifa Research Center, Israel. `amir@haifasc3.vnet.ibm.com`.

¶IBM Thomas J. Watson Research Center and Department of Electrical Engineering, Technion, Haifa 32000, Israel. `hugo@ee.technion.ac.il`.

Firewall product, and is now being ported to other IBM platforms.

1 Introduction

As Internet evolves from an academic/research network into a commercial network, more and more organizations and individuals will connect their internal networks and computers to it. Secrecy and integrity of the data transmitted over the Internet have become a primary concern, and cryptographic data encryption and authentication constitute the tools to address this concern. We subscribe to the view that the IP layer [Pos81] is a good place to secure the data being communicated. Reasons include:

- IP layer is at the choke point of Internet communication; it can capture all packets sent from the higher–layer protocols and applications, and all packets received by the lower–layer network protocols.
- By the very definition of IP, security provided at this layer is independent of lower–layer protocols.
- Security provided at this layer can be made transparent to the higher–layer protocols and applications.

Many application environments can benefit from security provided at the IP layer[1]. Figure 1 depicts some of them, including mobile–to–base communication, tele–commuting, and site–to–site or system–to–system communication.

Three components are needed in order to provide IP layer security through cryptographic means:

[1]A frequently asked question is why are both an IP layer security mechanism and a session layer security mechanism (such as SSL [FKK96] and TLS [DA97]) needed. In Section 7 we present a comparison of features between IP and session layer security.

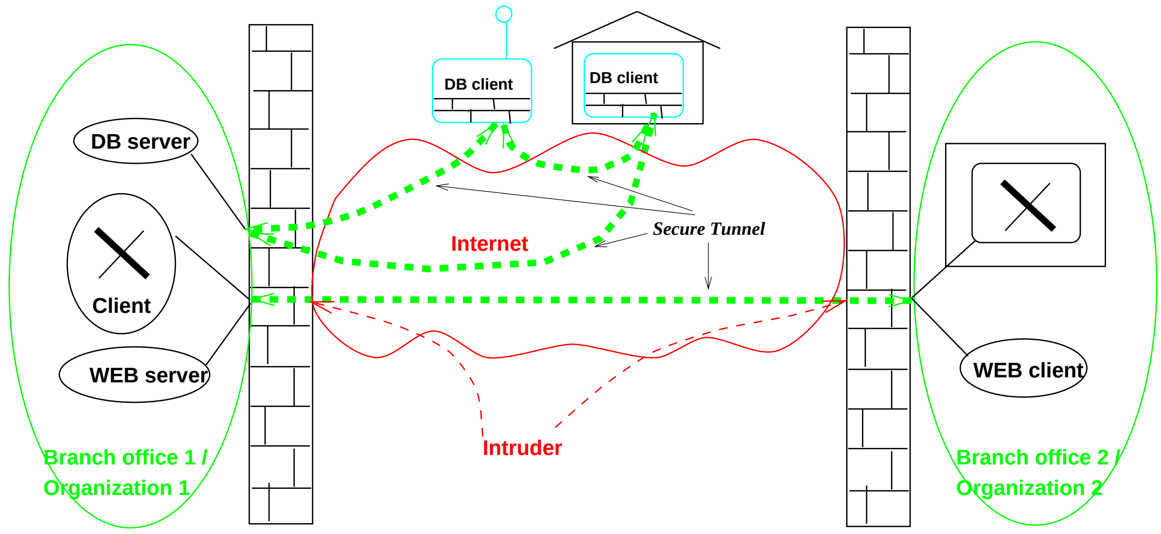

Figure 1: Some Applications of Secure Tunnels.

1. A *security policy*—to define the characteristics of the desired security. Such a policy specifies how the packets between two communicating systems must be protected.

2. A *key management protocol*—to establish and maintain the necessary information as prescribed by the security policy. Such information usually includes cryptographic algorithms, secret keys, and other parameters shared between two communicating systems.

3. A *protocol for security at the IP layer*—to protect IP packets as prescribed by the security policy, using information provided by the key management protocol. In our design, this protocol is the emerging Internet standard IP Security Protocol (IPSEC) [KA97c].

Also needed is a **concept** to thread the three components together; this is the *secure tunnel* concept discussed in section 2. In a nutshell, a secure tunnel is a secure passage for the IP packets transmitted between two communicating systems. The characteristics of a secure tunnel are defined by the security policy; the tunnel is established and maintained by the key management protocol, and the protection of IP packets is carried out through the IPSEC protocol.

This paper presents the protocols and system architecture which implement the above three components and the secure tunnel concept. At the heart of our security architecture are the *Modular Key Management Protocol* (*MKMP*) and the *IP Security Protocol.* MKMP provides the necessary information to establish a secure tunnel while IPSEC uses this information to provide data security by cryptographic means. MKMP is a set of protocols that we have designed for the management of the cryptographic keys used by secure tunnels. It provides secure mechanisms for the derivation and periodic refreshment of the cryptographic keys as required by the multiple cryptographic functions used within a secure tunnel.

IPSEC is an Internet standard from the IETF IPSEC Working Group. (Originally described in [KA97c, KA97b, KA97a], the IPSEC specifications are now under revision. See [KA97c, KA97b, KA97a] and follow-ups.) IPSEC is essentially an *encapsulation* protocol, namely, one that defines the syntax and semantics of placing one packet inside another. First, IPSEC protocol–specific operations are performed on an IP packet to protect its secrecy and/or integrity by cryptographic algorithms; then the output of the operations is appended to an IPSEC protocol header to form an IPSEC packet; finally, the IPSEC packet is placed inside an IP packet so it can be routed through the Internet.

We have prototyped our design, which became part of the new release of IBM's firewall product. In addition, our key management approach and techniques have had an impact on the Internet standard proposed by the IETF IPSEC Working Group [HC97].

An early proposal for providing IP layer security can be found in *swIPe* [IB94a, IB94b]. However, our work differs from swIPe in several aspects, in particular, in that: 1) the key management protocol is implemented and linked with network layer security protocol; and 2) the network layer security functionality is placed inside the kernel IP module and not in a network device driver.

The remainder of the paper is organized as follows. Sections 2 and 3 describe the notion of an IP Secure Tunnel and the MKMP protocol, respectively. Section 4 outlines the format of the IPSEC protocols. Section 5 presents the architecture and provides the details of our implementation, and Section 6 discusses its performance. We conclude the paper mentioning some related ongoing activities and directions for future work. A frequently asked question is why are both an IP layer security mechanism and a session layer security mechanism—such as SSL [FKK96] and TLS [DA97]—needed. In the Appendix we present a comparison of features between IP and session layer security.

2 IP Secure Tunnel

In this section we define the concept of a secure tunnel. We first make some clarifications on how the word "tunnel" is used in both the IPSEC arena and in this paper; then we discuss the concept of a *security association* on which the secure tunnel concept is based; finally we discuss the secure tunnel concept itself.

2.1 On the word "tunnel"

The word *tunnel* is widely used in the IPSEC arena. However, depending on the context, it could refer to several different but related concepts:

1. Conceptually, it refers to a *secure passage* (or channel) between two systems across the insecure Internet. This passage is a realization of the two system's *security policies*. In the context of IPSEC, a security policy establishes the specific requirements and *meta-characteristics* of a secure passage between two given systems. A passage's meta-characteristics usually include the identities or addresses of its two endpoints, the encapsulation mode, the cryptographic algorithms to be used, parameters for the algorithms (such as key lifetime and key size(s)),

etc. A security policy may also demand more than one secure passage between the two systems, each for a specific type of communication.

2. Implementation–wise, the word tunnel refers to a set of items of information shared between a secure passage's endpoints. This set enables the realization of a secure passage; it includes in particular the meta-characteristics and secret keys used by the cryptographic algorithms. In the IPSEC terminology, such a set is called a *security association*, or *SA*. Section 2.2 elaborates on SA's in more detail. However, as Section 2.3 will explain, an SA is *not* a secure tunnel but an incarnation of a secure tunnel during a particular time interval. An SA is usually created and maintained by a key management engine.

3. Finally, in IPSEC's standard terminology, tunnel refers to one of the two encapsulation modes defined by the IPSEC standard: tunnel mode and transport mode. Both modes can be used to construct a secure passage, although they provide slightly different protection. Section 4 presents a more detailed discussion on the IPSEC standard.

From now on, unless otherwise specified, we will use the words "tunnel" or "secure tunnel" to denote the secure passage concept or an instance of it.

2.2 Security association

A security association, or SA, is a set of items of information which, when shared between two communicating parties, enables the two parties to protect the communication in a desired way.

An IPSEC SA [KA97c] includes the following meta-characteristics:

- Destination ID/IP address: the intended receiver of IPSEC packets.
- Security protocol: the kind of security—integrity and/or secrecy—provided by the SA on the IP packets. Under the security protocol, a set of cryptographic algorithms (called *transforms*

in IPSEC) and its parameters, such as key life-time and key size, are specified.

- Secret keys: the keys to be used by the cryptographic transforms.
- Encapsulation mode: indicating which part(s) of the IP packet will be protected by the SA.
- *Security Parameter Index* (*SPI*): the identifier of the SA. On a given system, the SPI should be unique with respect to the SA's destination address so that the pair (*destination address*, *SPI*) uniquely identifies an SA. An IPSEC packet constructed according to an SA carries the SA's SPI so that the destination will know how to process the packet.

The SA's encapsulation mode, the security protocol, and the cryptographic transforms define the operations to be performed on a packet. Section 4 discusses these operations in more detail.

An IPSEC SA is *uni–directional* in the sense that the information in it should only be used to construct and process IPSEC packets intended for the destination address in the SA. However, a secure tunnel is bi–directional. Thus, a pair of IPSEC SA's, one for in–bound traffic and another for out–bound traffic, are needed for each of the end–points of a secure tunnel. Both MKMP and ISAKMP/OAKLEY, the new standard for IPSEC key management [MSST97, HC97], always generate such pairs. The two SA's in a pair share the same meta-characteristics but have different keys. Our design treats such a pair as a *bi–directional* SA.

IPSEC also has the notion of an *SA bundle*. An SA bundle is a pair of IPSEC SA's with different security protocols and transforms combined together to provide the desired security. For example, if one SA protects the integrity of packets and the other provides secrecy, then the two SA's can be combined to protect both integrity and secrecy. Note that the two SA's in a bundle are shared between the same two systems and the destination addresses of the two SA's must be the same. Section 4 discusses when an SA bundle may be needed. Like an IPSEC SA, an IPSEC SA bundle is uni–directional so a pair of SA bundles is needed for bi–directional security. Both MKMP and ISAKMP/OAKLEY can generate pairs of SA bundles. Our design treats such pairs as a bi–directional SA bundle.

From now on, unless otherwise specified, we will use the words "security association," or "SA" to refer to a bi–directional SA or SA bundle which provides the desired security. We will use the words "IPSEC security association," or "IPSEC SA" to denote a uni–directional SA as specified in [KA97c].

2.3 Secure tunnel

This section explains what we mean by a "secure tunnel" in this paper, and how it differs from a security association. We arrive at the concept through an example of the series of steps that are needed to establish secure communication between two systems.

Consider the case of two sites/systems A and B which are connected to the Internet through two systems X and Y, respectively, where A and X, and B and Y may or may not be the same. As a first step towards having secure communication, A and B negotiate a (hypothetical) set of rules as the policy for communication between them:[2]

Policy 1 :

Rule 1: Packets of type 1 must go through a secure passage between X and Y and the meta-characteristics of this passage are in set 1.

Rule 2: Packets of type 2 must go through a secure passage between X and Y and the meta-characteristics of this passage are in set 2.

Here the type of a packet is usually defined by the packet's source and destination addresses, the transport layer protocol (e.g., TCP or UDP), and/or the port numbers/types, etc.

While at first sight Policy 1 seems reasonable, another piece of information is needed, namely, the keys to be used with the cryptographic algorithms in sets 1 and 2 in order to enforce the policy.

[2] See [CB94] for a clear explanation of the necessity of a good, concise security policy.

In other words, the references to sets 1 and 2 must be translated into references to SA's. Rewriting Policy 1 yields:

Policy 2:

Rule 1: Packets of type 1 must go through a secure passage between X and Y using SA 1.

Rule 2: Packets of type 2 must go through a secure passage between X and Y using SA 2.

Policy 2 is already usable, but still not satisfactory from a cryptographic standpoint. The reason is that the keys in a security association should be changed frequently in oreder to defeat cryptanalysis or brute–force attacks. One first option could be changing the keys in an SA frequently but keeping everything else the same. It turns out that this is not a good solution either. In such a solution, the same reference to an SA must be reused for a key and its replacement. In practice, the life of a key and its replacement must have some time overlap in order to avoid a disruption of communication caused by key expiration. Using the same reference for different keys will cause false security alarms because the wrong key is used to check the integrity of a message, and/or a significant processing overhead because both the old key and the new key have to be tried.

Therefore, in our design, a security association expires as its keys expire. This design decision makes Policy 2 very impractical because a stable security policy cannot be defined in terms of short–lived SA's. To resolve the conflict between the need for a stable, relatively long–term security policy and the need for changing keys frequently, we took a further look at a security association and concluded that its meta-characteristics (i.e., the SA minus its keys) are relatively stable and long–term. In fact, the strength of protection usually does not depend on the exact values of the keys, but rather on their sizes, life time, the cryptographic algorithms, etc. With this conclusion, we are now ready to introduce the *secure tunnel* concept:

A *secure tunnel* is a secure passage between two systems across the insecure Internet. It has a set of meta-characteristics determined by the security policies of the two systems. Such a secure tunnel is realized by a series of SA's which change with time; these SA's share the same meta-characteristics but have different cryptographic keys.

Each of these SA's is the replacement of its immediate predecessor and can be considered an incarnation of the secure tunnel during the SA's life time.

With the secure tunnel concept, the final form of the security policy becomes:

Policy 3 :

Rule 1: Packets of type 1 must go through secure tunnel 1 between X and Y.

Rule 2 : Packets of type 2 must go through secure tunnel 2 between X and Y.

Our implementation assigns each secure tunnel an ID. Each SA will include the ID of the secure tunnel to which it belongs; this inclusion of the secure tunnel ID provides a two–way linkage between a secure tunnel and its current incarnation. Sections 5 and 5.1 give more details on these aspects of the implementation.

From now on, if a packet is encapsulated according to the information in a secure tunnel, we say that the packet *goes* or *comes through the tunnel.*

3 Modular Key Management Protocol

The basic goal of a key management scheme is to provide the two communicating parties with a common, "freshly" shared cryptographic key that is known to these parties only. In general, a typical key management scheme will achieve that in two phases. One in which a "master key" is shared between the parties, and the second, in which the already-shared master key is used for the derivation, sharing and/or refreshment of additional "session keys." The term "session key" is used here to denote short-lived keys (say, in the range of seconds or minutes); it does not imply or require a session-based communication model. The frequent change in the values of these keys usually requires a fast way to generate and share the keys as to reduce the key exchange overhead in terms of computation and communication. The term "master key" is used to denote keys with a longer life period than a session key (say, a range of hours), and then they may allow for more time-consuming procedures for their generation and sharing.

The split into these two phases is not mandatory, and in fact there are systems which do not establish (at least explicitly) this separation. However, we argue here that for most scenarios this explicit separation has a significant methodological and design value. In particular, we advocate the separation of these functions into two modules: one for the sharing of the master key, and one for the key management "below" the shared master key. Thus, our approach to key management is *hierarchical*, namely, session keys are derived from the shared master keys, and in turn, the master keys are derived using any of the well-established key exchange methods: public key exchange, key distribution centers (e.g., Kerberos [KN93]), and manual key installation. (See [Kra96] for a further elaboration of these various scenarios and trust models.) Our hierarchical approach is illustrated in Figure 2.

In this paper we concentrate on the basic mechanisms for the derivation and management of the session keys (i.e., the "lower" module). Our protocol can be extended to support the derivation of master keys from public keys; however, the description of these particular mechanisms is beyond the scope of this paper. We refer to SKEME [Kra96] for a design of such extensions as well as for a more comprehensive treatment of security requirements and mechanisms for key management. We now turn to the description of the session key protocol.

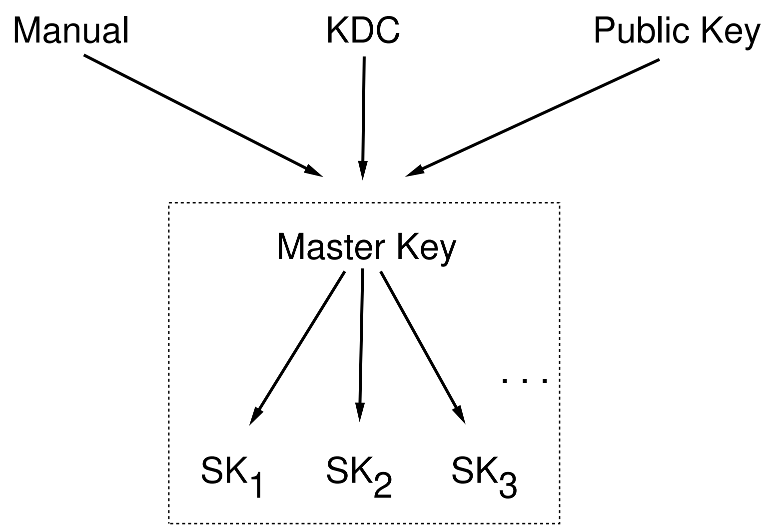

Figure 2: Hierarchical approach to key management.

3.1 Session key negotiation protocol

The basic goal of a key negotiation protocol is to provide both intervening IP nodes with a shared *session* key. The key is then used for data authentication and encryption, thus allowing the establishment of a secure tunnel between the two parties.

We now list some of the goals and requirements that our session key protocol is required to support.

SECURITY HANDSHAKES. A basic assumption that we make in this paper is that parties are able to establish periodic two-way (or interactive) communication, as opposed to just sending information in a one-way mode. This *interactive* mode of key refreshment allows for the parties to have periodic handshakes whereby cryptographic information is refreshed and verified by both parties simultaneously. This has a significant impact on the security aspects of the protocol, as pointed out below.

SECRECY AND AUTHENTICITY. The protocol needs to guarantee the intervening parties that only the intended party learns the key exchanged, and that this key is "fresh," random, and unique.

Secrecy and authenticity of the exchanged key need to be protected against passive (eavesdroppers) and active (man-in-the-middle) attackers, and these properties be guaranteed for as long as the underlying cryptographic functions in use are secure against such adversaries.

EFFICIENCY. Another important goal of a key negotiation protocol is efficiency, namely, to keep both the number of messages exchanged between the parties and the computational overhead (e.g., the number of expensive public key operations) to a minimum. It is worth noting that by having a highly efficient method for session key renewal, the need for frequent master key update, which is usually computationally intensive, is alleviated.

FORWARD SECRECY. Another consideration is the level of security provided by the protocol. One desired property of the session keys is that of *forward secrecy*: [DvOW92]: even if an attacker is eventually able to derive the key for one session, then past and future session (and, of course, master) keys are not compromised. Our protocol achieves this level of security for session keys. We remark that achieving such security even in the case of the compromise of a master key requires relatively costly mechanisms such as a Diffie-Hellman exchange for the sharing of the master keys. However, this cost is usually compensated by the long-lived nature of these keys.

SIMPLICITY. Finally, simplicity and being amenable to analysis and proof are important features of any cryptographic protocol.

Figure 3 shows the "cryptographic skeleton" of the session key negotiation protocol, including only the relevant (i.e., related to authentication) information. In the full implementation of the protocol further information, such as cryptographic parameters, negotiation attributes, key identifiers, etc., are transmitted and authenticated between the parties, using the same two communication flows. There are two parties, a sender S and a receiver R. S is the party that initiates the protocol. In the presentation of the protocol we use the following terminology:

ID_X: The identity of party X.

N_X: A nonce (i.e., a random number) chosen by party X.

K: The currently shared master key. In our case it represents a pair of keys K_1, K_2.

MAC_k: A Message Authentication Code (or integrity check function) which is applied to a piece of information for authentication using a secret key k. Examples include block ciphers (e.g. DES, in CBC-MAC mode [Ass86]), and key-ed cryptographic hash functions (e.g., keyed-MD5 [Tsu92], HMAC [BCK96],etc.).

f_k: A pseudorandom function with key k. Roughly speaking, pseudorandom functions [GGM86] are characterized by the pseudorandomness of their output, namely, each bit in the output of the function is unpredictable if k is unknown. We use pseudorandom functions primarily for the derivation or expansion of key material given an initial key k. The examples of MAC functions given above are also commonly believed to be pseudorandom functions.

SK: The session key, outcome of the protocol.

As Figure 3 shows, the protocol consists of two messages (or flows). It is assumed that S and R already share two master keys, K_1 and K_2, as well as the nonce N_R, exchanged in a previous run of the protocol. New nonces are exchanged and authenticated in each run of the protocol under the MAC function keyed with the master key K_1. There is no explicit transmission of a cryptographic key from one party to the other; instead, the shared session key is computed by both parties on the basis of the master key K_2 and the fresh information exchanged in the protocol. Upon (successful) termination of the protocol, parties S and R have exchanged a session key SK, besides mutually authenticating each other. (We note that the same protocol allows for periodic key refreshments within a session.)

We now turn to argue how the protocol satisfies the requirements listed above. We omit from this paper any formal proofs. However, we note that the protocol presented here is structured, simple, and thus easier to analyze, and hence rigorous methods similar to those of [BGH+93, BR93] can be used to establish the protocol's desired security properties.

The protocol is essentially a "cryptographic handshake" that allows the parties to directly authenticate each other. The challenge provided by the nonce guarantees the freshness of the authentication, and avoids the so-called *replay* attacks. The binding between master key, nonces and identities provides security even in the case where several such protocols are run in parallel, and so it prevents *interleaving attacks* [BGH+93]. We note that keeping a shared nonce (N_R) between runs of the protocol is not essential; it can be replaced by the use of time stamps (at the expense of requiring a secure clock synchronization) or by adding an extra flow to the protocol (at the expense of performance). The nonce also serves the purpose of alleviating the effect of the so-called *clogging*, or "denial of service" attacks, as it allows for rapid detection and dumping of maliciously replayed traffic. In any case, the nonces do not require any secrecy, that is, they can be transmitted in the clear.

Also notice, as mentioned before, that the session key SK is not explicitly transmitted. This avoids the need to encrypt this key as well as the need to explicitly authenticate it. The authenticity and freshness of SK are derived from the authenticity and freshness of the expression T. Even if an adversary succeeds in replaying an old message from S to R (for example, in case a time-stamp is used instead of the nonce N_S) the freshness of SK is guaranteed by the incorporation of N'_R, chosen by R, into the MAC expression T from which SK is derived. Thus, the session keys that are derived are fresh, and independent from the past session keys (the only existing dependence is to the current shared master key).

Regarding efficiency, the protocol requires only two message flows and no expensive operations (e.g., exponentiations) at all, given that the parties already share a master key; only efficient symmetric-key techniques are used. The forward secrecy of the session key SK follows from the unpredictability and one-wayness properties of pseudorandom functions and our particular application of these functions to derive SK.

We stress that usually more than one key is required for a single security association. For example, one needs different keys for the encryption and for the authentication of information; in some cases the keys are used uni-directionally; etc.. The derivation of more than one key using the protocol of Figure 3 is straightforward: instead of a single application of $f_{K_2}(T)$ one can apply $f_{K_2}(\text{"transform-id"}, T)$ for each required key, where "transform-id" is a unique identifier that identifies the algorithm for which the key is to be used (e.g., DES-CBC), the key length, the direction of the communication (e.g., for message authentication from S to R only), etc.

Finally, we note that one can replace the use of the MAC function in the above protocol with the pseudorandom function f. The latter would provide the authentication properties of a message authentication function (these properties follow from the strong unpredictability requirements of a pseudorandom function). In this case, one could use only one master key, K, to key both the uses of f_K as a MAC as well as for its uses for key derivation.

4 The IPSEC Protocols

This section discusses the Internet IPSEC protocols presented in [KA97c], [KA97b] and [KA97a]. These protocols are not in their final form yet, though they are converging towards a stable specification. These documents will eventually become standard RFC's ("Request For Comments"); we refer the reader to them for more details.[3] Section 2.2 already discusses the concept of an

[3] Information on the development of this standard can be found in the IPSEC home page, `http://www.ietf.org/html.charters/ipsec-charter.html`, and the IPSEC mailing list `ipsec@tis.com`.

IPSEC security association, which is essential to the protocols; now we will discuss the syntax and semantics of an IPSEC packet by explaining how an IP packet carrying an IPSEC packet is constructed.

An IPSEC packet is constructed according to the information in an IPSEC SA, and its format depends on the security protocol in that SA. The security protocol can be either *Encapsulating Security Payload* (*ESP*) [KA97b] or *Authentication Header* (*AH*) [KA97a]. ESP protects a packet's secrecy and optionally integrity, while AH protects only integrity. Secrecy is protected through encryption, and integrity is protected by a message authentication code (MAC) function. The specific algorithms and keys (and related parameters) for computing these cryptographic functions are specified in the IPSEC SA. The default IPSEC encryption algorithm is DES in CBC mode; the default algorithm for IPSEC message authentication code is HMAC–MD5 [OG97, BCK96, KBC97].

The high–level layout of an *ESP packet* is shown in Figure 4 and the high–level layout of an *AH packet* is shown in Figure 5. An IPSEC packet is either an ESP packet or an AH packet.

Although the ESP protocol allows for the application of encryption without integrity protection, this mode of operation is strongly discouraged. Except for very rare exceptions, a packet requiring secrecy will also require integrity protection. Moreover, as demonstrated in [Bel96], the lack of integrity protection may lead to a loss of confidentiality even if a secure encryption function is applied.

Another measure for integrity protection is providing with *anti–replay* defense. This type of defense prevents an attacker from re-injecting previously authenticated IP packets into the communication stream. More precisely, anti–replay measures make it possible to detect such "replayed" packets. To implement these measures, the AH and ESP protocols add to their headers a *sequence number* field. This field is meaningful only when the packet is authenticated via a MAC function. The value of the sequence number is maintained in an IPSEC SA. A sequence number is 32 bits long and always starts from 1; it is incremented by 1 each time an IPSEC packet is constructed according to the SA, and it is not allowed to wrap around. The receiver of an IPSEC packet (the "destination" in an IPSEC SA) uses a locally–defined *sliding window* to keep track of which sequence numbers have been already received. It rejects those packets that fall outside that window or that carry an already-seen sequence number.

While the security protocol in an IPSEC SA determines the format of the packet, the encapsulation mode determines which part of an IP packet is protected. There are two encapsulation modes:

1. *Tunnel mode*: this mode protects the entire IP packet. The entire IP packet is encapsulated in an IPSEC packet and a new IP protocol header is constructed and attached at the beginning of the IPSEC packet to form a new IP packet. The source and destination addresses may or may not be the same as those in the encapsulated IP packets. This mode is typically used for a secure tunnel between two firewalls, or between a firewall and a remote system, i.e., whenever either of the two communicating systems is not an endpoint of the tunnel.[4] The source and destination addresses in the new IP header are the addresses of the tunnel's endpoints.

2. *Transport mode*: this mode only protects the transport–layer packet (such as a TCP or a UDP packet) inside an IP packet. In this mode the IP protocol header is first separated from the transport–layer packet, then the transport–layer packet is encapsulated in an IPSEC packet, then the IP protocol header is attached to the IPSEC packet to form a new IP packet, and finally the *length*, *protocol* and *header checksum* fields in the IP protocol header are modified accordingly. The source and destination addresses in the IP protocol header remain unchanged. This mode is used when the endpoints of the secure tunnel are the two

[4] In other words, in the scenario in Section 2.3, either A and X or B and Y are not the same.

communicating systems.

There are two exceptions to the above description of the two modes if the security protocol is AH :

1. In the case of AH, the integrity protection extends to part of the “prepended” IP header. More details are given below when describing the procedure to construct an AH packet.
2. An ESP packet can be encapsulated in an AH packet. This may happen when an SA bundle is used to protect secrecy and integrity. Conceptually, this case can be thought of as first constructing an IP packet with an ESP packet inside, and then using this IP packet as the input to the construction of an AH packet. More details are given towards the end of this section when the use of AH and SA bundles are discussed.

4.1 Constructing the IPSEC packet

With the knowledge of the security protocol, the transform and the encapsulation mode, one can determine how to assemble an IPSEC packet from the IP packet. We fisrt describe the procedure to construct an ESP packet:

1. Construct an ESP protocol header; this includes the SPI of the IPSEC SA and the sequence number.
2. Append the to–be–encapsulated packet to the ESP header.
3. Construct an *ESP trailer* and append it to the to–be–encapsulated packet. The trailer serves the following purposes:
   - It contains the protocol number of the to–be–encapsulated packet, such as the TCP protocol number (6) or the IP–IN–IP protocol number (4, if tunnel mode is used).

- If a block cipher (e.g., DES) is used for encryption, then the trailer contains some padding bytes so that the size of the encapsulated packet, the padding bytes and the protocol number (1 byte) will be an integral multiple of the cipher's block size.

4. Encrypt the to–be–encapsulated packet and the trailer and append the ciphertext (namely the output of the encryption) to the ESP header.

5. If, as recommended, integrity protection is provided, compute the message authentication code over the ESP header and the ciphertext, and append the output of the MAC computation to the ciphertext. This output is indicated by the "MAC" field in Figure 4.

The ESP header, the ciphertext and, optionally, the MAC field form the ESP packet.

The procedure to construct an AH packet is as follows:

1. Construct an AH protocol header; this includes the SPI of the IPSEC SA, the sequence number, the protocol number of the to–be–encapsulated packet, and zero–filled bytes to hold the message authentication function's output.

2. Append the to–be–encapsulated packet to the AH header.

3. Prepend the IP protocol header to the AH header.

4. Compute the message authentication code over the IP header, AH header and the to–be–encapsulated packet; place the output in the AH header. The output is indicated by the word "MAC" inside the "AH Hdr" box in Figure 5.

The AH header and the packet attached to it form an AH packet. One special feature of the AH protocol is that the computation of the message authentication code includes the IP protocol header of the IP packet which carries the AH packet. However, not all fields in the IP protocol header are covered. Prior to the computation, a field in the IP protocol header is zero–ed if its content may change en-route. Such fields are called *mutable* in [KA97a]. After the computation, the values of these mutable fields must be restored. The efficient construction of an AH packet and restoration of the mutable fields is a tricky implementation issue which we do not discuss here.

Although ESP can provide both secrecy and integrity protection, AH is still needed for the following reasons :

- If secrecy protection is not needed or prohibited by law, AH can provide integrity protection without the cost of encryption.
- AH's integrity protection covers part of the IP protocol header, while ESP's does not. Whether this difference in coverage is needed depends on the specific operational security requirements. (For example, a military application may want to protect the sensitivity labels included in the option fields in an IP header.) If both the difference in coverage and secrecy protection are needed, then an IPSEC SA bundle (see section 2.2) can be used and the result is shown in Figure 6. In this figure an ESP packet is encapsulated inside an AH packet and the AH packet is encapsulated inside an IP packet; this layout implies that the ESP packet is constructed first. The sequence number field in the AH header enables the receiver to detect a replay attack early, therefore saving the decryption operation.

5 System Architecture

Figure 7 shows the system architecture of our implementation on IBM's AIX operating system. It consists of three parts. On the right is the **Key Management** part, including the MKMP engine, the *tunnel interface* and the *tunnel cache*. On the left is the **IPSEC policy** part, including the *policy engine*, the *policy interface* and the *policy cache*. In the middle is the **IPSEC encapsulation** part, including the *IPSEC* engine and the *crypto* engine. There is also an IPSEC–enabled IP engine.

The MKMP engine establishes and manages the secure tunnels, and stores these tunnel's security associations (Section 2.3) into the tunnel cache through the tunnel interface. The policy engine allows an administrator to define and to manage IPSEC policies; it translates a human–readable policy into binary form and stores the binary form into the policy cache through the policy interface. For each in–bound and out–bound IP packet, the IP engine queries the policy cache on what actions to take on the packet. For an in–bound packet, the action may be either to *permit* or *reject* the packet, a decision which may be partially based on what tunnel the packet has gone through. For an out–bound packet, the action may also be to *encapsulate*, meaning that the packet should go through a secure tunnel designated by the policy. The IPSEC encapsulation part is invoked by the IP engine to perform IPSEC encapsulation and decapsulation on IPSEC packets.

We stress that an important principle followed by our design is to decouple the IPSEC encapsulation part from the key management part; doing so allows us to link different key management protocols—and their underlying distributed security infrastructures—with the IPSEC encapsulation protocol. The linkages between the two parts are ID's of secure tunnels. The key management part generates SA's with their tunnel ID's inside. The IPSEC encapsulation part can use a secure tunnel ID to find the SA which is the current incarnation of the tunnel; it can also use the destination address and SPI in a received packet to find the corresponding SA and therefore the secure tunnel to which the SA belongs.

Another important design principle is to decouple the policy part from the key management part and the IPSEC encapsulation part. We believe that it should be up to the policy part to decide what to do with a packet, while the other two parts should implement the decision regardless of how the decision is made. Doing so allows us to accommodate different policy models and switch to better decision–making mechanisms in the future. The linkages between the policy part and the other parts are also secure tunnels; Section 5.1 below provides more details.

Figures 8 and 9 show the flowcharts of IP/IPSEC output and input processing, respectively. IPSEC processing was introduced into the so-called "vanilla IP" processing by adding *function call hooks* into the original AIX IP code. The blocks in dashed line indicate IPSEC processing. On output processing, a packet passes through the output packet filter [DBC92, CB94] implemented by the policy cache. The outcome of output packet filtering can be: (1) *permit*: proceed with vanilla IP processing, (2) *reject*: throw away the packet and (optionally) log the event, or (3) *encapsulate*: perform IPSEC encapsulation on the packet based on the specification of a policy–designated secure tunnel, and then transmit the resultant new IP packet.

On input processing, a packet first passes through the input packet filter implemented by the policy cache. The result of input packet filtering can be: (1) *permit*: proceed with vanilla IP processing, or (2) *reject*: throw away the packet and (optionally) log the event. If the result is "permit," then after some vanilla IP processing, the "protocol" field in the IP header is examined. If its value is either ESP or AH, which indicates that the payload is IPSEC-encapsulated, then the packet is sent through the IPSEC decapsulation process implemented by the IPSEC encapsulation part; otherwise, the packet is passed directly up to the transport layer protocol. In turn, the result of IPSEC decapsulation consists of two parts: (1) the IP packet that was encapsulated, and (2) the ID of the secure tunnel through which the packet has come; this ID indicates in what way the packet has been protected. This result is fed back to the beginning of IP input processing and passes through the input filter again. Now the filter can make decisions based on the packet and the way it was protected. For example, the policy can state "accept the packet only if it came through a certain tunnel." We now describe the parts of the architecture in more detail.

5.1 The IPSEC policy part

Conceptually, an IPSEC policy consists of packet filtering rules. These rules determine whether an IP packet should be received (i.e., passed up to the transport layer protocol) or transmitted. Basically, the rules are put in a list, and the first rule which matches the IP packet's relevant information determines the decision. A rule is composed of several fields, including the action to be taken (permit/deny/encapsulate), the source address and address mask, the destination address and address mask, the transport protocol, the port number/message type (if applicable), the direction (in–bound/out–bound), and the secure tunnel ID. For an out–bound packet, a non–zero tunnel ID specifies that the packet should be encapsulated by the IPSEC encapsulation defined by the tunnel—i.e, the packet should go through the tunnel. For an in–bound packet, a non–zero tunnel ID says that the packet should be protected by the IPSEC encapsulation designated by the tunnel ID—the packet should have come through the tunnel.

We have realized the policy part with the following components:

- *Administration Tools*: these are applications running in the user space which allow the administration to create and modify IPSEC policies. One important function of the tools is to translate human–friendly policy statements into binary form, and put the binary form into the kernel *policy cache* through the *policy interface*.

- *Policy Interface*: this is a pseudo-device driver acting as the interface between the administration tools and the policy cache. It allows the administration to change/view the kernel policy cache in a *whole–sale* manner.

- *Policy Cache*: this is a repository for IPSEC policies in the kernel, used by the IP module. The policy cache exports an IP packet filter interface to the IP module. The interface is divided into *input filtering* and *output filtering*. The input to the interface is an IP packet.

For input filtering, the output is a simple permit/deny answer. For output filtering, the output is a permit/deny/encapsulate answer. In the case of encapsulate, the packet is encapsulated according to the current SA in the corresponding tunnel.

In our implementation the policy cache can only be changed *as a whole*. If the administration tool wants to add one rule to the policy, it will read the whole cache into a copy in the user space, add the rule to the copy, and write the modified copy back to the policy interface. The policy interface will lock the policy cache, do a pointer–swap operation to make the cache point to the new copy, and then unlock and delete the old copy. The reason for this whole–sale manner of operation is performance. Since the processing of every packet involves searching and matching the policy cache, a per–rule lock would definitely degrade the performance. In fact, our experience shows that the number of locking operations on the policy cache should be kept to a minimal.

5.2 The key management part

A brief description of the key management part is given at the beginning of Section 5. This subsection discusses the MKMP engine and tunnel administration tools shown in Figure 7. More details can be found in [CGHK95].

5.2.1 The MKMP engine

The MKMP engine establishes and manages secure tunnels by creating and refreshing SA's within the tunnels; it also caches a secure tunnel's SA's in the tunnel cache through the tunnel interface. The engine is divided into two modules: the session key engine and the master key engine, which we now turn to describe in detail.

The master key engine negotiates the master keys, the first shared nonce, and the meta-characteristics of a secure tunnel, and passes the information to the session key engine in a data structure called *master key context*. The master key is actually a pair of keys: one key is used to authenticate the messages from the session key protocol, while the other is used as an input to the pseudo–random function in order to derive session keys (see Figure 3). The master key engine can be instantiated in several ways as: it can be a simple user-level command which implements the manual distribution of master keys, or it may use a Key Distribution Center–based protocol, such as Kerberos or NetSP [BGH$^+$95], or it may be a process which derives master keys using public key cryptography, such as ISAKMP/OAKLEY [MSST97, HC97]. At present, we have implemented two master key engines: (1) manual parameter negotiation and key distribution, and (2) manual parameter negotiation and Diffie–Hellman key exchange [DH76], authenticated by a pre–shared secret. We are currently in the process of prototyping the ISAKMP/OAKLEY protocol.

The session key engine implements the session key protocol described in Section 3. It accepts a master key context from the master key engine and uses this context to establish and maintain a secure tunnel with the session key engine on the other end of the tunnel. A run of the session key protocol derives two session keys, one for IP packet encryption and the other for authentication. The ID's of the encryption and authentication algorithms are used to index the pseudo–random functions to generate different keys. The session key engine refreshes a key before it expires to ensure un–interrupted secure communication. The length of the overlapping period between the old and new keys is in the master key context. For each key refreshment, the session key engine creates new SA's with new SPI's and keys, but keeps all other information unchanged.

Figure 10 shows the architecture of the session key engine and the master key engine, and their relations to other parts of the operating system.

The design goals of this part of the architecture are modularity, flexibility and portability. Since we envision that the session key engine may interoperate with many master key engines of different types, the session key engine is an independent process separated from the master key engines. A session key context is sent in an UDP message from the master key engine to the session key engine. In order to prevent an adversary from sending a bogus master key context to the session key engine, the content of the UDP message is authenticated by a secret key which is pre–shared between the session key engine and the master key engine. In order to provide portability across different platforms, we have implemented an OS dependency library. This library provides OS–independent API's, which in turn provide the following services:

- **Secure communication:** for network communication, encryption and authentication of messages from the master key engine to the session key engine;
- **timer/alarm:** for retransmission and key refreshment/deletion;
- **asynchronous wait:** for capturing asynchronous events (e.g, time–out events and receipts of messages); and
- **tunnel caching:** for the caching of security associations.

5.2.2 The tunnel administration tools

Briefly, the tunnel administration tools are a set of applications to examine and delete security associations in the tunnel cache. they also provide an interface for inserting manually created security associations into the tunnel cache. These tools have been proven to be invaluable for the debugging of IPSEC protocols and for diagnosing secure tunnel configurations.

5.3 The IPSEC Encapsulation part

Following the discussion in Section 4, the IPSEC encapsulation process can be divided into two main steps:

1. *Packaging*: which handles all the IP headers, IPSEC headers and ESP trailers, and invokes the necessary crypto operations; and

2. *cryptographic operation*: which performs the actual cryptographic computation.

Accordingly, the IPSEC encapsulation part is divided into the IPSEC engine and the crypto engine. This division of functionality has the following advantages:

- Implementations of cryptographic algorithms are decoupled from the details of IPSEC syntax and semantics. Thus, new algorithms and improved implementations can be easily introduced.
- It is much easier to keep up with the changes in IPSEC syntax and semantics. We have actually updated the IPSEC engine a few times as the IPSEC standards evolved; the changes were all within the IPSEC engine and no other parts were affected.

We now describe the two engines in detail.

5.3.1 The IPSEC engine

The IPSEC engine is responsible for the packaging part of the IPSEC encapsulation and decapsulation processes.

For encapsulation, the engine's inputs are an IP packet from the IP engine and a secure tunnel ID from the policy cache. The IPSEC engine will find the tunnel's current SA through the tunnel ID and encapsulate the input packet using the information in the SA. The encapsulation procedure is discussed in Section 4; the output of the procedure is an IPSEC packet inside an IP packet.

For decapsulation, the engine's input is a received IP packet with an IPSEC packet inside; this IP packet should be the output of its sender's encapsulation procedure. The IPSEC engine will find the SA corresponding to the SPI in the IPSEC packet and the IP packet's destination address. The decapsulation procedure is the inverse of the encapsulation procedure. If the SA indicates that the integrity of the message should be protected, the decapsulation procedure will first check if the received IP packet is a replay and then verify the MAC inside the received IPSEC packet. The received packet will be rejected if it is a replay or if the MAC verification fails. Other error conditions, such as a failure of decryption, may also happen during decapsulation. All error conditions will cause the received packet being rejected.

5.3.2 The crypto engine

The crypto engine is divided into a lower and an upper layer. The lower–layer module implements and exports a specific cryptographic algorithm. For example, there is one module implementing DES–CBC and another module implementing HMAC–MD5. The lower–layer module is identified by an implementation–dependent integer ID of the specific cryptographic algorithm it implements, and exports one of two generic interfaces, depending on whether it implements encryption or message authentication.

The upper layer is also a framework which holds different lower–layer modules. However, the crypto upper layer does not hide the interfaces exported by the lower–layer. A reference to a lower–layer crypto module can be acquired through a *search* function provided by the upper–layer, with the ID of the crypto transform as the search key.

All the lower-layer modules in the IPSEC engine and the crypto engine are optimized to use `mbuf chains` [LJFK86] as I/O buffers.

6 Performance

The performance of our implementation is very similar to that reported in our early work [CGHK95]. Although the code has been improved and modified to fit the new standard, the dominant factor of performance, namely, the cost of the cryptographic operations, remains unchanged. Since the same cryptographic operations are still used by the new standard, the performance stays about the same. The reader is referred to [CGHK95] for more details.

Currently, work is underway to produce handcrafted, high–performance cryptographic code. We are also investigating the use of cryptographic hardware.

7 On–going and Future Work

We are currently implementing a key management engine based on the emerging Internet standard key management protocols, ISAKMP [MSST97] and OAKLEY [HC97, Orm97]. OAKLEY has been strongly influenced by the SKEME protocol [Kra96], which in turn is a natural extension of the MKMP protocol presented here.

As IPSEC and other security protocols, such as TLS [DA97] become ubiquitous, we believe it is important to refine the policy part in order to provide a unified, easy–to–use administration/user interface so that different security protocols can be combined in a correct and desirable way to achieve a security goal.

Acknowledgments

We wish to thank the following colleagues for their many useful technical advises and logistic support; this work would have been impossible without their help: Erol Basturk, Mihir Bellare, Maria A. Butrico, Chee–Seng Chow, Mark C. Davis, Edie E. Gunter, Donald B. Johnson, Dilip D. Kandlur, Jed Kaplan, Arvind Krishna, Mark H. Linehan, Charles C. Palmer, Ed Pring, Stephen E. Smith and Moti M. Yung. We also thank the anonymous referees for their many comments, which helped improving the presentation of the paper.

References

[Ass86] American Bankers Association. *American National Standard for Financial Institution Message Authentication (Wholesale).* ANSI X9.9, 1981, revised 1986.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keyed Hash Functions and Message Authentication. In N. Koblitz, editor, *Advances in Cryptology — Crypto '96*, pages 1–15. Springer-Verlag, 1996. Lecture Notes in Computer Science No. 1109.

[Bel96] Steven M. Bellovin. Problem Areas for the IP Security Protocols. In *the Proceedings of 6th USENIX UNIX Security Symposium*, July 1996.

[BGH+93] Ray Bird, Inder Gopal, Amir Herzberg, Philippe A. Jason, Shay Kutten, Refik Molva, and Moti Yung. Systematic Design of a Family of Attack–Resistant Authentication Protocols. *IEEE Journal on Selected Areas in Communications*, 11(5), June 1993.

[BGH+95] Ray Bird, Inder Gopal, Amir Herzberg, Phil Jason, Shay Kutten, Refik Molva, and Moti Yung. The KryptoKnight Family of Light–Weight Protocols for Authentication and Key Distribution. *IEEE/ACM Transc. on Networking*, 3(1), February 1995.

[BR93] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *Advances in Cryptography*, August 1993.

[CB94] William R. Cheswick and Steven M. Bellovin. *Firewalls and Internet Security Repelling the Willy Hacker.* Addison Wesley, 1994.

[CGHK95] Pau-Chen Cheng, Juan A. Garay, Amir Herzberg, and Hugo Krawczyk. Design and Implementation of Modular Key Management Protocol and IP Secure Tunnel on AIX. In *Proc. 5th USENIX UNIX Security Symposium*, June 1995.

[DA97] Tim Dierks and Christopher Allen. The TLS Protocol Version 1.0. IETF <draft-ietf-tls-protocol-02.txt>, March 1997.

[DBC92] *Great Circle Associates* D. Brent Chapman. Network (In)Security Through IP Packet Filtering. In *UNIX Security Symposium III Proceedings*, pages 63–76, 1992.

[DH76] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transc. on Information Theory*, IT–22(6), November 1976.

[DvOW92] W. Diffie, P. van Oorschot, and M. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[FKK96] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL Protocol Version 3.0. IETF <draft-ietf-tls-ssl-version3-00.txt>, November 1996.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):210–217, 1986.

[HC97] Dan Harkins and Dave Carrel. The resolution of ISAKMP with Oakley. IETF <draft-ietf-ipsec-isakmp-oakley-04.txt>, July 1997.

[IB94a] John Ioannidis and Matt Blaze. The Architecture and Implementation of Network–Layer Security under UNIX. In *the Proceedings of 1994 Winter USENIX Conference Proceedings*, 1994.

[IB94b] John Ioannidis and Matt Blaze. *The swIPe IP Security Protocol.* IETF <draft-ipsec-swipe-01.txt>, June 1994.

[KA97a] Stephen Kent and Randall Atkinson. IP Authentication Header. IETF <draft-ietf-ipsec-auth-header-01.txt>, July 1997.

[KA97b] Stephen Kent and Randall Atkinson. IP Encapsulating Security Payload (ESP). IETF <draft-ietf-ipsec-esp-v2-00>, July 1997.

[KA97c] Stephen Kent and Randall Atkinson. Security Architecture for the Internet Protocol. IETF <draft-ietf-ipsec-arch-sec-01.txt>, March 1997.

[KBC97] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Message Authentication. Internet RFC 2104, February 1997.

[KN93] John Kohl and B. Clifford Neuman. The Kerberos Network Authentication Service (V5). Internet RFC 1510, September 1993.

[Kra96] Hugo Krawczyk. SKEME: A Versatile Secure Key Exchange Mechanism for Internet. In *the Proceedings of 1996 Internet Society Symposium on Network and Distributed Systems Security*, February 1996.

[LJFK86] Samuel J. Leffler, William N. Joy, Robert S. Farby, and Michale J. Karel. Networking Implementation Notes, 4.3BSD Edition. In *UNIX System Manager's Manual, 4.3 Berkeley Software Distribution, Virtual VAX–11 Edition*. USENIX Association, April 1986.

[MSST97] Douglas Maughan, Mark Schertler, Mark Schneide, and Jeff Turner. Internet Security Association and Key Management Protocol (ISAKMP). IETF <draft-ietf-ipsec-isakmp-08.txt>, July 1997.

[OG97] Mike Oehler and Robert Glenn. HMAC–MD5–96 IP Authentication with Replay Prevention. IETF <draft-ietf-ipsec-ah-hmac-md5-96-00.txt>, March 1997.

[Orm97] Hilarie Orman. The Oakley Key Determination Protocol. draft-ietf-ipsec-oakley-02.txt, July 1997.

[Pos81] J. Postel. Internet Protocol. Internet RFC 791, September 1981.

[Tsu92] G. Tsudik. Message authentication with one-way hash functions. In *Proceedings of Infocom 92*, 1992.

IP Layer Security vs. Session Layer Security

A frequently asked question is why are both an IP layer security mechanism and a session layer security mechanism—such as SSL [FKK96] and TLS [DA97]—needed. Table 1 presents a comparison of features between IPSEC and SSL.

In summary, we feel that IPSEC and SSL are largely complementary technologies. When the traffic is restricted to Web/HTTP type, then SSL is more suitable, assuming that all the browsers and Web servers have SSL already built-in. When traffic cannot be restricted only to Web/HTTP, but must include other popular Internet applications such as telnet, ftp, NFS, directory services, DB applications, real-time audio/video as well as many other legacy applications, then IPSEC is simpler more suitable.

Authors' Biographies

PAU-CHEN CHENG. Pau-Chen Cheng is a Research Staff Member at the IBM T.J. Watson Research Center. He received his Ph.D. in Electrical Engineering from University of Maryland at College Park, Maryland, USA, in 1990. He joined the Computing System Department of the IBM T.J. Watson Research Center in 1990 to work on the development of security functions of AIX. In 1994 he joined the Network Security Group in IBM T.J. Watson Research Center to work on IP Security technology. He is the principal developer of the IPSEC technology on the AIX operating system. His areas of interest is in the system aspect of computer and network security. He has been involved in the design, analysis and implementation of solutions for data encryption and authentication, key management, user authentication and Internet security.

JUAN A. GARAY. Juan Alberto Garay received his Ph.D. degree in Computer Science from Penn State University in 1989. He also holds the degree of Electrical Engineer from the Universidad Nacional de Rosario, Rosario, Argentina, where he is originally from, and a Master's degree in Electronic Engineering from the Eindhoven International Institute of the Eindhoven University of Technology, Eindhoven, Holland. He has been with IBM Research since 1990. In 1992 he was a post-doctoral fellow at The Weizmann Institute of Science, Rehovot, Israel, and in 1996 a visiting scientist at the Centrum voor Wiskunde en Informatica (CWI) of the Stichting Mathematish Centrum, Amsterdam, The Netherlands. Dr. Garay has published extensively in the areas of algorithms, distributed computing, fault tolerance, and cryptographic protocols.

AMIR HERZBERG. Amir Herzberg received the B.Sc. (Computer Engineering), M.Sc. (Electrical Engineering) and D.Sc. (Computer Science) degrees from the Technion – Israel Institute of Technology, in 1982, 1986, and 1991, respectively.

Since 1991, he is with the IBM Research Division. His current position (since January 1996) is as manager of the Network Computing and Security group of the Haifa Research Lab. His previous assignment was as manager of the Network Security group in the Watson Research Center. His research areas include network security, applied cryptography, electronic commerce, communication protocols and fault tolerant distributed algorithms.

Hugo Krawczyk. Hugo Krawczyk is a Senior Lecturer in the Department of Electrical Engineering at the Technion, Israel, and a Visiting Scientist at the IBM T.J. Watson Research Center. He received his Ph.D. in Computer Science from the Technion, Israel, in 1990. In 1990/91 he spent a year in the Computer Science department of Princeton University under a Weizmann post-doctoral fellowship. From 1991 to 1997 he was a Research Stuff Member at the Cryptography and Network Security Group in IBM T.J. Watson Research Center.

His areas of interest span applied and theoretical aspects of cryptography with particular emphasis on applications to network security. He has been involved in the design and implementation of solutions for data encryption and authentication, key management, public key cryptography, Internet security, electronic commerce, payment systems, and security of mobile and wireless systems.

S R

S randomly chooses N_S (N_R shared in previous run)

$$\xrightarrow{\quad N_S, MAC_{K_1}(N_R, N_S, ID_S, ID_R) \quad}$$

R randomly chooses N'_R

$$\xleftarrow{\quad N'_R, MAC_{K_1}(N_S, N'_R, ID_R, ID_S) \quad}$$

$N_R := N'_R$

Let $T = MAC_{K_1}(N'_R, N_S)$

$SK := f_{K_2}(T)$

Figure 3: MKMP: The session key negotiation protocol.

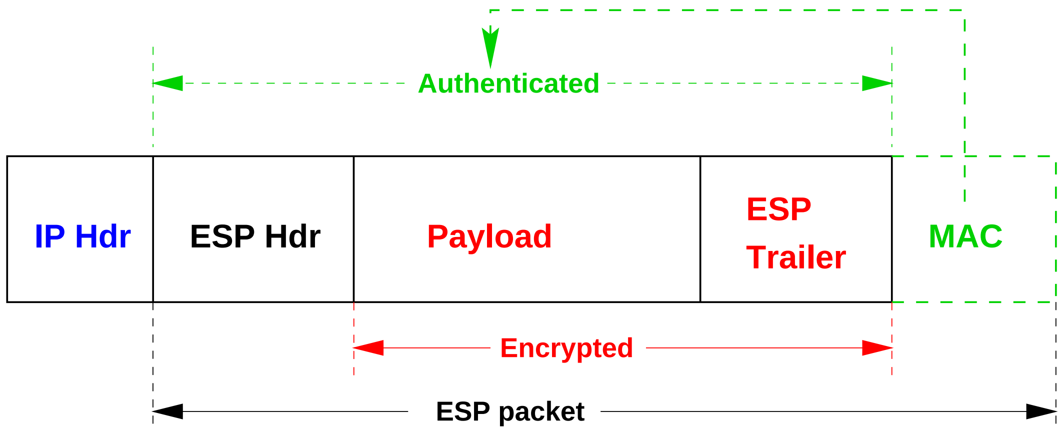

Figure 4: An ESP packet encapsulated in an IP packet.

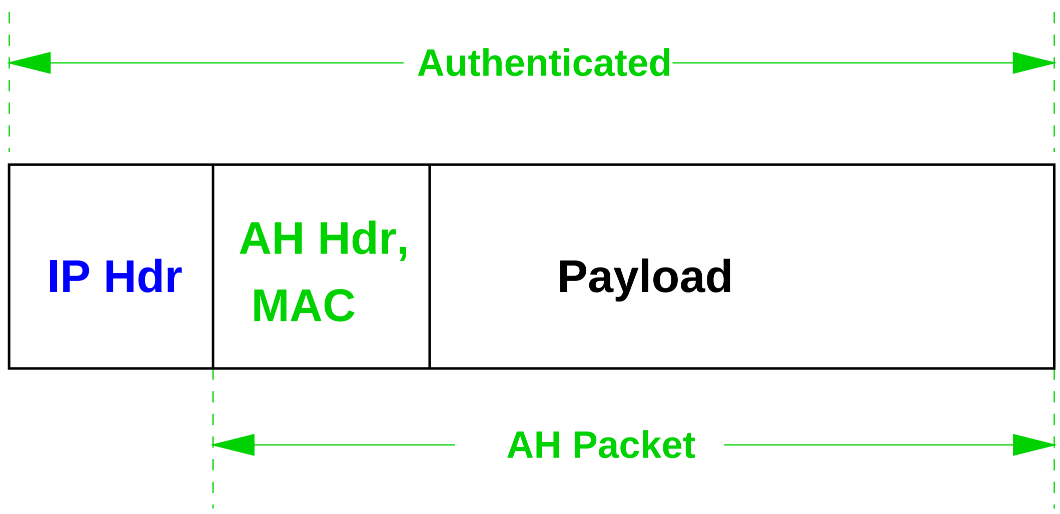

Figure 5: An AH packet encapsulated in an IP packet.

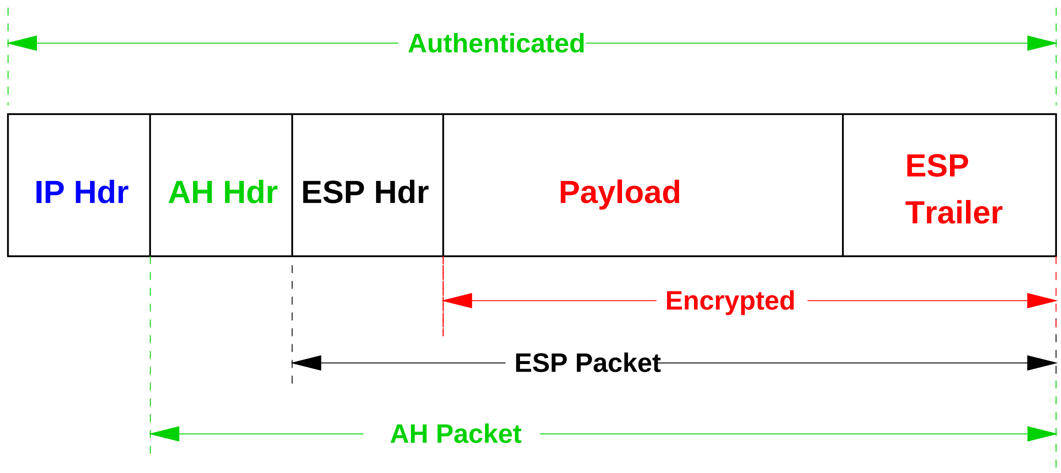

Figure 6: An ESP packet encapsulated in an AH packet encapsulated in an IP packet.

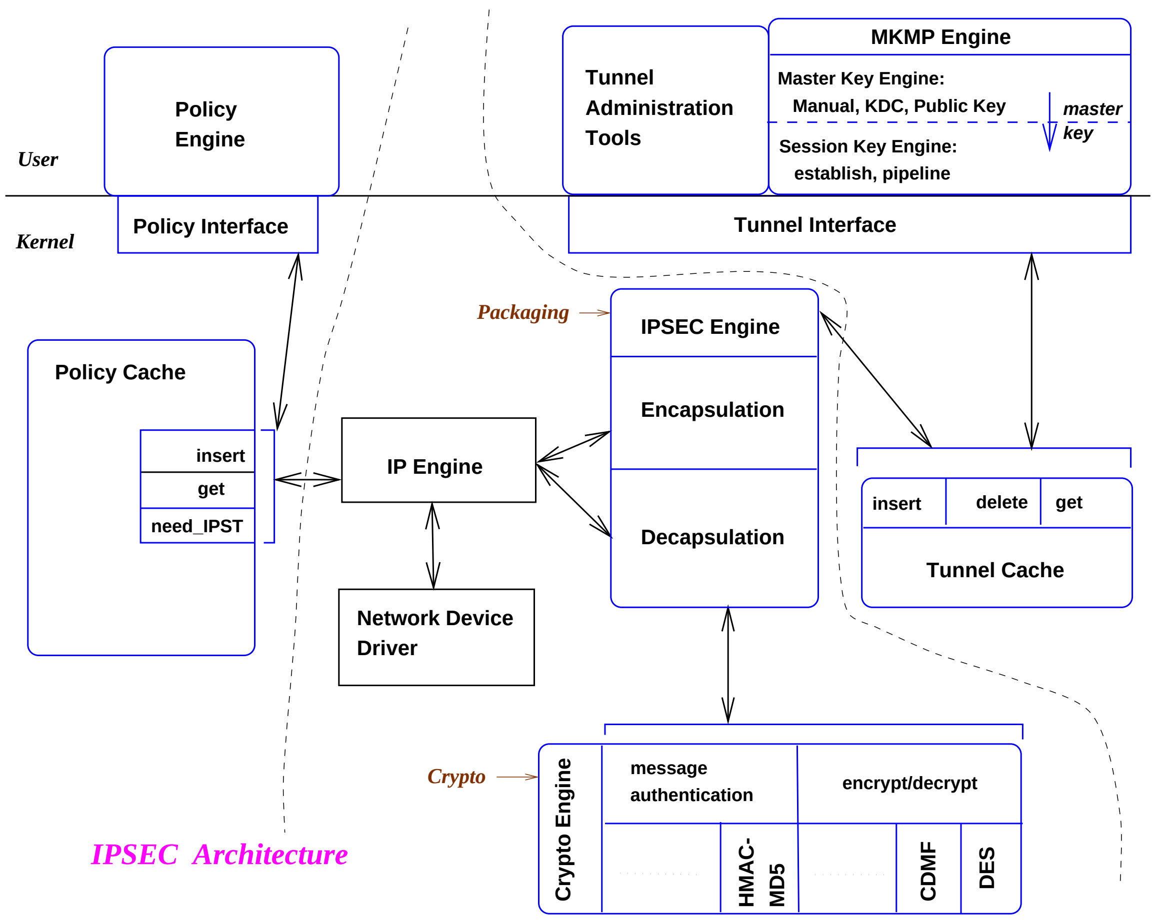

Figure 7: IPSEC System Architecture.

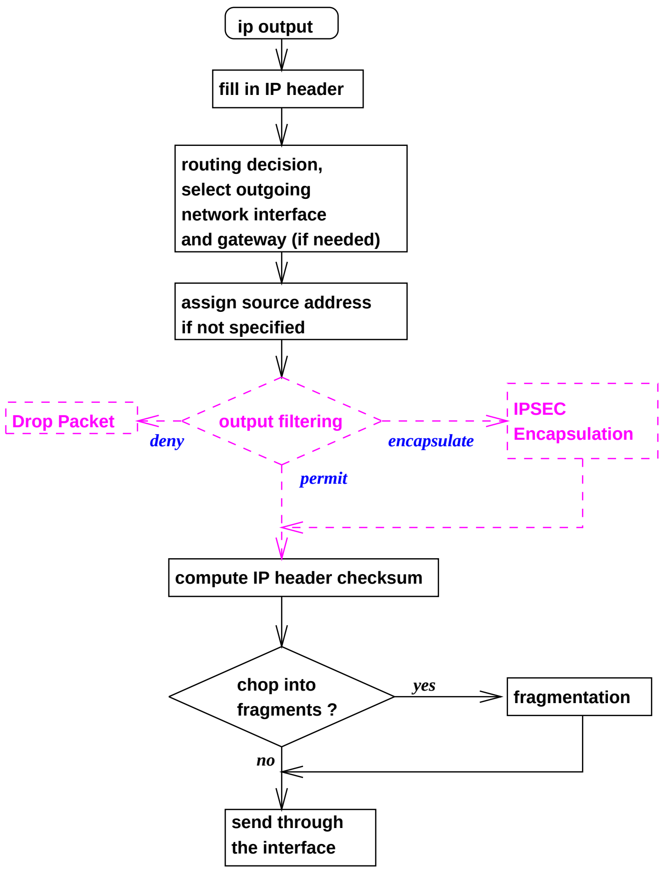

Figure 8: IP/IPSEC output processing.

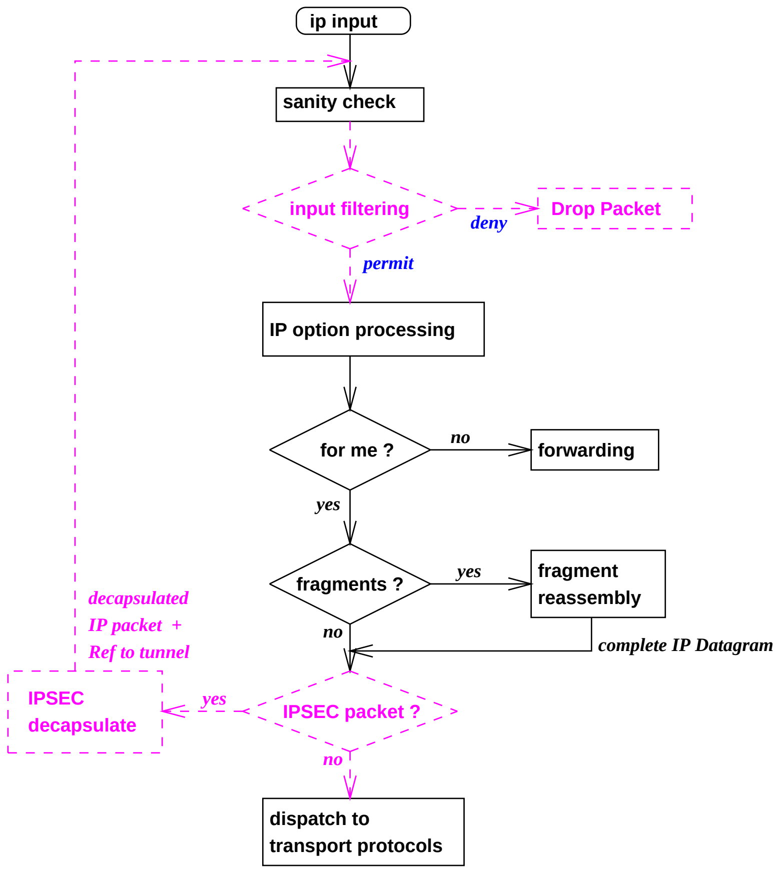

Figure 9: IP/IPSEC input processing.

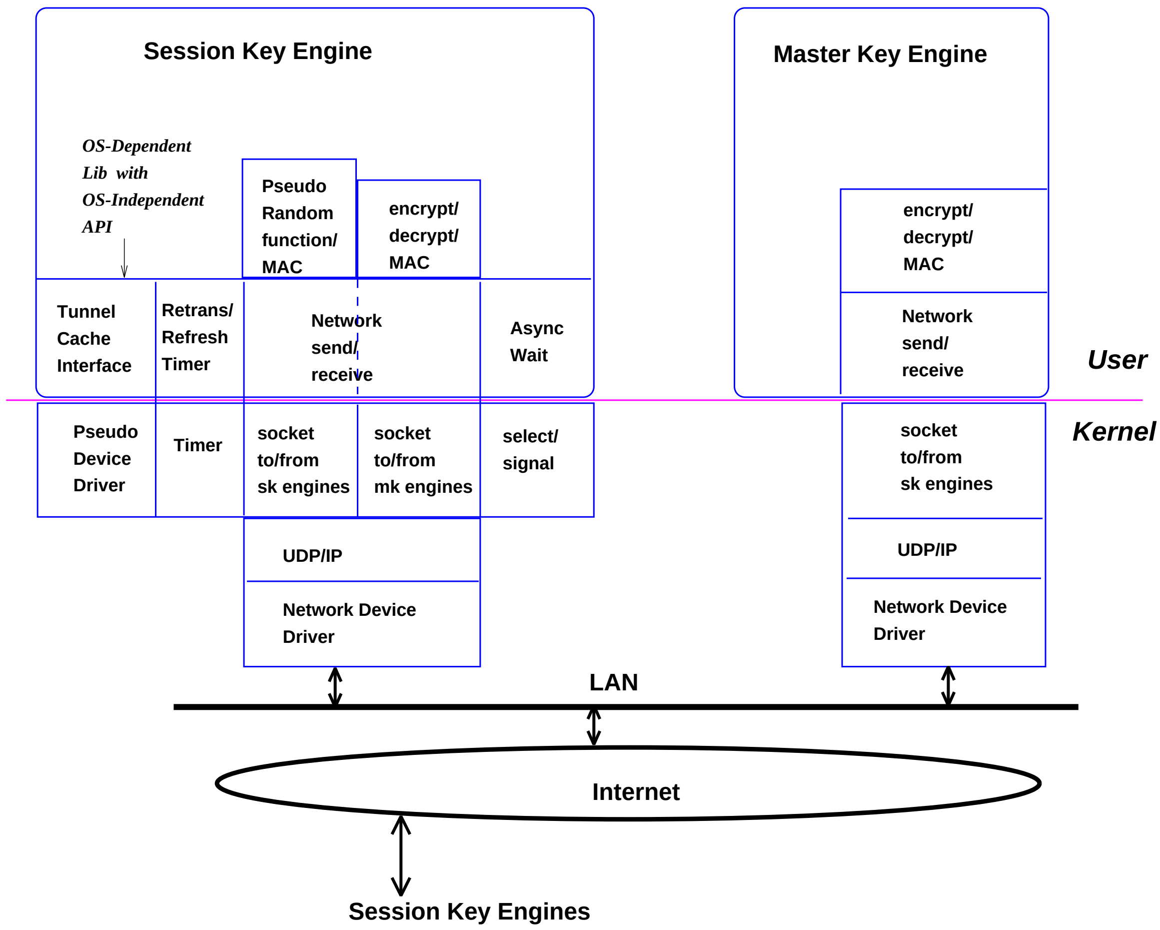

Figure 10: Architecture of Key Management engines.

	IPSEC	SSL
Hardware–Independence	YES	YES
Code	No modifications to applications. *May* need access to TCP/IP stack source code.	Modifications to applications. May need new DLL or access to application source code.
Protection	Entire IP packet. Includes protection for higher layer protocols.	Only application layer.
Packet Filtering	Based on authenticated headers, source/destination addresses, etc. Simpler and lower cost. Suitable for routers.	Based on content and higher-level semantics. More intelligent and more complex. But also desirable.
Performance	Less context switching and data movement.	More context switching and data movement. Larger data units may help speedup of cryptographic operations and better compression.
Platform	Any systems, routers included.	Mainly end systems (clients/servers), also on firewalls.
Firewall/VPN	All traffic is protected.	Only application-level traffic is protected. ICMP, RSVP, QoS, etc. may not be protected.
Transparency	To users and applications.	To users only.
Current Deployment	Emerging standard, supported by most firewall vendors.	Widely used by WWW browsers; also used by some products to provide session layer secure tunnel.

Table 1: IP Layer Security vs. Session Layer Security