MODULE 5

Domainkeys Identified Mail.

IP Security: IP Security overview, IP Security Policy, Encapsulating Security Payload, Combining security associations, Internet key exchange.

Domainkeys Identified Mail

DomainKeys Identified Mail (DKIM) is a widely adopted technique used to *cryptographically sign* email messages so that a receiving system can verify the authenticity of the sender's domain. In DKIM, the **signing domain** attaches a digital signature to the email header. The receiving mail server retrieves the corresponding **public key** from the sender's DNS records to verify the signature. If the signature matches, the message is confirmed as having been sent by a system in possession of the domain's **private key**. DKIM is defined as an **Internet Standard (RFC 6376)** and is used by major providers such as **Gmail**, **Yahoo!**, corporations, government agencies, and many large **ISPs**. Its core purpose is to enhance trust in email by preventing attackers from falsely claiming to send mail from a legitimate domain.

Email Threats

DKIM, it is essential to examine the range of **email threats**, as outlined in **RFC 4686**. The document identifies different types of attackers based on their **characteristics**, **capabilities**, and **location** within the network.

Attacker characteristics are categorized into three threat levels.

1. At the simplest level are individuals who send unwanted messages by **forging origin addresses** using easily available tools. This makes traditional filtering techniques ineffective because the sender's domain can be faked.
2. The next level consists of **professional bulk spammers**. These attackers operate commercially, using **Mail Transfer Agents (MTAs)**, **registered domains**, and networks of **compromised systems (zombies)** to send large volumes of spam and even harvest email addresses.

3. The most advanced attackers are those engaged in **financially motivated fraud**, such as phishing operations. These adversaries may also target the **Internet infrastructure**, carrying out **DNS cache poisoning** or **IP routing attacks** to mislead users or redirect traffic.

Capabilities

1. **Submit messages to MTAs/MSAs:**

   Attackers may send emails through multiple **Mail Transfer Agents (MTAs)** or **Message Submission Agents (MSAs)** across the Internet, giving them many entry points.

2. **Construct arbitrary header fields:**

   They can create fake **Message Header fields**, posing as **mailing lists**, **resenders**, or other legitimate agents.

3. **Sign messages for domains they control:**

   If an attacker owns a domain, they can generate **valid signatures** for malicious emails.

4. **Generate large volumes of unsigned or fake-signed messages:**

   This capability can be used to launch **denial-of-service (DoS)** attacks on recipient mail servers.

5. **Resend previously signed messages:**

   Attackers can reuse earlier emails that were legitimately signed by a domain to mislead recipients.

6. **Manipulate envelope information:**

   They can send messages with any **envelope sender/receiver details**, disguising the true origin.

7. **Act as an authorized submitter from a compromised system:**

   If a computer is infected, attackers may send mail as though they are a trusted user of that system.

8. **Manipulate IP routing:**

   They can influence **routing paths**, allowing messages to come from specific or hard-to-trace IP addresses or divert traffic to particular domains.

9. **Influence portions of DNS:**

   Through techniques like **DNS cache poisoning**, attackers may alter message routing or falsify DNS-based **public key records** used for DKIM.

10. **Use large-scale computing resources:**

5

Networks of compromised **zombie computers** give attackers the power to perform **brute-force attacks** or other resource-heavy operations.

11. **Eavesdrop on existing traffic:**

Attackers may capture email data by tapping into unsecured channels, such as **wireless networks**, enabling monitoring or theft of information.

LOCATION

DKIM primarily focuses on attackers located **outside the administrative boundaries** of the sender and recipient. Within these protected internal networks, authenticated submission methods already exist and scale well. The real challenge arises in the open, "**any-to-any**" nature of the global email system, where external attackers exploit the fact that most mail servers accept messages from any origin. Such attackers may send messages with **no signatures**, **invalid signatures**, or **valid signatures from obscure domains** that provide little traceability. They may imitate **mailing lists**, **greeting card services**, or other legitimate agents to mislead recipients.

DKIM Strategy

DomainKeys Identified Mail, or **DKIM**, provides an email authentication method that works quietly in the background, without requiring the end user to do anything. The idea is straightforward: whenever a message is sent, the **administrative domain** from which it originates signs the message using its **private key**. This signature covers the entire message content along with selected **RFC 5322 headers**, ensuring that both the text and certain header fields are protected from tampering.

At the receiving side, the **Mail Delivery Agent (MDA)** retrieves the corresponding **public key** published in DNS by the sender's domain. Using this public key, the MDA verifies the signature. If the verification succeeds, the message can be trusted to have genuinely originated from the claimed domain. If someone tries to send mail from outside the domain but pretends to be that domain, the signature will not validate, allowing the system to **reject fraudulent mail**.

This design stands apart from **S/MIME** and **PGP**, both of which rely on the *originator's* personal private key to sign message content. The reasoning behind DKIM's approach is practical:

- In reality, most users neither send nor receive many S/MIME-protected messages. A user's inbox typically contains mail from senders who do not use S/MIME.

- S/MIME protects only the message body, not crucial RFC 5322 headers such as the sender's address. These headers can still be altered by attackers.
- DKIM requires no action from the user because it is not implemented at the email client (MUA) level; everything happens in the background at the server level.
- All messages from cooperating domains are automatically signed, strengthening trust across large parts of the email ecosystem.
- This system enables legitimate senders to prove they truly sent a message and prevents attackers from impersonating them.

Figure **19.10** illustrates a simple flow of DKIM in action. A user composes an email using their client program. Once this message enters the Mail Handling System (MHS) and reaches the domain's **Mail Submission Agent (MSA)**, the provider signs the message content and selected headers using the provider's private key. The signing entity is tied to a domain, which could

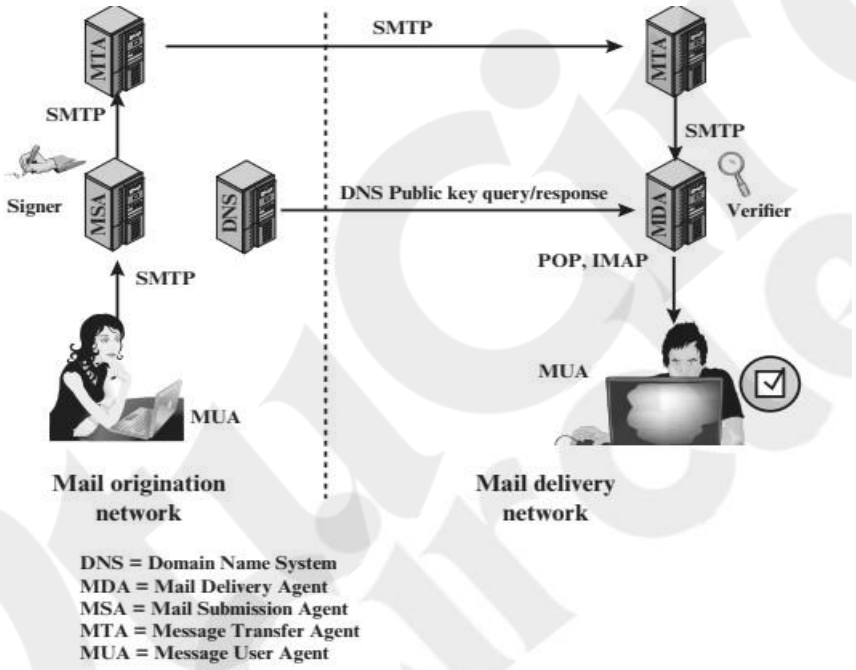


Figure 19.10 Simple Example of DKIM Deployment

belong to an enterprise, an ISP, or a large public service such as Gmail.

After signing, the message travels through the Internet via one or more **Mail Transfer Agents (MTAs)**. When the message arrives at the destination domain, the **MDA** fetches the sender domain's public key from DNS, verifies the signature, and then forwards the validated message to the recipient's email client. The default algorithm used for signing is **RSA with SHA-256**, though **RSA with SHA-1** may still appear in some deployments.

DKIM Functional Flow

Figure **19.11** shows a more detailed sequence of how DKIM works in practice. The entire processing is divided between two entities: a **signing Administrative Management Domain (ADMD)** and a **verifying ADMD**. In the simplest path, these are the sender's ADMD and the recipient's ADMD, though additional ADMDs may be involved along the mail route.

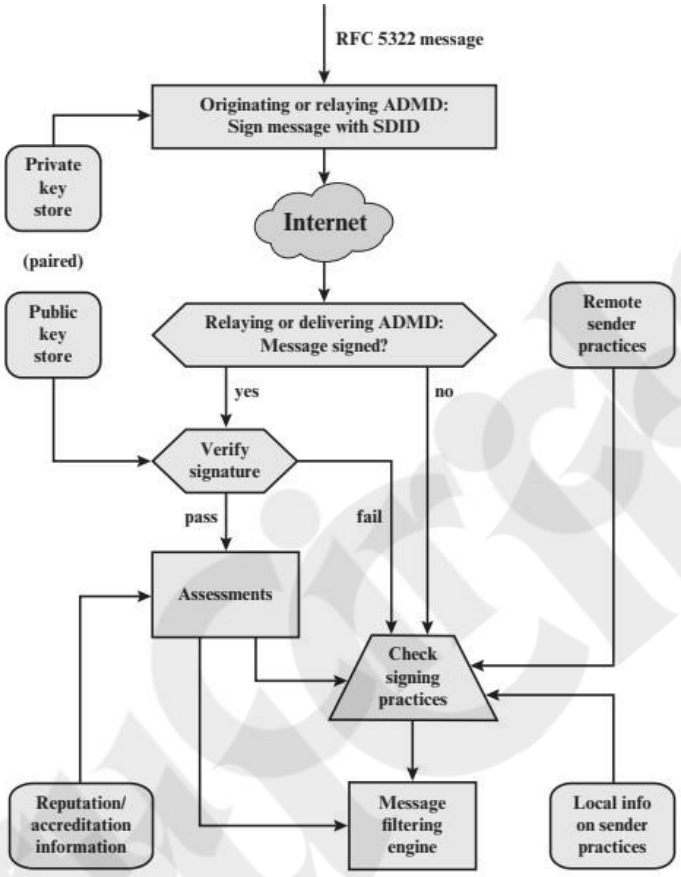


Figure 19.11 DKIM Functional Flow

Within the signing ADMD, an authorized module performs the signing operation using private key information fetched from its **Key Store**. This signing step could be handled by the MUA, MSA, or even an MTA, depending on how the originating domain is configured.

In the verifying ADMD, another authorized module checks the signature. This may again be executed by an MTA, an MDA, or even the user's MUA. The verification step relies on the public key stored in DNS and retrieved from the Key Store. After verification, the module decides whether the signature is valid or whether a signature was expected in the first place.

If verification succeeds, the system uses **reputation information** about the signer to help the

filtering mechanisms classify the message properly. If the signature fails, or if a domain that normally signs its messages (like Gmail) has sent a message without a DKIM signature, the verifier may retrieve information about that domain's signing practices—both locally and remotely—and pass this information to the filtering system. This helps identify fraudulent or suspicious messages.

The DKIM signature itself appears within the RFC 5322 header as a new header field beginning with **Dkim-Signature**. Users can see this field by choosing the "View Long Headers" option in their mail client. An example of such a signature is shown below:

```
Dkim-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;
d=gmail.com; s=gamma; h=domainkey-
signature:mime-version:received:date:
message-id:subject:from:to:content-type:
content-transfer-encoding;
bh=5mZvQDyCRuyLb1Y28K4zgS2MPOemFToDBgvbJ
7GO90s=;
b=PcUvPSDygb4ya5Dyj1rbZGp/VyRiScuaz7TTG
J5qW5slM+klzv6kcfYdGDHzEVJW+Z
FetuPfF1ETOVhELtwH0zjSccOyPkEiblOf6gILO
bm3DDRm3Ys1/FVrbhVOlA+/jH9Aei
uIIw/5iFnRbSH6qPDVv/beDQqAWQfA/wF7O5k=
```

Before the message is signed, both the header and body undergo a process known as **canonicalization**. This step prepares the message to withstand small, routine changes made during mail handling—such as variations in whitespace, line endings, encoding changes, or header folding and unfolding. Canonicalization aims to produce a minimal but consistent transformation that increases the likelihood of matching the same canonical form at the verifying end.

DKIM offers two canonicalization algorithms for both the header and body: **simple** and **relaxed**. The simple algorithm expects no modifications at all, while the relaxed version allows for common adjustments that often occur during transit.

The signature is composed of several fields. Each field begins with a tag code followed by "=", and ends with a semicolon. Key fields include:

- **v=** DKIM version.
- **a=** Signature algorithm, typically *rsa-sha1* or *rsa-sha256*.
- **c=** Canonicalization method used for both header and body.
- **d=** The signing domain (SDID), identifying the responsible organization or sender domain.
- **s=** The selector, allowing multiple keys to be used within the same domain for rotation or

departmental separation.

- **h=** List of header fields that were included during signing.
- **bh=** Hash of the canonicalized body, useful for diagnosing signature failures.
- **b=** The actual signature in base64 form, representing the encrypted hash.

IP SECURITY OVERVIEW

- In 1994, the Internet Architecture Board (IAB) released *RFC 1636: Security in the Internet Architecture*, stressing the need to protect network infrastructure from **unauthorized monitoring** and **control of traffic**, and to secure **end-to-end communication** using authentication and encryption.

- To address these concerns, the IAB recommended integrating **authentication** and **encryption** into the next-generation IP (IPv6). These mechanisms were designed to work with **both IPv4 and IPv6**, enabling early vendor adoption. Today, these security features are implemented through a standardized set of protocols known as **IPsec (IP Security)**.

APPLICATIONS OF IPsec (Concise Notes)

IPsec supports secure communication over **LANs**, **private/public WANs**, and the **Internet**, making it widely useful in enterprise networking.

- **Secure branch office connectivity:**
  Organizations can create **VPNs over the Internet** instead of relying on costly private networks. IPsec ensures encrypted, authenticated communication between branch offices, reducing costs and simplifying network management.

- **Secure remote access:**
  Remote users can connect securely to the company network by dialing into any ISP and then using IPsec. This avoids long-distance toll charges and supports employees working from different locations.

- **Secure extranet and intranet connectivity:**
  IPsec helps build secure links with partners or between internal networks. It ensures

5

authentication, confidentiality, and a secure **key-exchange mechanism**, making cross-organizational communication trustworthy.

- **Strengthening electronic commerce:**
  Although many web or e-commerce platforms use their own security protocols, IPsec adds an extra layer by ensuring that *all designated traffic* is **encrypted and authenticated**, enhancing overall transaction security.

The strength of IPsec lies in its ability to **encrypt and/or authenticate traffic at the IP layer**. Because protection happens at this fundamental level, all distributed applications—remote login, client/server communication, email, file transfer, and web access—automatically benefit from IPsec security without needing changes at the application layer.
IPsec supports a mode called **Tunnel Mode**, whose simplified packet structure is shown in *Figure 20.1a*. Tunnel mode uses:

- **ESP (Encapsulating Security Payload)** for combined encryption and authentication
- A **key exchange mechanism** to establish secure communication

In VPN deployments, both encryption and authentication are essential to ensure that:

1. **Only authorized users** are able to access the VPN.
2. **No eavesdropper** on the Internet can read the transmitted data.

Figure 20.1b illustrates a common IPsec usage scenario. An organization has LANs located at different sites. Inside each LAN, normal (nonsecure) IP traffic flows. When traffic is sent offsite through a private or public WAN, IPsec mechanisms protect it.
Routers or firewalls at the network boundary usually perform the IPsec operations—encrypting outbound WAN traffic and decrypting inbound WAN traffic. These actions are **transparent** to internal systems on the LAN.

5

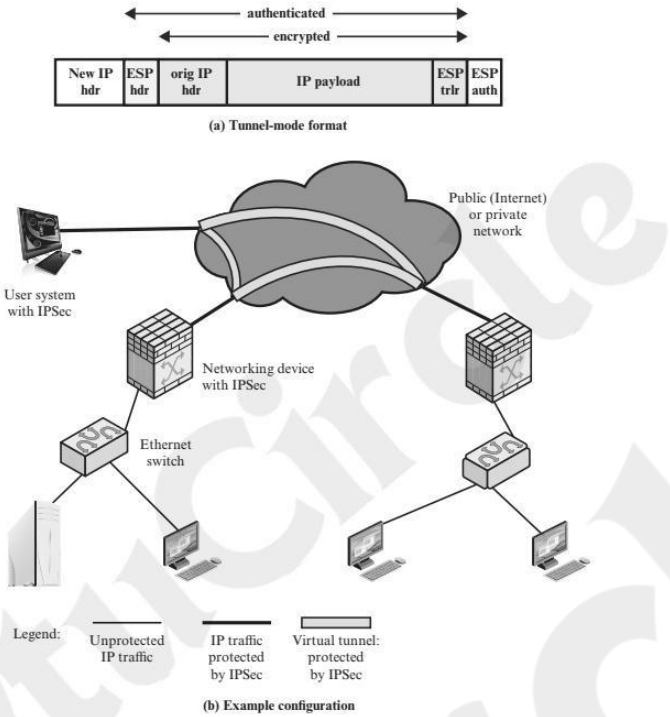


Figure 20.1 An IPSec VPN Scenario

Individual remote users can also establish secure connections by dialing into the WAN. Their devices must run IPsec protocols to ensure protection similar to that used by the organization's gateways.

BENEFITS OF IPsec

• When IPsec is deployed on a **firewall or router**, it delivers strong protection for all traffic crossing the network boundary. Internal traffic inside the organization does not carry the extra processing overhead of security mechanisms.

• A firewall using IPsec is **highly resistant to bypass**. If all external communication must use IP, and the firewall is the only entry point to the organization, attackers cannot avoid the IPsec policies enforced at the perimeter.

• Because IPsec works **below the transport layer** (under TCP/UDP), it is completely **transparent to applications**. No user-level or server-level software needs modification. Even when implemented in end systems, higher-layer applications continue to function without change.

• IPsec operation can be **transparent to end users**. Users do not require training in cryptographic methods, nor must administrators distribute or revoke keying material on a per-user basis.

• IPsec can also secure communication for **individual users** when necessary. This supports

remote employees and allows the organization to create secure virtual subnets for sensitive internal applications.

ROUTING APPLICATIONS OF IPsec

Besides protecting end users and organizational networks, IPsec also strengthens the **routing architecture** that connects multiple networks. Secure routing is essential to prevent attackers from misleading routers or diverting traffic.

According to examples listed in [HUIT98], IPsec helps ensure that:

- **Router advertisements** originate only from authorized routers, preventing fake routers from announcing themselves on the network.
- **Neighbor advertisements**, used when routers attempt to establish or maintain neighbor relationships across routing domains, are sent only by legitimate routers.
- **Redirect messages** genuinely come from the router that originally received the IP packet, preventing false redirections of traffic.
- **Routing updates** are not forged, ensuring that routing tables are accurate and not tampered with.

Without these protections, attackers can disrupt communication, reroute traffic, or create denial-of-service conditions.

Routing protocols such as **OSPF** should therefore operate over security associations established through IPsec, ensuring authenticated and integrity-protected routing exchanges.

IPsec DOCUMENTS

IPsec provides three major security functions—**authentication**, **confidentiality**, and **key management**. Its full specification is spread across many RFCs, making it one of the most complex IETF standards. To understand the complete structure of IPsec, the best reference is the **IPsec Document Roadmap**, currently documented in **RFC 6071 (2011)**.

The various documents defining IPsec can be grouped into the following categories:

- **Architecture:**

Defines the core concepts, security requirements, and mechanisms behind IPsec.

Current reference: **RFC 4301 – Security Architecture for the Internet Protocol.**

- **Authentication Header (AH):**

AH provides message authentication using an extension header.

Current reference: **RFC 4302 – IP Authentication Header.**

Since ESP already offers authentication, AH is **deprecated** in IPsecv3 and kept only for compatibility. It is not recommended for new applications.

• Encapsulating Security Payload (ESP):

ESP adds an encapsulating header and trailer to support **encryption** or combined **encryption + authentication**.

Current specification: **RFC 4303 – IP Encapsulating Security Payload (ESP).**

• Internet Key Exchange (IKE):

A set of protocols for establishing and managing cryptographic keys used by IPsec. Main standard: **RFC 7296 – IKEv2 Protocol**, along with several supporting RFCs.

• Cryptographic Algorithms:

A large group of documents defining algorithms for **encryption**, **message authentication**, **PRFs**, and **key exchange** to be used with IPsec.

• Other Documents:

Includes RFCs related to **security policy**, management information (MIB), and other operational aspects required to support IPsec deployment.

IPsec SERVICES

IPsec secures communication at the **IP layer** by allowing systems to choose the appropriate security protocol, select required algorithms, and establish the cryptographic keys needed for protection. It relies on two main protocols:

- **Authentication Header (AH):** Provides authentication and integrity.
- **Encapsulating Security Payload (ESP):** Provides **encryption** and can also offer **authentication**.

According to **RFC 4301**, IPsec supports the following key services:

- **Access control:** Restricts communication to authorized users and systems.
- **Connectionless integrity:** Ensures packets are not altered during transit.
- **Data origin authentication:** Confirms the true source of received data.
- **Anti-replay protection:** Rejects duplicated or replayed packets using sequence numbers.
- **Confidentiality:** Encrypts data to prevent unauthorized viewing.
- **Limited traffic flow confidentiality:** Hides traffic patterns to reduce information leakage.

5

TRANSPORT AND TUNNEL MODES

Both **AH** and **ESP** can operate in two ways: **transport mode** and **tunnel mode**. These modes differ in how much of the IP packet they protect.

Transport Mode

Transport mode protects the **upper-layer protocols**—that is, only the **IP payload** (e.g., TCP, UDP, ICMP).

It is mainly used for **end-to-end communication** between two hosts, such as a client and a server.

• In IPv4, the payload is everything following the IP header.

• In IPv6, the payload includes all fields after the IP header and extension headers (except possibly the destination options header).

Functions:

- **ESP (transport mode):** Encrypts and optionally authenticates only the IP payload; IP header is not encrypted.
- **AH (transport mode):** Authenticates the payload and selected fields of the IP header.

Transport mode does **not** hide the original IP header.

Tunnel Mode

Tunnel mode protects the **entire original IP packet**.

The original packet (inner IP header + payload) is encapsulated inside a **new outer IP packet**.

Procedure:

1. AH/ESP fields are added to the original packet.
2. The entire packet becomes the payload of a new IP packet with a new outer IP header.
3. Routers along the path can see only the **outer** header, not the original (inner) one.

Advantages:

- Hides original source/destination addresses.
- Allows multiple hosts behind a firewall to communicate securely **without running IPsec on each host** (the firewall handles it).

Tunnel mode is used when at least one endpoint is a **security gateway** (firewall or secure router).

5

Example:

Host A sends a packet to Host B.

A's firewall detects that IPsec is required and encapsulates A's packet inside a new IP packet.

This outer packet is routed to B's firewall, which strips the outer header and forwards the inner packet to Host B.

Functions:

- **ESP (tunnel mode):** Encrypts and optionally authenticates the entire inner IP packet (including inner header).
- **AH (tunnel mode):** Authenticates the inner packet and parts of the outer header.

Table 20.1 Tunnel Mode and Transport Mode Functionality

	Transport Mode SA	Tunnel Mode SA
AH	Authenticates IP payload and selected portions of IP header and IPv6 extension headers.	Authenticates entire inner IP packet (inner header plus IP payload) plus selected portions of outer IP header and outer IPv6 extension headers.
ESP	Encrypts IP payload and any IPv6 extension headers following the ESP header.	Encrypts entire inner IP packet.
ESP with Authentication	Encrypts IP payload and any IPv6 extension headers following the ESP header. Authenticates IP payload but not IP header.	Encrypts entire inner IP packet. Authenticates inner IP packet.

IP SECURITY POLICY

IPsec operates based on a **security policy** that determines how each IP packet should be handled as it travels from source to destination. This policy is driven by two key databases:

- **Security Policy Database (SPD)**
- **Security Association Database (SAD)**

Together, these determine whether a packet should be **protected (via IPsec)**, **bypassed**, or **discarded**.

Figure 20.2 in the textbook shows how these databases interact with IPsec processing.

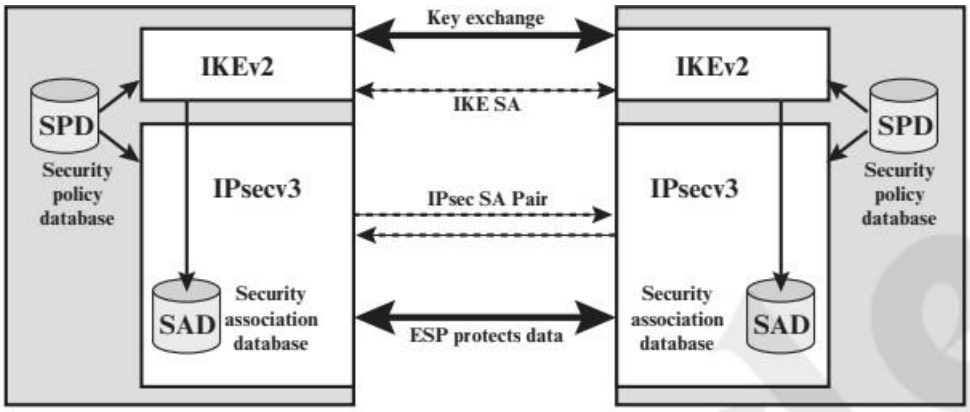


Figure 20.2 IPsec Architecture

Security Associations (SA)

A **Security Association (SA)** is the fundamental building block of IPsec. It is a **one-way logical connection** that defines how traffic should be protected. Since it is one-way, secure two-way communication requires **two separate SAs**.

An SA specifies the security services and keys used for protecting packets.

Each SA is uniquely identified by **three parameters**:

1. **Security Parameters Index (SPI)**
   - A 32-bit value used to identify the SA.
   - Included in AH or ESP headers.
   - Has meaning only to the receiving system.
2. **IP Destination Address**
   - The endpoint receiving the protected traffic.
   - Could be a host, firewall, or router.
3. **Security Protocol Identifier**
   - Indicates whether the SA uses **AH** or **ESP**.

Using these three fields—**Destination IP + SPI + Protocol (AH/ESP)**—the system uniquely identifies the correct SA for processing any incoming or outgoing packet.

Security Association Database (SAD)

Every IPsec implementation maintains a **Security Association Database (SAD)**.

This database stores all the parameters required to process packets for each **Security Association (SA)**.

5

Each SA corresponds to **one SAD entry** and contains the operational details needed for AH or ESP processing.

The typical parameters stored in an SAD entry include:

- **Security Parameters Index (SPI):**

A 32-bit identifier selected by the *receiving* side.

– For **outbound** traffic: used to construct the AH/ESP header.

– For **inbound** traffic: used to select the correct SA for processing.

- **Sequence Number Counter:**

A 32-bit counter used to generate the sequence number in AH/ESP headers.

- **Sequence Counter Overflow Flag:**

Indicates whether overflow should trigger an audit event and block further packet transmission on that SA.

- **Anti-Replay Window:**

Used to detect and reject replayed packets.

- **AH Information:**

Authentication algorithm, keys, key lifetimes, and related parameters (only if AH is used).

- **ESP Information:**

Encryption and authentication algorithms, keys, initialization values, key lifetimes, etc. (only if ESP is used).

- **SA Lifetime:**

Specifies when the SA should be replaced or terminated.

Defines lifetime based on time OR number of bytes processed.

- **IPsec Protocol Mode:**

Either **transport**, **tunnel**, or **wildcard**.

- **Path MTU:**

Stores the discovered maximum packet size allowed along the path and related aging information.

Security Policy Database (SPD)

The **Security Policy Database** plays the central role in deciding how each IP packet should be handled—whether it must use IPsec protection or be allowed to pass without it. Every entry in the SPD defines a specific category of IP traffic and associates it with the appropriate **Security**

5

Association (SA). In situations involving complex networks, multiple SPD entries may map to the same SA, or a single SPD entry may point to several SAs.

Each SPD entry is defined by a group of **selectors**, which are specific IP-layer and upper-layer field values. These selectors act as filters to match outbound packets to their required SAs. The outbound processing of any packet follows a clear sequence:

1. The packet's selector fields are compared with the entries in the SPD to locate a match.
2. Once the correct entry is identified, the system determines the associated **SA** and its **SPI**.
3. The corresponding **IPsec processing** (AH or ESP) is then applied.

The selectors commonly used when defining SPD entries include:

- **Remote IP Address**: Can be a single address, a list, a range, or a wildcard. Ranges and wildcards allow several destination systems to share the same SA, especially when located behind a firewall.
- **Local IP Address**: Similar flexibility is provided here, enabling multiple internal hosts behind a firewall to share a single SA.
- **Next Layer Protocol**: Identifies the protocol running over IP. It may be a specific protocol number, **ANY**, or for IPv6, **OPAQUE**. If AH or ESP is being used, their headers follow directly after this field.
- **Name**: A user identifier from the operating system. Though not a packet field, it is available if IPsec operates within the same OS environment.
- **Local and Remote Ports**: These may be individual TCP/UDP port values, a listed set, or wildcard entries.

An example SPD (such as in Table 20.2 of the reference) illustrates a corporate network with **1.2.3.0/24** and a secure **DMZ at 1.2.4.0/24**. The host **1.2.3.10** is allowed to communicate with a secure server **1.2.4.10** in the DMZ. SPD entries reflect such policies clearly—for example, traffic using **UDP port 500** (IKE) is configured to **bypass IPsec**, ensuring key exchange procedures function without interference.

5

Table 20.2 Host SPD Example

Protocol	Local IP	Port	Remote IP	Port	Action	Comment
UDP	1.2.3.101	500	*	500	BYPASS	IKE
ICMP	1.2.3.101	*	*	*	BYPASS	Error messages
*	1.2.3.101	*	1.2.3.0/24	*	PROTECT: ESP intransport-mode	Encrypt intranet traffic
TCP	1.2.3.101	*	1.2.4.10	80	PROTECT: ESP intransport-mode	Encrypt to server
TCP	1.2.3.101	*	1.2.4.10	443	BYPASS	TLS: avoid double encryption
*	1.2.3.101	*	1.2.4.0/24	*	DISCARD	Others in DMZ
*	1.2.3.101	*	*	*	BYPASS	Internet

IP Traffic Processing

IPsec operates **on every individual IP packet**, ensuring that both outgoing and incoming traffic passes through the required security checks. When enabled, IPsec processes each **outbound packet** before transmission and each **inbound packet** after reception but before the data is delivered to higher-layer protocols such as TCP or UDP.

Outbound Packet Processing

The flow for outbound traffic begins with a block of data from a higher layer (e.g., TCP), which is handed to the IP layer to form an IP header and body. Once the packet is formed, IPsec performs a series of steps:

1. **SPD Lookup:** IPsec checks the **Security Policy Database** to find an entry that matches the packet's selector fields.
2. **No Match → Error:** If no entry matches, the packet is discarded and an error is generated.
3. **Match Found → Policy Applied:**
   - If the policy is **DISCARD**, the packet is dropped.
   - If the policy is **BYPASS**, the packet skips IPsec processing entirely and is sent out normally.
4. **Policy = PROTECT:**
   IPsec searches the **Security Association Database (SAD)** for a relevant SA.
   - If none exists, **IKE** is invoked to create an SA and install the necessary keys.
5. **IPsec Transformation:**
   Based on the SAD entry, the packet undergoes **encryption, authentication, or both**, and either **transport mode** or **tunnel mode** is applied.
   After processing, the packet is passed to the network for transmission.

5

This sequence ensures that outbound traffic is forwarded only after meeting the exact security requirements defined by the SPD and SAD.

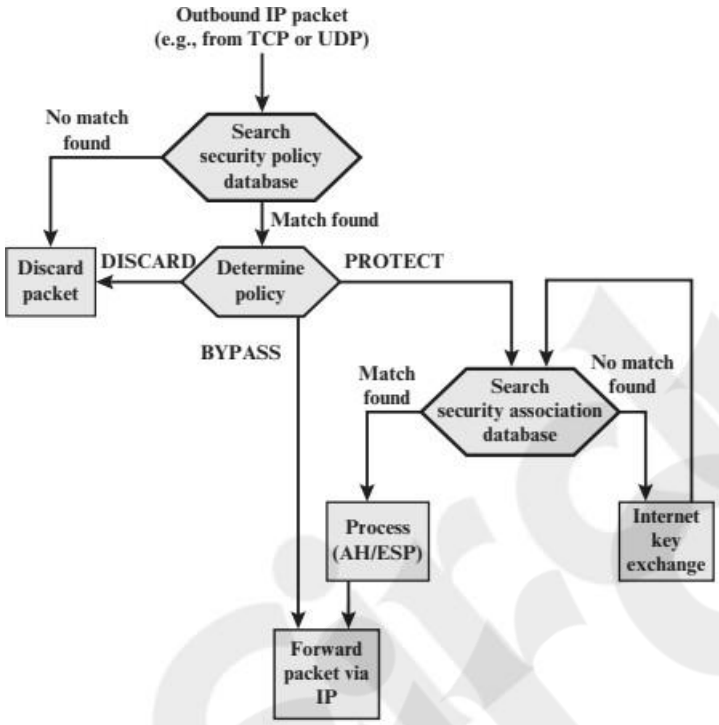


Figure 20.3 Processing Model for Outbound Packets

Inbound Packet Processing

For incoming packets, IPsec examines each one as soon as it arrives. The processing flow begins by looking at the **IP Protocol field (IPv4)** or the **Next Header field (IPv6)** to determine whether the packet contains **ESP**, **AH**, or no IPsec protection.

1. **Check Packet Type:**

- If the packet is **unsecured**, IPsec consults the SPD.
  - If the policy is **BYPASS**, the IP header is removed and the payload is delivered to the next layer (such as TCP).
  - If the policy is **PROTECT**, **DISCARD**, or if no entry matches, the packet is discarded.

2. **Secured Packet (ESP/AH):**

   IPsec searches the **SAD** for a matching SA.

- If no SA matches, the packet is discarded.
- If a match is found, appropriate **ESP or AH processing** is applied.

3. **Deliver to Higher Layer:**

After authentication/decryption (as required), the IP header is removed and the inner payload is forwarded to the corresponding protocol (e.g., TCP).

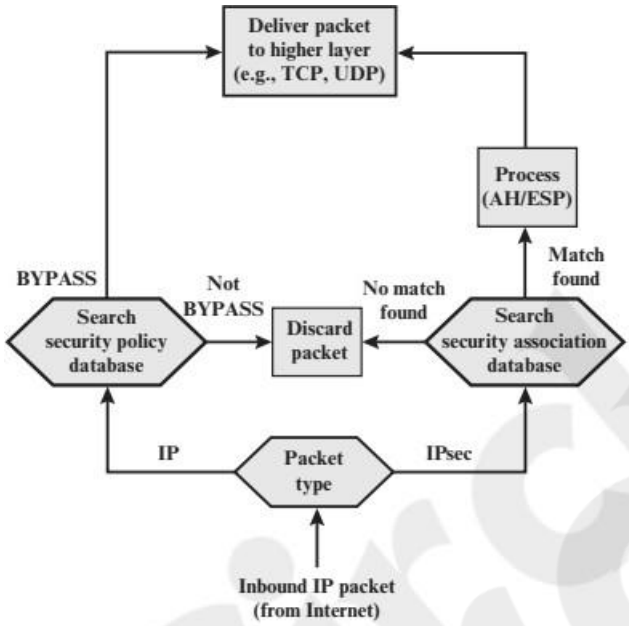


Figure 20.4 Processing Model for Inbound Packets

Encapsulating Security Payload (ESP)

ESP is designed to enhance IP-layer security by offering a collection of protective services. Depending on the configuration negotiated during **Security Association (SA)** setup and the placement of IPsec within the network, ESP can provide **confidentiality**, **data origin authentication**, **connectionless integrity**, **anti-replay protection**, and limited **traffic flow confidentiality**.

ESP supports multiple encryption and authentication algorithms, including modern **authenticated encryption (AEAD)** schemes such as **AES-GCM**, which combine confidentiality and integrity into a single operation.

ESP Packet Format

The overall structure of an ESP packet is illustrated in Figure 20.5a. The main fields are:

- **Security Parameters Index (SPI) – 32 bits:**
  Uniquely identifies the **security association** used to process this packet.
- **Sequence Number – 32 bits:**
  A **monotonically increasing counter** that supports anti-replay protection.

5

- **Payload Data – variable length:**
  Contains either a **transport-layer segment** (in transport mode) or a complete **IP packet** (in tunnel mode). This portion is encrypted.
- **Padding – 0 to 255 bytes:**
  Used for alignment, block-cipher requirements, or to obscure actual message length.
- **Pad Length – 8 bits:**
  Specifies how many bytes of padding precede this field.
- **Next Header – 8 bits:**
  Identifies the type of payload carried (e.g., TCP, UDP, IPv6 extension header).
- **Integrity Check Value (ICV) – variable:**
  Contains the integrity value computed over the ESP header and encrypted portions, except the authentication field itself. Required only when integrity protection is selected and not using a combined mode algorithm.

  When **authenticated encryption (combined mode)** is used, the algorithm returns both the **decrypted payload** and a **pass/fail integrity indication**. In such cases, the external ICV field may be omitted, since integrity verification is handled internally by the algorithm and embedded within the payload structure.

5

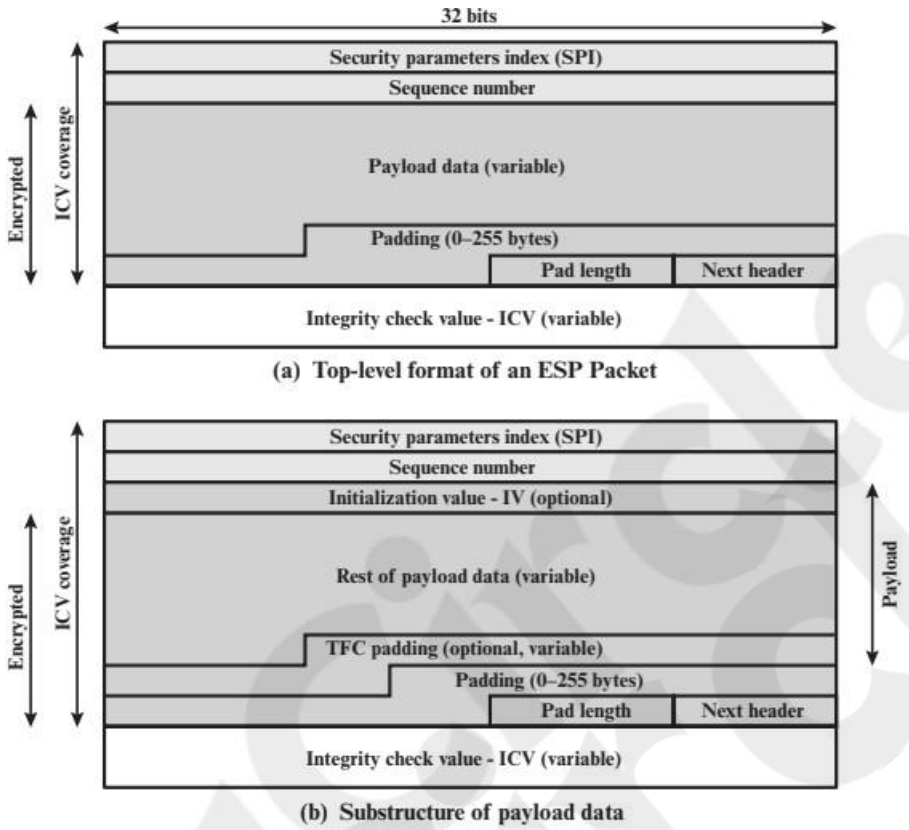


Figure 20.5 ESP Packet Format

Additional ESP Components (Figure 20.5b)

Two optional fields may appear inside the encrypted payload portion:

- **Initialization Vector (IV) or Nonce:**
  Included whenever required by the encryption or AEAD algorithm. It ensures that repeated plaintexts encrypt differently.
- **Traffic Flow Confidentiality (TFC) Padding:**
  Used only in **tunnel mode** to obscure traffic patterns by adding extra bytes before the padding field. This helps protect against traffic analysis attacks.

Encryption and Authentication Algorithms in ESP

In ESP, the fields **Payload Data**, **Padding**, **Pad Length**, and **Next Header** are protected by encryption. When an encryption algorithm requires **cryptographic synchronization data**, such as an **Initialization Vector (IV)**, this value may appear at the start of the encrypted portion. Although the IV is placed within the payload region, it is typically **not encrypted**, yet it is still considered part of the overall ciphertext structure.

5

The **Integrity Check Value (ICV)** field is optional. It appears only when the integrity service is enabled, and the protection can be provided either by a **separate integrity algorithm** or by certain **combined mode algorithms** that also output an ICV. Importantly, the ICV is always computed **after** encryption. This ordering enables the receiver to quickly identify and reject **replayed** or **malformed** packets before performing decryption, helping reduce exposure to denial-of-service attempts. It also supports **parallel processing** at the receiver, where decryption and integrity verification can proceed concurrently.

Since the ICV is transmitted **without encryption**, it must always be produced using a **keyed integrity algorithm** to prevent tampering by an attacker.

Padding

The **Padding** field in ESP is included for multiple reasons. When an encryption algorithm demands that the plaintext be an exact multiple of a specific byte size—such as a full block in block cipher modes—the padding extends the plaintext accordingly. This plaintext includes the **Payload Data**, **Padding**, **Pad Length**, and **Next Header** fields, all of which must align to the encryption algorithm's requirements.

ESP also specifies that the **Pad Length** and **Next Header** values must be positioned so they end on a **32-bit boundary**. To meet this requirement, the final encrypted block (ciphertext) must be an integer multiple of 32 bits, and padding is used to achieve the proper alignment.

Beyond technical alignment, padding may also be added intentionally to obscure the true size of the payload. This provides a degree of **traffic-flow confidentiality**, making it more difficult for an observer to infer meaningful information from packet length patterns.

Anti-Replay Service

A **replay attack** occurs when an attacker captures a legitimate, authenticated packet and later retransmits it to the original destination. Even though the packet is genuine, receiving it again can disrupt operations or produce unwanted effects. The **Sequence Number** field in ESP/AH is the primary mechanism that guards against such attacks.

When a new **Security Association (SA)** is created, the sender begins with a sequence number counter set to **0**. For every outgoing packet, this counter is incremented and the resulting value is placed in the **Sequence Number** field, beginning with **1**. If anti-replay protection is active (which is the default), the sequence number must never wrap around after reaching $\mathbf{2^{32} - 1}$.

Allowing it to roll back to zero would result in two different packets sharing the same sequence number. When the limit is reached, the sender must end the SA and establish a fresh one with new keys.

Since IP does not guarantee ordered or reliable delivery, the receiver uses a **sliding window** of size **W = 64** by default. The rightmost edge of this window corresponds to **N**, the highest sequence number seen so far in a valid, authenticated packet. For every correctly received packet within the window (from **N – W + 1** to **N**), its position in the window is marked.

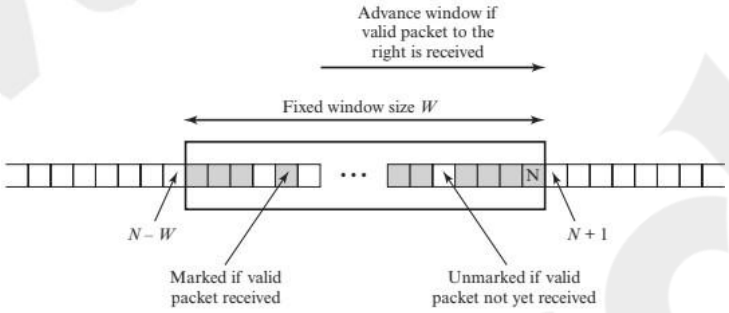


Figure 20.6 Anti-replay Mechanism

When a new packet arrives, the receiver follows three checks:

1. If the sequence number is **within the window** and hasn't been seen before, the receiver verifies its MAC. A valid packet results in its slot being marked.
2. If the sequence number is **greater than N** (to the right of the window), the MAC is checked. If valid, the receiver **slides the window forward** so that this new sequence value becomes the new right edge, and marks its slot.
3. If the sequence number is **to the left of the window**, or the authentication fails, the packet is discarded and the event is logged.

Transport and Tunnel Modes

IPsec ESP can protect traffic in two distinct ways, depending on how the communication path is organized. When two hosts directly secure their communication, encryption (and optionally authentication) is applied end-to-end between them. This corresponds to **transport mode**. When security is applied between gateways to create protected paths across an untrusted network such as the Internet, the setup forms a **virtual private network (VPN)**. This is achieved through **tunnel mode**. In such a configuration, internal hosts rely on their security gateways, rather than implementing IPsec themselves.

5

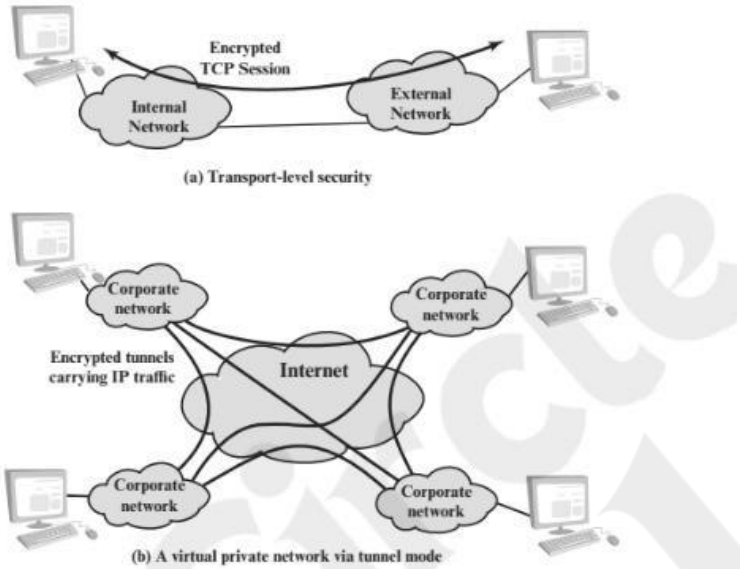


Figure 20.7 Transport-Mode versus Tunnel-Mode Encryptionx

The packet arrangements shown in Figure 20.8 give a basis for understanding how ESP fits into both IPv4 and IPv6. The behavior differs slightly between the two protocols because of the structure of their headers.

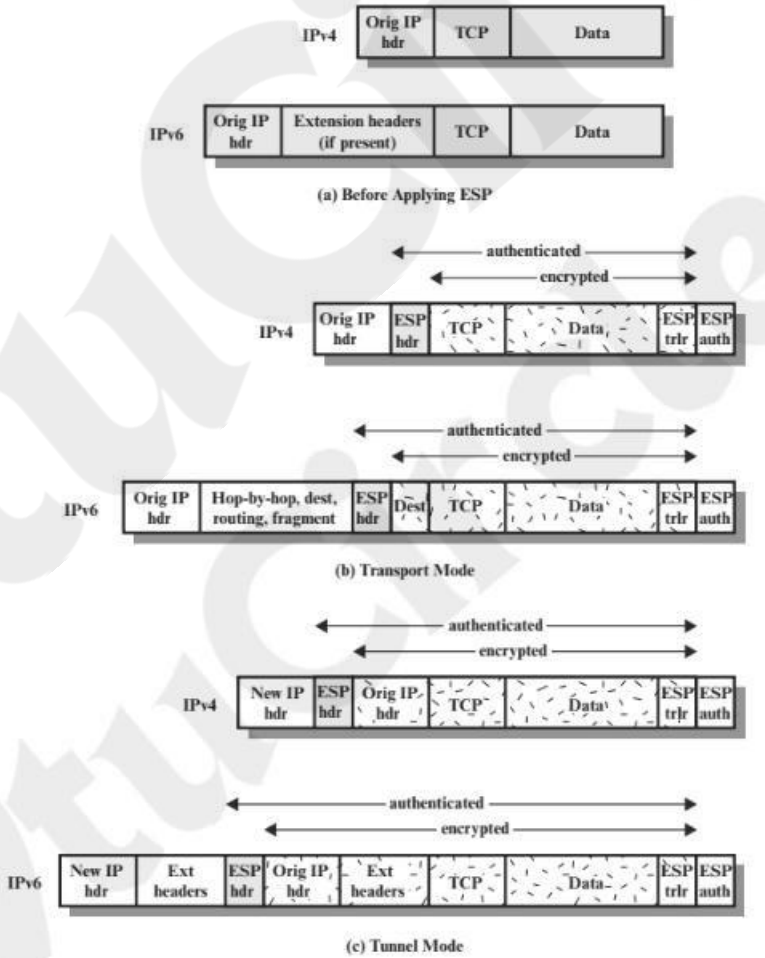


Figure 20.8 Scope of ESP Encryption and Authentication

5

Transport Mode ESP

Transport mode focuses on protecting the **payload** carried by IP—typically a TCP, UDP, or ICMP segment. In IPv4, the ESP header is placed just before the transport-layer header, and the ESP trailer (containing padding, pad length, and next-header information) follows the encrypted block. If authentication is selected, an Authentication Data field comes after the trailer. Everything from the transport-layer segment through the ESP trailer is encrypted. Authentication, when used, covers this ciphertext and the ESP header.

For IPv6, ESP appears after the IPv6 base header and after any hop-by-hop, routing, or fragmentation extension headers. The destination options extension header may appear before or after the ESP header depending on the intended behavior. Encryption in IPv6 covers the transport-layer segment, the ESP trailer, and the destination options header if it appears after ESP. Authentication again spans the ciphertext and the ESP header. Operation in transport mode proceeds as follows. At the sender, the transport-layer segment along with the ESP trailer is encrypted, and the resulting ciphertext replaces the original plaintext. Authentication, if enabled, is added after encryption. The packet then travels through the network. Routers along the path inspect only the IP header and any plaintext extension headers; they do not need to access the protected payload. When the packet reaches the final destination, the node processes the IP header, consults the SPI in the ESP header, and then decrypts the ciphertext to recover the original transport-layer data.

This mode offers confidentiality in a clean, end-to-end manner without placing that burden on every individual application. However, the outer IP header remains visible, which means that traffic patterns can still be observed—making traffic analysis possible.

Tunnel Mode ESP

Tunnel mode uses ESP to *encrypt the entire IP packet*. A fresh ESP header is added before the original packet, and then this entire block — ESP header, the original IP packet, the ESP trailer, and optional Authentication Data — is encrypted. This approach prevents outsiders from learning anything about the original IP header, helping defend against **traffic analysis**.

Because routers still need routing information, the encrypted block cannot be sent as-is. An encrypted IP header would hide the destination address, making routing impossible. To solve this, once encryption is done, a **new outer IP header** is added. This outer header carries just enough information for routing across the Internet, while the inner header remains hidden.

This mode is especially suitable when a firewall or security gateway protects an internal network. Hosts inside the trusted network do not need to perform encryption themselves. Instead, encryption happens only between the external host and the security gateway, or between two gateways. This reduces the key-management complexity and shields the internal hosts from the computation involved in cryptography. It also conceals the ultimate destination within the internal network, further limiting what attackers can infer from packet flows.

When an external host communicates with an internal host through a firewall using tunnel-mode ESP, the process unfolds as follows. The sender first builds an **inner IP packet** addressed to the internal target. An ESP header is placed before this inner packet, and then the packet plus ESP trailer are encrypted. Authentication Data may also be added. This encrypted block is wrapped inside a **new outer IP header**, whose destination address is the firewall. This combined structure forms the outer IP packet for transmission.

Routers on the Internet process only the outer IP header and any associated extension headers. They do not access or decrypt the enclosed ciphertext. When the packet reaches the firewall, it processes the outer header and then uses the SPI in the ESP header to decrypt the encrypted block, retrieving the plain inner IP packet. This inner packet is then forwarded within the internal network until it reaches the intended internal host.

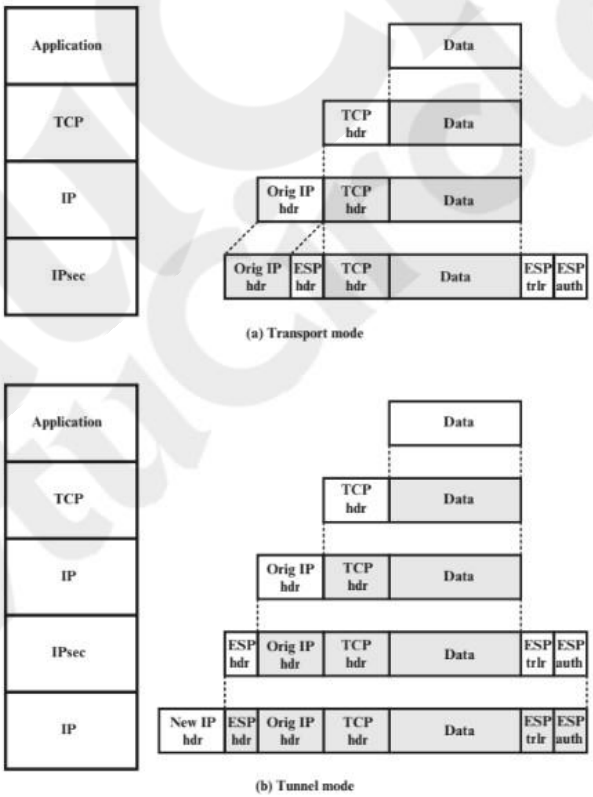


Figure 20.9 Protocol Operation for ESP

5

Tunnel mode is thus an effective way to build **VPN-style secure connections**, ensuring privacy, authentication (if selected), and protection against traffic analysis.

Combining Security Associations

A single Security Association (SA) supports only one IPsec protocol at a time—either AH or ESP, but not both together. Yet some traffic flows need **authentication plus confidentiality**, while others require protection at multiple points, such as host-to-host and gateway-to-gateway. To provide these combined services, multiple SAs are used together. A sequence of such SAs is called a **security association bundle**, and the endpoints of the individual SAs in the bundle may be the same or different.

There are two main ways to combine SAs in a bundle.

Transport adjacency, more than one security protocol is applied to a single IP packet without tunneling. AH and ESP can be combined once in this manner, but additional nesting is pointless because all processing happens at the same final IPsec instance.

Iterated tunneling builds multiple layers of IPsec protection through tunneling. Each layer can begin or end at a different IPsec-capable node along the path, allowing deeper nesting and more flexible security structures. These methods can also be mixed; for example, a transport-mode SA between two hosts may pass through a tunnel-mode SA that exists between two firewalls.

When combining services within a single pair of endpoints, one must consider the order of applying **authentication and encryption**, as it affects what is protected, what is visible to routers, and how replay protection is handled. Before exploring combinations involving tunnels, it is useful to look at how both services operate together between the same endpoints.

Authentication Plus Confidentiality

IPsec supports several approaches for providing both authentication and confidentiality for the same traffic. One common method uses **ESP with its authentication option**, as shown in the ESP format. ESP is first applied to the data, providing encryption and optional integrity for the payload, and then an authentication field is appended.

Two variations arise depending on the mode:

- In **transport-mode ESP**, both authentication and confidentiality apply only to the IP payload (the transport-layer segment). The original IP header itself remains exposed and is not protected

5

by either service.

- In **tunnel-mode ESP**, authentication is applied to the entire inner IP packet. The outer header, added for routing purposes, remains visible to routers, but the complete original packet — including its original IP header — is hidden and protected as it travels to the outer destination, such as a firewall.

  In both transport and tunnel modes, the authentication function covers the **ciphertext**, not the plaintext. This ensures that any tampering with the encrypted data can still be detected without exposing the original content.

Transport Adjacency

Another method to apply **authentication after encryption** is by using **two bundled transport SAs**:

- **Inner SA → ESP (without authentication)**: encrypts only the **IP payload**.
- **Outer SA → AH (transport mode)**: authenticates the **ESP + original IP header** (except mutable fields).

  This gives stronger protection because **authentication covers more fields**, including **source and destination IP addresses**, which ESP alone does not protect.

  The drawback is the **overhead of maintaining two SAs** instead of one.

Transport–Tunnel Bundle

Sometimes authentication is preferred **before encryption**:

- Authentication data becomes **protected by encryption**, preventing tampering.
- Easier to **store authentication info** with the message for later verification.
- Better when authentication should apply to the **plaintext** rather than ciphertext.

A common method is to use:

- **Inner SA → AH (transport mode)**: authenticates **IP payload + IP header** (except mutable fields).
- **Outer SA → ESP (tunnel mode)**: encrypts the **entire authenticated inner packet** and adds a **new outer IP header**.

5

Basic Combinations of Security Associations

IPsec allows multiple **Security Associations (SAs)** to be combined for authentication and confidentiality. Each SA can use **AH** or **ESP**; host-to-host SAs may use **transport or tunnel mode**, while gateway SAs must use **tunnel mode**.

Case 1: End-to-End Security Between Hosts

Both hosts implement IPsec and share **secret keys**. Possible combinations include **AH (transport)**, **ESP (transport)**, **ESP inside AH**, or any of these inside an **outer tunnel SA**. These support **authentication**, **encryption**, and ordered combinations like **auth-before-encrypt** or **auth-after-encrypt**.

Case 2: Gateway-to-Gateway Security (VPN)

Hosts do not use IPsec; only gateways protect traffic. A **single tunnel-mode SA** (AH, ESP, or ESP+auth) is enough because the **entire inner packet** is secured. This is the basic **VPN model**.

Case 3: Gateway Security + End-to-End Security

A **gateway-to-gateway tunnel SA** protects all traffic between networks. Individual hosts may add **end-to-end SAs** for extra confidentiality or authentication. ESP tunneling also offers limited **traffic-flow confidentiality**.

Case 4: Remote Access Through Firewall

A remote host connects securely to an organization's internal network using a **tunnel-mode SA** with the firewall. This enables secure access to internal servers, similar to **remote-access VPNs**.

5

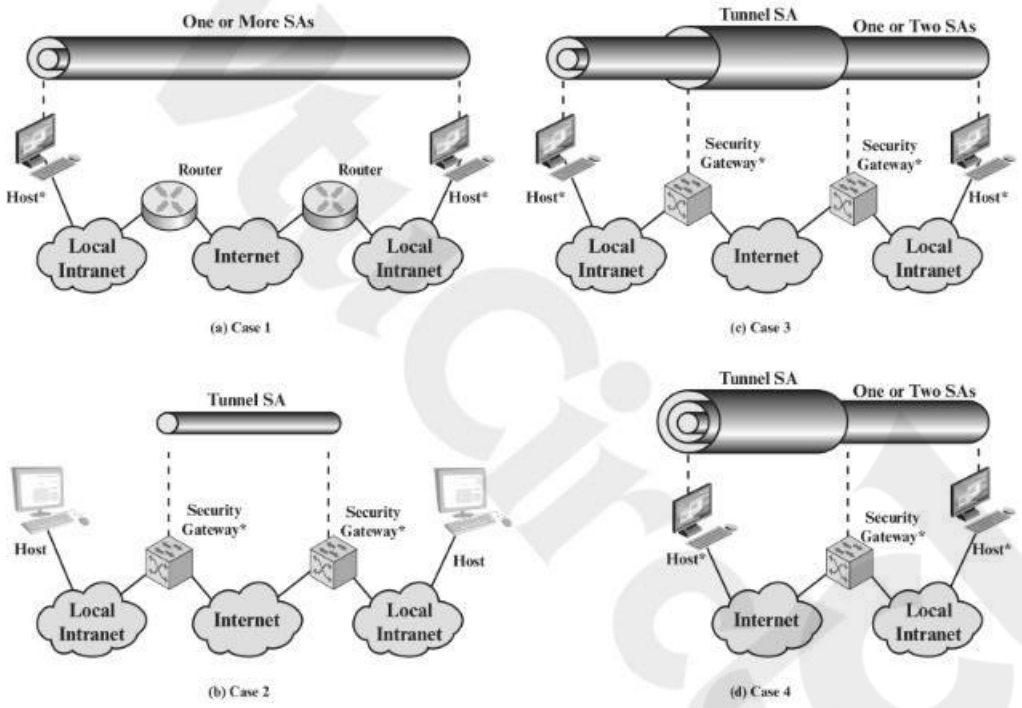


Figure 20.10 Basic Combinations of Security Associations

Internet Key Exchange

Internet Key Exchange (**IKE**) is the part of **IPsec** responsible for **key management**. Its role is to determine and distribute **secret keys** required for secure communication. Typically, two applications need **four keys**—a **transmit** and **receive** pair each for **integrity** and **confidentiality**.

IPsec supports two main forms of key management.

- **manual** key management, the administrator configures all **keys** directly on each system, which works only for **small** and **static** environments. In contrast,

- **automated** key management allows on-demand **key creation**, supporting **large, dynamic** networks by managing **Security Associations (SAs)** efficiently.

The traditional automated approach in early IPsec used **ISAKMP/Oakley**.

The **Oakley** component provides a secure **key exchange** mechanism based on **Diffie–Hellman**, offering additional protections but staying flexible by not fixing message formats.

5

ISAKMP(Internet Security Association and Key Management Protocol provides the **framework** and **message structures** needed to negotiate **security attributes**. Although ISAKMP does not bind to any particular exchange algorithm, **Oakley** was mandated for the initial ISAKMP version.

With **IKEv2**, the names **Oakley** and **ISAKMP** are no longer used. The protocol design is simplified and improved, though the **core purpose**—secure creation and management of keys for IPsec—remains unchanged.

Key Determination Protocol

IKE uses a refined version of the **Diffie–Hellman key exchange** to securely generate shared keys between two communicating parties. In the basic Diffie–Hellman method, both users agree beforehand on two global values: a large prime **q** and a **primitive root a** of q. Each party chooses a **private key** (X_A for A and X_B for B) and computes its corresponding **public key** using the formula $Y_A = \alpha^{X_A} \bmod q.$ and $Y_B = \alpha^{X_B} \bmod q.$ After public keys YA and YB are exchanged, both sides independently calculate the **same session key**, , ensuring that the shared key is never transmitted directly.

$$K = (Y_B)^{X_A} \bmod q = (Y_A)^{X_B} \bmod q = \alpha^{X_A X_B} \bmod q$$

This approach has two strong advantages. **Keys are generated only when required**, reducing long-term exposure, and there is **no need for prior infrastructure**, other than agreeing on the parameters q and a.

However, pure Diffie–Hellman also presents critical weaknesses.

- It provides **no identity information**, so the communicating parties cannot be sure who they are talking to.
- It is also vulnerable to a **man-in-the-middle attack**, in which an attacker intercepts and replaces public keys, tricking both sides into establishing separate keys with the attacker.
- This allows the attacker to read and modify all exchanged data without detection. Another issue is the algorithm's **computational intensity**, making it vulnerable to **clogging attacks** where an adversary forces a system to perform many expensive exponentiations, consuming resources unnecessarily.

5

The IKE key determination process addresses these problems while keeping the strengths of Diffie–Hellman. It **preserves on-demand key creation** and avoids long-term key storage, while adding mechanisms for **authentication**, preventing man-in-the-middle interference, and reducing the risk of computational overload.

FEATURES OF IKE KEY DETERMINATION

The **IKE key determination algorithm** strengthens the basic Diffie–Hellman approach through five essential features:

- **Cookies** are used to prevent **clogging attacks**.
- Both parties **negotiate a group**, which defines the **global parameters** for the Diffie–Hellman exchange.
- **Nonces** are included to protect against **replay attacks**.
- The protocol supports secure **exchange of Diffie–Hellman public keys**.
- The entire exchange is **authenticated** to prevent **man-in-the-middle attacks**.

To understand these features, consider how IKE handles clogging attacks. In such an attack, an adversary sends forged Diffie–Hellman public keys, forcing the victim to perform expensive **modular exponentiation** repeatedly. IKE prevents this by introducing **cookie exchange**, where each side sends a **pseudorandom cookie** and expects it to be returned before performing any heavy computation. If the source address was forged, the attacker receives no reply, limiting the victim's workload to cheap acknowledgments.

Requirements for cookie generation include:

- The cookie must depend on the **specific communicating parties**, preventing reuse from unrelated addresses.
- Only the **issuing entity** should be able to create valid cookies, relying on a **local secret** that cannot be derived.
- Cookie generation and verification must be **fast**, preventing resource-draining attacks.

A practical method is a fast **hash function** (e.g., **MD5**) over source/destination IPs, ports, and a local secret.

IKE allows negotiation of different **Diffie–Hellman groups**, which define the parameters and algorithm type. These include:

- **Modular exponentiation groups** with **768-bit**, **1024-bit**, and **1536-bit** moduli.

5

- Modular exponentiation with a 768-bit modulus

$$q = 2^{768} - 2^{704} - 1 + 2^{64} \times (\lfloor 2^{638} \times \pi \rfloor + 149686)$$
$$\alpha = 2$$

- Modular exponentiation with a 1024-bit modulus

$$q = 2^{1024} - 2^{960} - 1 + 2^{64} \times (\lfloor 2^{894} \times \pi \rfloor + 129093)$$
$$\alpha = 2$$

- Modular exponentiation with a 1536-bit modulus
  - Parameters to be determined
- Elliptic curve group over 2^{155}
  - Generator (hexadecimal): X = 7B, Y = 1C8
  - Elliptic curve parameters (hexadecimal): A = 0, Y = 7338F
- Elliptic curve group over 2^{185}
  - Generator (hexadecimal): X = 18, Y = D
  - Elliptic curve parameters (hexadecimal): A = 0, Y = 1EE9

- **Elliptic curve groups**, including curves over $\mathbf{2^{155}}$ and $\mathbf{2^{185}}$, each with defined **generators** and **curve parameters**.

  Classic modular exponentiation corresponds to traditional Diffie–Hellman, while elliptic curve groups implement the **EC-Diffie–Hellman** variant.

IKE also uses **nonces**, which are locally generated **pseudorandom numbers**. These protect the protocol from **replay**, and are encrypted during parts of the exchange for security.

To authenticate the Diffie–Hellman process and prevent man-in-the-middle attacks, IKE supports three methods:

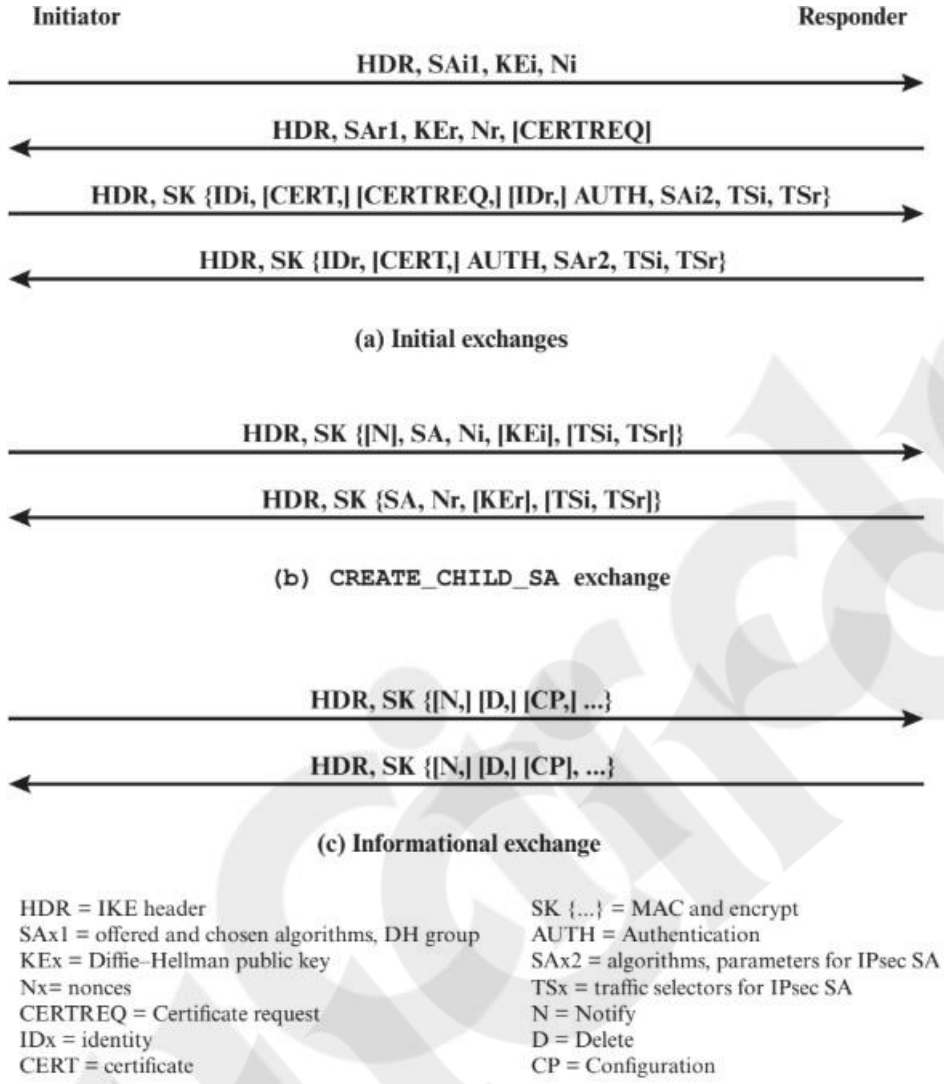
- **Digital signatures**: each party signs a shared hash using its **private key**.
- **Public-key encryption**: sensitive fields like IDs and nonces are encrypted with the sender's **private key**.
- **Symmetric-key encryption**: an externally derived shared key encrypts exchange parameters for authentication.

IKEV2 EXCHANGES

IKEv2 operates through a series of **paired message exchanges**, with the first two pairs known as the **initial exchanges**.

5

These initial exchanges establish the secure foundations required for IPsec communication.



Initial Exchange – Pair 1

- Both peers share their supported **cryptographic algorithms**, **security parameters**, **nonces**, and **Diffie–Hellman (DH) values**.
- The outcome of this first exchange is the creation of the **IKE SA (Security Association)**.
- The **IKE SA** defines the parameters for a **secure, authenticated channel** between the peers.
- All exchanges after this step occur **within this protected channel**, using **encryption** and **message authentication**.

Initial Exchange – Pair 2

5

- The peers **authenticate each other** using configured authentication methods.
- They establish the **first IPsec SA**, which is stored in the **SADB**.
- This IPsec SA is used for securing **normal data traffic** (i.e., **non-IKE communication**).
- In total, **four messages** (two pairs) are required to establish the first IPsec SA for general use.

CREATE_CHILD_SA Exchange

- Used to set up **additional IPsec SAs** when needed.
- Supports tasks such as:
    - Creating fresh SAs for new traffic flows
    - Rekeying existing SAs
    - Updating cryptographic keys

Informational Exchange

- Used for exchanging **management information**, **error messages**, and **status notifications**.
- Does not establish new SAs, but helps maintain the session by handling:
    - **IKEv2 errors**
    - **Delete notifications**
    - **Configuration updates**

IKE Header and Payload Formats

The **Internet Key Exchange (IKE)** protocol defines the **procedures** and **packet formats** required to establish, negotiate, modify, and delete **Security Associations (SAs)**. As part of SA setup, IKE uses a variety of **payloads** to exchange **key generation** and **authentication** data. These payload formats form a **consistent and flexible framework**, independent of the specific **key exchange**, **encryption**, or **authentication** method used.

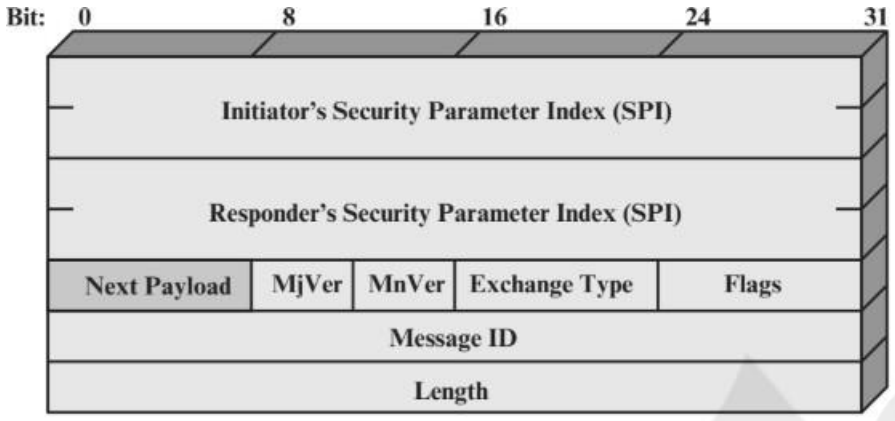


Figure 20.12 IKE Formats

IKE Header Format

An **IKE message** is made up of an **IKE header** followed by one or more **payloads**. The entire message is carried over a transport protocol, and IKE requires support for **UDP** as the default transport.

The IKE header contains the following fields:

- **Initiator SPI (64 bits)**
  A value generated by the **initiator** to uniquely identify an **IKE SA**.
- **Responder SPI (64 bits)**
  A value generated by the **responder** to uniquely identify the same **IKE SA**.
- **Next Payload (8 bits)**
  Indicates the **type of the first payload** in the packet. Additional payloads follow in sequence.
- **Major Version (4 bits)**
  Specifies the **IKE major version** being used.
- **Minor Version (4 bits)**
  Specifies the **minor version** of IKE.
- **Exchange Type (8 bits)**
  Identifies the **exchange type**, such as **IKE_SA_INIT**, **IKE_AUTH**, **CREATE_CHILD_SA**, or

5

INFORMATIONAL.

- **Flags (8 bits)**

  Includes bits indicating:

  - **Initiator bit** – whether the sender is the initiator
  - **Version bit** – whether the sender supports a higher version
  - **Response bit** – whether this message is a **response** to a previous message
- **Message ID (32 bits)**

  Used for **retransmission control** and for matching **request–response pairs**.
- **Length (32 bits)**

  Total length of the **complete IKE message**, including header and all payloads, in **octets**.

IKE Payload Types

Every IKE payload begins with a **generic payload header** containing:

- **Next Payload** – type of the following payload (0 if last).
- **Payload Length** – total size of the payload including the header.
- **Critical Bit** –
  - **0** → ignore the payload if not understood
  - **1** → reject the entire message if the payload type is not understood

IKE defines multiple payload types, each carrying specific information needed for **SA establishment** and **key management**.

SA Payload

Used to start creation of a **Security Association**. It has a hierarchical internal structure:

- **Proposal** – contains proposal number, **protocol ID** (AH, ESP, IKE), number of transforms.
- **Transform** – specifies **cryptographic algorithms** for that protocol.
- **Attribute** – provides additional parameters such as **key length**.

Key Exchange Payload

Supports different key exchange methods like **Oakley**, **Diffie–Hellman**, and **RSA-based methods (PGP style)**.

- Contains the key exchange data necessary for generating a **session key**.

Identification Payload

Used to identify communicating peers.

- **ID Data** typically includes **IPv4/IPv6 address**, domain name, or other identity information.
- Helps in **authentication**.

Certificate Payload

Transfers **public-key certificates**.

The **Certificate Encoding** field specifies certificate type, such as:

- **PKCS #7 wrapped X.509**
- **PGP certificate**
- **DNS signed key**
- **X.509 (signature or key exchange)**
- **Kerberos tokens**
- **CRL / ARL**
- **SPKI certificate**

Certificate Request Payload

Requests certificates from the other peer.

- Can list multiple acceptable **certificate types** and **certificate authorities**.

Authentication Payload

Contains data used to authenticate the exchange.

Supports methods such as:

- **RSA digital signature**
- **Shared-key MAC**
- **DSS digital signature**

Nonce Payload

Carries **random values** to ensure **liveness** and protect against **replay attacks**.

Notify Payload

Carries **error** or **status information** related to an SA or negotiation. Examples include invalid SPI, authentication failure, or unsupported features.

Delete Payload

Indicates one or more **SAs that have been deleted** from the sender's database.

- Ensures both peers remove invalid or expired SAs.

Vendor ID Payload

Contains a vendor-defined constant used to identify **vendor-specific implementations**.

- Enables experimentation with new features while retaining compatibility.

Traffic Selector Payload

Specifies **packet flows** that should be protected by IPsec services. Examples: specific IP ranges, ports, or protocols.

Encrypted Payload

Wraps other payloads in **encrypted form**.

- Similar to **ESP format**
- May include **IV** (for encryption) and **ICV** (for authentication)

Configuration Payload

Used to exchange configuration data such as:

- IP addresses
- DNS information
- Other network settings

EAP Payload

Supports authentication via **Extensible Authentication Protocol (EAP)**.

Allows integration with advanced authentication systems.

5

Table 20.3 IKE Payload Types

Type	Parameters
Security Association	Proposals
Key Exchange	DH Group #, Key Exchange Data
Identification	ID Type, ID Data
Certificate	Cert Encoding, Certificate Data
Certificate Request	Cert Encoding, Certification Authority
Authentication	Auth Method, Authentication Data
Nonce	Nonce Data
Notify	Protocol-ID, SPI Size, Notify Message Type, SPI, Notification Data
Delete	Protocol-ID, SPI Size, # of SPIs, SPI (one or more)
Vendor ID	Vendor ID
Traffic Selector	Number of TSs, Traffic Selectors
Encrypted	IV, Encrypted IKE payloads, Padding, Pad Length, ICV
Configuration	CFG Type, Configuration Attributes
Extensible Authentication Protocol	EAP Message

The following table lists the IKE notify messages.

Error Messages	Status Messages
Unsupported Critical Payload	Initial Contact
Invalid IKE SPI	Set Window Size
Invalid Major Version	Additional TS Possible
Invalid Syntax	IPCOMP Supported
Invalid Payload Type	NAT Detection Source IP
Invalid Message ID	NAT Detection Destination IP
Invalid SPI	Use Transport Mode

Error Messages	Status Messages
No Proposal Chosen	HTTP Cert Lookup Supported
Invalid KE Payload	Rekey SA
Authentication Failed	ESP TFC Padding Not Supported
Single Pair Required	Non First Fragments Also
No Additional SAS	
Internal Address Failure	
Failed CP Required	
TS Unacceptable	
Invalid Selectors	