

with its ciphertext to form the IP packet for transmission. Authentication is added if this option is selected.

2. The packet is then routed to the destination. Each intermediate router needs to examine and process the IP header plus any plaintext IP extension headers but does not need to examine the ciphertext.
3. The destination node examines and processes the IP header plus any plaintext IP extension headers. Then, on the basis of the SPI in the ESP header, the destination node decrypts the remainder of the packet to recover the plaintext transport-layer segment.

Transport mode operation provides confidentiality for any application that uses it, thus avoiding the need to implement confidentiality in every individual application. One drawback to this mode is that it is possible to do traffic analysis on the transmitted packets.

TUNNEL MODE ESP Tunnel mode ESP is used to encrypt an entire IP packet (Figure 9.8c). For this mode, the ESP header is prefixed to the packet and then the packet plus the ESP trailer is encrypted. This method can be used to counter traffic analysis.

Because the IP header contains the destination address and possibly source routing directives and hop-by-hop option information, it is not possible simply to transmit the encrypted IP packet prefixed by the ESP header. Intermediate routers would be unable to process such a packet. Therefore, it is necessary to encapsulate the entire block (ESP header plus ciphertext plus Authentication Data, if present) with a new IP header that will contain sufficient information for routing but not for traffic analysis.

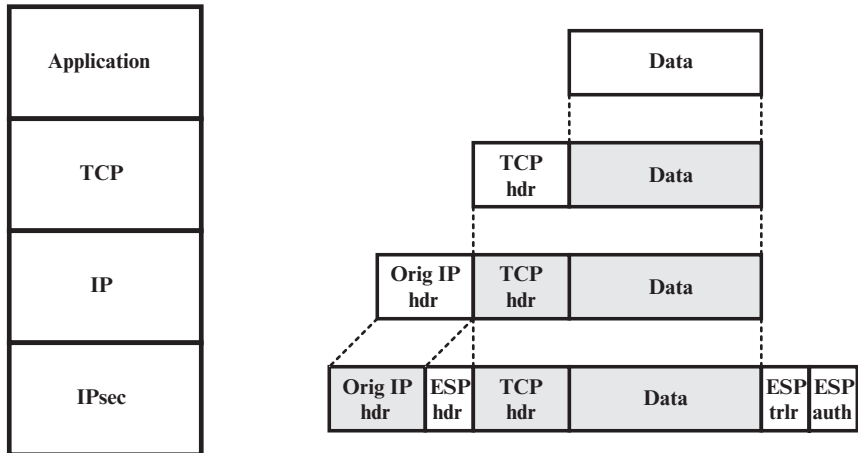
Whereas the transport mode is suitable for protecting connections between hosts that support the ESP feature, the tunnel mode is useful in a configuration that includes a firewall or other sort of security gateway that protects a trusted network from external networks. In this latter case, encryption occurs only between an external host and the security gateway or between two security gateways. This relieves hosts on the internal network of the processing burden of encryption and simplifies the key distribution task by reducing the number of needed keys. Further, it thwarts traffic analysis based on ultimate destination.

Consider a case in which an external host wishes to communicate with a host on an internal network protected by a firewall, and in which ESP is implemented in the external host and the firewalls. The following steps occur for transfer of a transport-layer segment from the external host to the internal host.

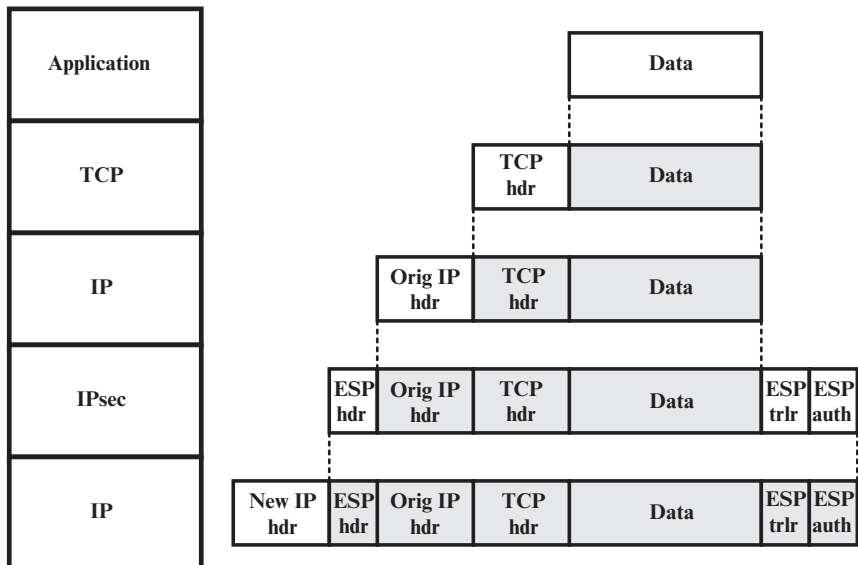
1. The source prepares an inner IP packet with a destination address of the target internal host. This packet is prefixed by an ESP header; then the packet and ESP trailer are encrypted and Authentication Data may be added. The resulting block is encapsulated with a new IP header (base header plus optional extensions such as routing and hop-by-hop options for IPv6) whose destination address is the firewall; this forms the outer IP packet.
2. The outer packet is routed to the destination firewall. Each intermediate router needs to examine and process the outer IP header plus any outer IP extension headers but does not need to examine the ciphertext.

3. The destination firewall examines and processes the outer IP header plus any outer IP extension headers. Then, on the basis of the SPI in the ESP header, the destination node decrypts the remainder of the packet to recover the plaintext inner IP packet. This packet is then transmitted in the internal network.
4. The inner packet is routed through zero or more routers in the internal network to the destination host.

Figure 9.9 shows the protocol architecture for the two modes.



(a) Transport mode



(b) Tunnel mode

Figure 9.9 Protocol Operation for ESP

9.4 COMBINING SECURITY ASSOCIATIONS

An individual SA can implement either the AH or ESP protocol but not both. Sometimes a particular traffic flow will call for the services provided by both AH and ESP. Further, a particular traffic flow may require IPsec services between hosts and, for that same flow, separate services between security gateways, such as firewalls. In all of these cases, multiple SAs must be employed for the same traffic flow to achieve the desired IPsec services. The term *security association bundle* refers to a sequence of SAs through which traffic must be processed to provide a desired set of IPsec services. The SAs in a bundle may terminate at different endpoints or at the same endpoints.

Security associations may be combined into bundles in two ways:

- **Transport adjacency:** Refers to applying more than one security protocol to the same IP packet without invoking tunneling. This approach to combining AH and ESP allows for only one level of combination; further nesting yields no added benefit since the processing is performed at one IPsec instance: the (ultimate) destination.
- **Iterated tunneling:** Refers to the application of multiple layers of security protocols effected through IP tunneling. This approach allows for multiple levels of nesting, since each tunnel can originate or terminate at a different IPsec site along the path.

The two approaches can be combined, for example, by having a transport SA between hosts travel part of the way through a tunnel SA between security gateways.

One interesting issue that arises when considering SA bundles is the order in which authentication and encryption may be applied between a given pair of endpoints and the ways of doing so. We examine that issue next. Then we look at combinations of SAs that involve at least one tunnel.

Authentication Plus Confidentiality

Encryption and authentication can be combined in order to transmit an IP packet that has both confidentiality and authentication between hosts. We look at several approaches.

ESP WITH AUTHENTICATION OPTION This approach is illustrated in Figure 9.8. In this approach, the user first applies ESP to the data to be protected and then appends the authentication data field. There are actually two subcases:

- **Transport mode ESP:** Authentication and encryption apply to the IP payload delivered to the host, but the IP header is not protected.
- **Tunnel mode ESP:** Authentication applies to the entire IP packet delivered to the outer IP destination address (e.g., a firewall), and authentication is performed at that destination. The entire inner IP packet is protected by the privacy mechanism for delivery to the inner IP destination.

For both cases, authentication applies to the ciphertext rather than the plaintext.

TRANSPORT ADJACENCY Another way to apply authentication after encryption is to use two bundled transport SAs, with the inner being an ESP SA and the outer being an AH SA. In this case, ESP is used without its authentication option. Because the inner SA is a transport SA, encryption is applied to the IP payload. The resulting packet consists of an IP header (and possibly IPv6 header extensions) followed by an ESP. AH is then applied in transport mode, so that authentication covers the ESP plus the original IP header (and extensions) except for mutable fields. The advantage of this approach over simply using a single ESP SA with the ESP authentication option is that the authentication covers more fields, including the source and destination IP addresses. The disadvantage is the overhead of two SAs versus one SA.

TRANSPORT-TUNNEL BUNDLE The use of authentication prior to encryption might be preferable for several reasons. First, because the authentication data are protected by encryption, it is impossible for anyone to intercept the message and alter the authentication data without detection. Second, it may be desirable to store the authentication information with the message at the destination for later reference. It is more convenient to do this if the authentication information applies to the unencrypted message; otherwise the message would have to be reencrypted to verify the authentication information.

One approach to applying authentication before encryption between two hosts is to use a bundle consisting of an inner AH transport SA and an outer ESP tunnel SA. In this case, authentication is applied to the IP payload plus the IP header (and extensions) except for mutable fields. The resulting IP packet is then processed in tunnel mode by ESP; the result is that the entire, authenticated inner packet is encrypted and a new outer IP header (and extensions) is added.

Basic Combinations of Security Associations

The IPsec Architecture document lists four examples of combinations of SAs that must be supported by compliant IPsec hosts (e.g., workstation, server) or security gateways (e.g., firewall, router). These are illustrated in Figure 9.10. The lower part of each case in the figure represents the physical connectivity of the elements; the upper part represents logical connectivity via one or more nested SAs. Each SA can be either AH or ESP. For host-to-host SAs, the mode may be either transport or tunnel; otherwise it must be tunnel mode.

Case 1. All security is provided between end systems that implement IPsec. For any two end systems to communicate via an SA, they must share the appropriate secret keys. Among the possible combinations are

- a. AH in transport mode
- b. ESP in transport mode
- c. ESP followed by AH in transport mode (an ESP SA inside an AH SA)
- d. Any one of a, b, or c inside an AH or ESP in tunnel mode

We have already discussed how these various combinations can be used to support authentication, encryption, authentication before encryption, and authentication after encryption.

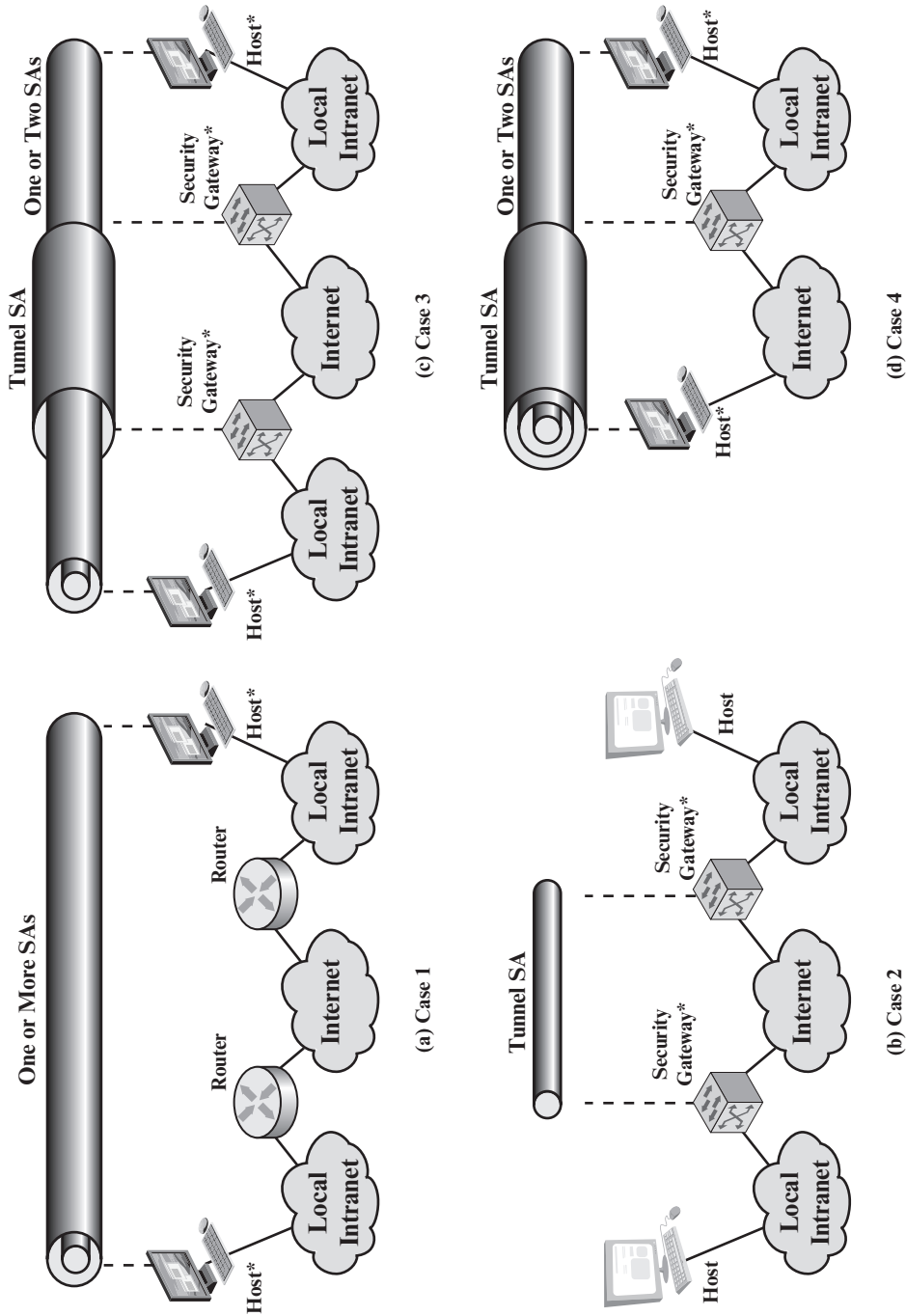


Figure 9.10 Basic Combinations of Security Associations

Case 2. Security is provided only between gateways (routers, firewalls, etc.) and no hosts implement IPsec. This case illustrates simple virtual private network support. The security architecture document specifies that only a single tunnel SA is needed for this case. The tunnel could support AH, ESP, or ESP with the authentication option. Nested tunnels are not required, because the IPsec services apply to the entire inner packet.

Case 3. This builds on case 2 by adding end-to-end security. The same combinations discussed for cases 1 and 2 are allowed here. The gateway-to-gateway tunnel provides either authentication, confidentiality, or both for all traffic between end systems. When the gateway-to-gateway tunnel is ESP, it also provides a limited form of traffic confidentiality. Individual hosts can implement any additional IPsec services required for given applications or given users by means of end-to-end SAs.

Case 4. This provides support for a remote host that uses the Internet to reach an organization's firewall and then to gain access to some server or workstation behind the firewall. Only tunnel mode is required between the remote host and the firewall. As in case 1, one or two SAs may be used between the remote host and the local host.

9.5 INTERNET KEY EXCHANGE

The key management portion of IPsec involves the determination and distribution of secret keys. A typical requirement is four keys for communication between two applications: transmit and receive pairs for both integrity and confidentiality. The IPsec Architecture document mandates support for two types of key management:

- **Manual:** A system administrator manually configures each system with its own keys and with the keys of other communicating systems. This is practical for small, relatively static environments.
- **Automated:** An automated system enables the on-demand creation of keys for SAs and facilitates the use of keys in a large distributed system with an evolving configuration.

The default automated key management protocol for IPsec is referred to as ISAKMP/Oakley and consists of the following elements:

- **Oakley Key Determination Protocol:** Oakley is a key exchange protocol based on the Diffie–Hellman algorithm but providing added security. Oakley is generic in that it does not dictate specific formats.
- **Internet Security Association and Key Management Protocol (ISAKMP):** ISAKMP provides a framework for Internet key management and provides the specific protocol support, including formats, for negotiation of security attributes.

ISAKMP by itself does not dictate a specific key exchange algorithm; rather, ISAKMP consists of a set of message types that enable the use of a variety of key exchange algorithms. Oakley is the specific key exchange algorithm mandated for use with the initial version of ISAKMP.

In IKEv2, the terms Oakley and ISAKMP are no longer used, and there are significant differences from the use of Oakley and ISAKMP in IKEv1. Nevertheless, the basic functionality is the same. In this section, we describe the IKEv2 specification.

Key Determination Protocol

IKE key determination is a refinement of the Diffie–Hellman key exchange algorithm. Recall that Diffie–Hellman involves the following interaction between users A and B. There is prior agreement on two global parameters: q , a large prime number; and α , a primitive root of q . A selects a random integer X_A as its private key and transmits to B its public key $Y_A = \alpha^{X_A} \bmod q$. Similarly, B selects a random integer X_B as its private key and transmits to A its public key $Y_B = \alpha^{X_B} \bmod q$. Each side can now compute the secret session key:

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

The Diffie–Hellman algorithm has two attractive features:

- Secret keys are created only when needed. There is no need to store secret keys for a long period of time, exposing them to increased vulnerability.
- The exchange requires no pre-existing infrastructure other than an agreement on the global parameters.

However, there are a number of weaknesses to Diffie–Hellman, as pointed out in [HUIT98].

- It does not provide any information about the identities of the parties.
- It is subject to a man-in-the-middle attack, in which a third party C impersonates B while communicating with A and impersonates A while communicating with B. Both A and B end up negotiating a key with C, which can then listen to and pass on traffic. The man-in-the-middle attack proceeds as
 1. B sends his public key Y_B in a message addressed to A (see Figure 3.14).
 2. The enemy (E) intercepts this message. E saves B's public key and sends a message to A that has B's User ID but E's public key Y_E . This message is sent in such a way that it appears as though it was sent from B's host system. A receives E's message and stores E's public key with B's User ID. Similarly, E sends a message to B with E's public key, purporting to come from A.
 3. B computes a secret key K_1 based on B's private key and Y_E . A computes a secret key K_2 based on A's private key and Y_E . E computes K_1 using E's secret key X_E and Y_B and computes K_2 using X_E and Y_A .
 4. From now on, E is able to relay messages from A to B and from B to A, appropriately changing their encipherment en route in such a way that neither A nor B will know that they share their communication with E.
- It is computationally intensive. As a result, it is vulnerable to a clogging attack, in which an opponent requests a high number of keys. The victim spends considerable computing resources doing useless modular exponentiation rather than real work.

IKE key determination is designed to retain the advantages of Diffie–Hellman, while countering its weaknesses.

FEATURES OF IKE KEY DETERMINATION The IKE key determination algorithm is characterized by five important features:

1. It employs a mechanism known as cookies to thwart clogging attacks.
2. It enables the two parties to negotiate a *group*; this, in essence, specifies the global parameters of the Diffie–Hellman key exchange.
3. It uses nonces to ensure against replay attacks.
4. It enables the exchange of Diffie–Hellman public key values.
5. It authenticates the Diffie–Hellman exchange to thwart man-in-the-middle attacks.

We have already discussed Diffie–Hellman. Let us look at the remainder of these elements in turn. First, consider the problem of clogging attacks. In this attack, an opponent forges the source address of a legitimate user and sends a public Diffie–Hellman key to the victim. The victim then performs a modular exponentiation to compute the secret key. Repeated messages of this type can *clog* the victim’s system with useless work. The **cookie exchange** requires that each side send a pseudorandom number, the cookie, in the initial message, which the other side acknowledges. This acknowledgment must be repeated in the first message of the Diffie–Hellman key exchange. If the source address was forged, the opponent gets no answer. Thus, an opponent can only force a user to generate acknowledgments and not to perform the Diffie–Hellman calculation.

IKE mandates that cookie generation satisfy three basic requirements:

1. The cookie must depend on the specific parties. This prevents an attacker from obtaining a cookie using a real IP address and UDP port and then using it to swamp the victim with requests from randomly chosen IP addresses or ports.
2. It must not be possible for anyone other than the issuing entity to generate cookies that will be accepted by that entity. This implies that the issuing entity will use local secret information in the generation and subsequent verification of a cookie. It must not be possible to deduce this secret information from any particular cookie. The point of this requirement is that the issuing entity need not save copies of its cookies, which are then more vulnerable to discovery, but can verify an incoming cookie acknowledgment when it needs to.
3. The cookie generation and verification methods must be fast to thwart attacks intended to sabotage processor resources.

The recommended method for creating the cookie is to perform a fast hash (e.g., MD5) over the IP Source and Destination addresses, the UDP Source and Destination ports, and a locally generated secret value.

IKE key determination supports the use of different groups for the Diffie–Hellman key exchange. Each group includes the definition of the two global parameters and the identity of the algorithm. The current specification includes the following groups.

- 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

The first three groups are the classic Diffie–Hellman algorithm using modular exponentiation. The last two groups use the elliptic curve analog to Diffie–Hellman.

IKE key determination employs **nonces** to ensure against replay attacks. Each nonce is a locally generated pseudorandom number. Nonces appear in responses and are encrypted during certain portions of the exchange to secure their use.

Three different **authentication** methods can be used with IKE key determination:

- **Digital signatures:** The exchange is authenticated by signing a mutually obtainable hash; each party encrypts the hash with its private key. The hash is generated over important parameters, such as user IDs and nonces.
- **Public-key encryption:** The exchange is authenticated by encrypting parameters such as IDs and nonces with the sender's private key.
- **Symmetric-key encryption:** A key derived by some out-of-band mechanism can be used to authenticate the exchange by symmetric encryption of exchange parameters.

IKEv2 EXCHANGES The IKEv2 protocol involves the exchange of messages in pairs. The first two pairs of exchanges are referred to as the **initial exchanges** (Figure 9.11a). In the first exchange, the two peers exchange information concerning cryptographic algorithms and other security parameters they are willing to use along with nonces and Diffie–Hellman (DH) values. The result of this exchange is to set up a special SA called the IKE SA (see Figure 9.2). This SA defines parameters for a secure channel between the peers over which subsequent message exchanges take place. Thus, all subsequent IKE message exchanges are protected by encryption and message authentication. In the second exchange, the two parties authenticate one another and set up a first IPsec SA to be placed in the SADB and used for

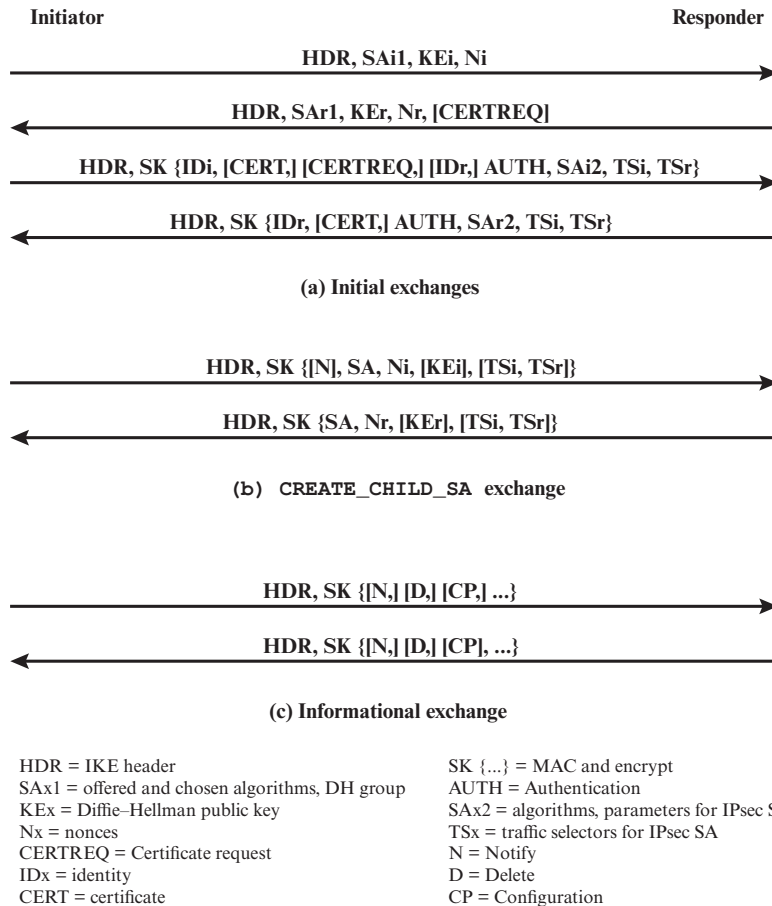


Figure 9.11 IKEv2 Exchanges

protecting ordinary (i.e. non-IKE) communications between the peers. Thus, four messages are needed to establish the first SA for general use.

The **CREATE_CHILD_SA exchange** can be used to establish further SAs for protecting traffic. The **informational exchange** is used to exchange management information, IKEv2 error messages, and other notifications.

Header and Payload Formats

IKE defines procedures and packet formats to establish, negotiate, modify, and delete security associations. As part of SA establishment, IKE defines payloads for exchanging key generation and authentication data. These payload formats provide a consistent framework independent of the specific key exchange protocol, encryption algorithm, and authentication mechanism.

IKE HEADER FORMAT An IKE message consists of an IKE header followed by one or more payloads. All of this is carried in a transport protocol. The specification dictates that implementations must support the use of UDP for the transport protocol.