```
Internet Engineering Task Force (IETF)                        C. Kaufman
Request for Comments: 7296                                     Microsoft
STD: 79                                                       P. Hoffman
Obsoletes: 5996                                           VPN Consortium
Category: Standards Track                                         Y. Nir
ISSN: 2070-1721                                              Check Point
                                                               P. Eronen
                                                             Independent
                                                              T. Kivinen
                                                           INSIDE Secure
                                                            October 2014
```

Internet Key Exchange Protocol Version 2 (IKEv2)

Abstract

This document describes version 2 of the Internet Key Exchange (IKE) protocol. IKE is a component of IPsec used for performing mutual authentication and establishing and maintaining Security Associations (SAs). This document obsoletes RFC 5996, and includes all of the errata for it. It advances IKEv2 to be an Internet Standard.

Status of This Memo

Table of Contents

1. Introduction

IP Security (IPsec) provides confidentiality, data integrity, access control, and data source authentication to IP datagrams. These services are provided by maintaining shared state between the source and the sink of an IP datagram. This state defines, among other things, the specific services provided to the datagram, which cryptographic algorithms will be used to provide the services, and the keys used as input to the cryptographic algorithms.

IKE performs mutual authentication between two parties and establishes an IKE Security Association (SA) that includes shared secret information that can be used to efficiently establish SAs for Encapsulating Security Payload (ESP) [ESP] and a set of cryptographic algorithms to be used by the SAs to protect the traffic that they carry. In this document, the term "suite" or "cryptographic suite" refers to a complete set of algorithms used to protect an SA. An initiator proposes one or more suites by listing supported algorithms that can be combined into suites in a mix-and-match fashion. The SAs for ESP that get set up through that IKE SA we call "Child SAs".

All IKE communications consist of pairs of messages: a request and a response. The pair is called an "exchange", and is sometimes called a "request/response pair". The first two exchanges of messages establishing an IKE SA are called the IKE_SA_INIT exchange and the IKE_AUTH exchange; subsequent IKE exchanges are called either CREATE_CHILD_SA exchanges or INFORMATIONAL exchanges. In the common case, there is a single IKE_SA_INIT exchange and a single IKE_AUTH exchange (a total of four messages) to establish the IKE SA and the first Child SA. In exceptional cases, there may be more than one of each of these exchanges. In all cases, all IKE_SA_INIT exchanges MUST complete before any other exchange type, then all IKE_AUTH exchanges MUST complete, and following that, any number of CREATE_CHILD_SA and INFORMATIONAL exchanges may occur in any order.

An IKE message flow always consists of a request followed by a response. It is the responsibility of the requester to ensure reliability. If the response is not received within a timeout interval, the requester needs to retransmit the request (or abandon the connection).

The first exchange of an IKE session, IKE_SA_INIT, negotiates security parameters for the IKE SA, sends nonces, and sends Elliptic Curve Diffie-Hellman values. Additionally, it MUST contain a CHILDLESS_IKEV2_SUPPORTED notification according to [RFC6023].

The second exchange, IKE_AUTH, transmits identities, proves knowledge of the secrets corresponding to the two identities.

The types of subsequent exchanges are CREATE_CHILD_SA (which creates a Child SA) and INFORMATIONAL (which deletes an SA, reports error conditions, or does other housekeeping). Every request requires a response. An INFORMATIONAL request with no payloads (other than the empty Encrypted payload required by the syntax) is commonly used as a check for liveness. These subsequent exchanges cannot be used until the initial exchanges have completed.

In the description that follows, we assume that no errors occur. Modifications to the flow when errors occur are described in Section 2.21.

1.1. Usage Scenarios

IKE is used to negotiate ESP SAs in a number of different scenarios, each with its own special requirements.

1.1.1. Security Gateway to Security Gateway in Tunnel Mode

```
             +-+-+-+-+-+           +-+-+-+-+-+
             |         | IPsec     |         |
Protected    |Tunnel   | tunnel    |Tunnel   |     Protected
Subnet   <-->|Endpoint |<--------->|Endpoint |<--> Subnet
             |         |           |         |
             +-+-+-+-+-+           +-+-+-+-+-+
```

Figure 1: Security Gateway to Security Gateway Tunnel

In this scenario, neither endpoint of the IP connection implements IPsec, but network nodes between them protect traffic for part of the way. Protection is transparent to the endpoints, and depends on ordinary routing to send packets through the tunnel endpoints for processing. Each endpoint would announce the set of addresses "behind" it, and packets would be sent in tunnel mode where the inner IP header would contain the IP addresses of the actual endpoints.

1.1.2. Endpoint-to-Endpoint Transport Mode

Transport mode is not supported.

1.1.3. Endpoint to Security Gateway in Tunnel Mode

```
   +-+-+-+-+-+                          +-+-+-+-+-+
   |         |         IPsec            |         |     Protected
   |Protected|         tunnel           |Tunnel   |     Subnet
   |Endpoint |<------------------------>|Endpoint |<--- and/or
   |         |                          |         |     Internet
   +-+-+-+-+-+                          +-+-+-+-+-+
```

Figure 3: Endpoint to Security Gateway Tunnel

In this scenario, a protected endpoint (typically a portable roaming computer) connects back to its corporate network through an IPsec-protected tunnel. It might use this tunnel only to access information on the corporate network, or it might tunnel all of its traffic back through the corporate network in order to take advantage

of protection provided by a corporate firewall against Internet-based attacks. In either case, the protected endpoint will want an IP address associated with the security gateway so that packets returned to it will go to the security gateway and be tunneled back. This IP address may be static or may be dynamically allocated by the security gateway. In support of the latter case, IKEv2 includes a mechanism (namely, configuration payloads) for the initiator to request an IP address owned by the security gateway for use for the duration of its SA.

In this scenario, packets will use tunnel mode. On each packet from the protected endpoint, the outer IP header will contain the source IP address associated with its current location (i.e., the address that will get traffic routed to the endpoint directly), while the inner IP header will contain the source IP address assigned by the security gateway (i.e., the address that will get traffic routed to the security gateway for forwarding to the endpoint). The outer destination address will always be that of the security gateway, while the inner destination address will be the ultimate destination for the packet.

In this scenario, it is possible that the protected endpoint will be behind a NAT. In that case, the IP address as seen by the security gateway will not be the same as the IP address sent by the protected endpoint, and packets will have to be UDP encapsulated in order to be routed properly. Interaction with NATs is covered in detail in Section 2.23.

1.1.4. Other Scenarios

Other scenarios are possible, as are nested combinations of the above. One notable example combines aspects of Sections 1.1.1 and 1.1.3. A subnet may make all external accesses through a remote security gateway using an IPsec tunnel, where the addresses on the subnet are routed to the security gateway by the rest of the Internet. An example would be someone's home network being virtually on the Internet with static IP addresses even though connectivity is provided by an ISP that assigns a single dynamically assigned IP address to the user's security gateway (where the static IP addresses and an IPsec relay are provided by a third party located elsewhere).

1.2. The Initial Exchanges

Communication using IKE always begins with IKE_SA_INIT and IKE_AUTH exchanges. These initial exchanges normally consist of six messages. All communications using IKE consist of request/ response pairs. We'll describe the base exchange first, followed by

variations. The first pair of messages (IKE_SA_INIT) negotiate cryptographic algorithms, exchange nonces, and do a Elliptic Curve Diffie-Hellman exchange [ECDH]. The second pair of messages (IKE_AUTH) authenticate the previous messages, exchange identities and certificates. See Section 2.14 for information on how the encryption keys are generated.

All messages following the initial exchange are cryptographically protected using the cryptographic algorithms and keys negotiated in the IKE_SA_INIT exchange. These subsequent messages use the syntax of the Encrypted payload described in Section 3.14, encrypted with keys that are derived as described in Section 2.14. All subsequent messages include an Encrypted payload, even if they are referred to in the text as "empty". For the CREATE_CHILD_SA, IKE_AUTH, or INFORMATIONAL exchanges, the message following the header is encrypted and the message including the header is integrity protected using the cryptographic algorithms negotiated for the IKE SA.

Every IKE message contains a Message ID as part of its fixed header. This Message ID is used to match up requests and responses, and to identify retransmissions of messages.

In the following descriptions, the payloads contained in the message are indicated by names as listed below.

Notation	Payload
AUTH	Authentication
CERT	Certificate

CERTREQ	Certificate Request
CP	Configuration
D	Delete
HDR	IKE header (not a payload)
IDi	Identification - Initiator
IDr	Identification - Responder
KE	Key Exchange
Ni, Nr	Nonce
N	Notify
SA	Security Association
SK	Encrypted and Authenticated

TSi	Traffic Selector - Initiator
TSr	Traffic Selector - Responder
V	Vendor ID

The details of the contents of each payload are described in Section 3. Payloads that may optionally appear will be shown in brackets, such as [CERTREQ]; this indicates that a Certificate Request payload can optionally be included.

Note that this is a simplified initial exchange. The section 2.6 presents the full initial exchange that MUST be implemented.

The initial exchanges are as follows:

```
Initiator                         Responder
-------------------------------------------------------------------
HDR, SAi1, KEi, Ni  -->
```

HDR contains the Security Parameter Indexes (SPIs), version numbers, Exchange Type, Message ID, and flags of various sorts. The SAi1 payload states the cryptographic algorithms the initiator supports for the IKE SA. The KE payload sends the initiator's Elliptic Curve Diffie-Hellman value. Ni is the initiator's nonce.

```
                             <--  HDR, SAr1, KEr, Nr, [CERTREQ],
                                  N(CHILDLESS_IKEV2_SUPPORTED)
```

The responder chooses a cryptographic suite from the initiator's offered choices and expresses that choice in the SAr1 payload, completes the Elliptic Curve Diffie-Hellman exchange with the KEr payload, and sends its nonce in the Nr payload.

At this point in the negotiation, each party can generate a quantity called SKEYSEED (see Section 2.14), from which all keys are derived for that IKE SA. The messages that follow are encrypted and integrity protected in their entirety, with the exception of the message headers. The keys used for the encryption and integrity protection are derived from SKEYSEED and are known as SK_e (encryption) and SK_a (authentication, a.k.a. integrity protection); see Sections 2.13 and 2.14 for details on the key derivation. A separate SK_e and SK_a is computed for each direction. In addition to the keys SK_e and SK_a derived from the Elliptic Curve Diffie-Hellman value for protection of the IKE SA, another quantity SK_d is derived and used for derivation of further keying material for Child SAs. The notation SK { ... } indicates that these payloads are encrypted and integrity protected using that direction's SK_e and SK_a.

```
   HDR, SK {IDi, [CERT,] [CERTREQ,]
       IDr, [V+,] [N+,] AUTH}  -->
```

The initiator asserts its identity with the IDi payload, proves knowledge of the secret corresponding to IDi and integrity protects the contents of the first message using the AUTH payload (see Section 2.15). It might also send its certificate(s) in CERT payload(s) and a list of its trust anchors in CERTREQ payload(s). If any CERT payloads are included, the first certificate provided MUST contain the public key used to verify the AUTH field.

The payload IDr enables the initiator to specify to which of the responder's identities it wants to talk. This is useful when the machine on which the responder is running is hosting multiple identities at the same IP address. If the IDr proposed by the initiator is not acceptable to the responder, the responder might use some other IDr to finish the exchange. If the initiator then does not accept the fact that responder used an IDr different than the one that was requested, the initiator can close the SA after noticing the fact. The payload IDr MUST be included by the initiator.

```
                             <--  HDR, SK {IDr, [CERT,] [V+,] [N+,]
                                       AUTH}
```

The responder asserts its identity with the IDr payload, optionally sends one or more certificates (again with the certificate containing the public key used to verify AUTH listed first), authenticates its identity and protects the integrity of the second message with the AUTH payload.

Both parties in the IKE_AUTH exchange MUST verify that all signatures and Message Authentication Codes (MACs) are computed correctly. If either side uses a shared secret for authentication, the names in the

ID payload MUST correspond to the key used to generate the AUTH payload.

A Notify message types in the IKE_AUTH exchange MUST prevent an IKE SA from being set up. If the failure is related to creating the IKE SA (for example, an AUTHENTICATION_FAILED Notify error message is returned), the IKE SA is not created. Note that although the IKE_AUTH messages are encrypted and integrity protected (only after authentication), if the peer receiving this Notify error message has not yet authenticated the other end (or if the peer fails to authenticate the other end for some reason), the information needs to be treated with caution. More precisely, assuming that the MAC verifies correctly, the sender of the error Notify message is known to be the responder of the IKE_SA_INIT exchange, but the sender's identity cannot be assured.

1.3. The CREATE_CHILD_SA Exchange

The CREATE_CHILD_SA exchange is used to create new Child SAs and to create or rekey Child SAs and to rekey IKE SAs. This exchange consists of a single request/response pair. It MAY be initiated by either end of the IKE SA after the initial exchanges are completed.

An SA is rekeyed by creating a new SA and then deleting the old one. This section describes the first part of rekeying, the creation of new SAs; Section 2.8 covers the mechanics of rekeying, including moving traffic from old to new SAs and the deletion of the old SAs. The two sections must be read together to understand the entire process of rekeying.

Either endpoint may initiate a CREATE_CHILD_SA exchange, so in this section the term initiator refers to the endpoint initiating this exchange. An implementation MAY refuse all CREATE_CHILD_SA requests within an IKE SA.

The CREATE_CHILD_SA request MUST contain a KE payload for an additional Elliptic Curve Diffie-Hellman exchange to enable stronger guarantees of forward secrecy for the Child SA. The keying material for the Child SA is a function of SK_d established during the establishment of the IKE SA, the nonces exchanged during the CREATE_CHILD_SA exchange, and the Elliptic Curve Diffie-Hellman value.

The responder sends a NO_ADDITIONAL_SAS notification to indicate that a CREATE_CHILD_SA request is unacceptable because the responder is unwilling to accept any more Child SAs on this IKE SA. This notification MUST NOT be used to reject IKE SA rekey.

1.3.1. Creating New Child SAs with the CREATE_CHILD_SA Exchange

A Child SA may be created by sending a CREATE_CHILD_SA request. The CREATE_CHILD_SA request for creating a new Child SA is:

```
Initiator                         Responder
-------------------------------------------------------------------
HDR, SK {SA, Ni, KEi, TSi, TSr}  -->
```

The initiator sends SA offer(s) in the SA payload, a nonce in the Ni payload, a Elliptic Curve Diffie-Hellman value in the KEi payload, and the proposed Traffic Selectors for the proposed Child SA in the TSi and TSr payloads.

The CREATE_CHILD_SA response for creating a new Child SA is:

```
                             <--  HDR, SK {SA, Nr, KEr, TSi, TSr}
```

The responder replies (using the same Message ID to respond) with the accepted offer in an SA payload, a nonce in the Nr payload, and a Elliptic Curve Diffie-Hellman value in the KEr payload.

The Traffic Selectors for traffic to be sent on that SA are specified in the TS payloads in the response, which may be a subset of what the initiator of the Child SA proposed.

The ESP_TFC_PADDING_NOT_SUPPORTED notification asserts that the sending endpoint will not accept packets that contain Traffic Flow Confidentiality (TFC) padding over the Child SA being negotiated. If neither endpoint accepts TFC padding, this notification is included in both the request and the response. If this notification is included in only one of the messages, TFC padding can still be sent in the other direction.

The NON_FIRST_FRAGMENTS_ALSO notification is used for fragmentation control. See [IPSECARCH] for a fuller explanation. Both parties need to agree to sending non-first fragments before either party does so. It is enabled only if NON_FIRST_FRAGMENTS_ALSO notification is included in both the request proposing an SA and the response accepting it. If the responder does not want to send or receive non-first fragments, it only omits NON_FIRST_FRAGMENTS_ALSO notification from its response, but does not reject the whole Child SA creation.

A failed attempt to create a Child SA MUST NOT tear down the IKE SA: there is no reason to lose the work done to set up the IKE SA. See Section 2.21 for a list of error messages that might occur if creating a Child SA fails.

1.3.2. Rekeying IKE SAs with the CREATE_CHILD_SA Exchange

The CREATE_CHILD_SA request for rekeying an IKE SA is:

```
Initiator                         Responder
-------------------------------------------------------------------
HDR, SK {SA, Ni, KEi} -->
```

The initiator sends SA offer(s) in the SA payload, a nonce in the Ni payload, and a Elliptic Curve Diffie-Hellman value in the KEi payload. The KEi payload MUST be included. A new initiator SPI is supplied in the SPI field of the SA payload. Once a peer receives a request to rekey an IKE SA or sends a request to rekey an IKE SA, it SHOULD NOT start any new CREATE_CHILD_SA exchanges on the IKE SA that is being rekeyed.

The CREATE_CHILD_SA response for rekeying an IKE SA is:

```
                             <--  HDR, SK {SA, Nr, KEr}
```

The responder replies (using the same Message ID to respond) with the accepted offer in an SA payload, a nonce in the Nr payload, and a Diffie-Hellman value in the KEr payload if the
selected cryptographic suite includes that group. A new responder SPI is supplied in the SPI field of the SA payload.

The new IKE SA has its message counters set to 0, regardless of what they were in the earlier IKE SA. The first IKE requests from both sides on the new IKE SA will have Message ID 0. The old IKE SA retains its numbering, so any further requests (for example, to delete the IKE SA) will have consecutive numbering. The new IKE SA also has its window size reset to 1, and the initiator in this rekey exchange is the new "original initiator" of the new IKE SA.

Section 2.18 also covers IKE SA rekeying in detail.

1.3.3. Rekeying Child SAs with the CREATE_CHILD_SA Exchange

The CREATE_CHILD_SA request for rekeying a Child SA is:

```
Initiator                         Responder
-------------------------------------------------------------------
HDR, SK {N(REKEY_SA), SA, Ni, KEi,
    TSi, TSr}   -->
```

The initiator sends SA offer(s) in the SA payload, a nonce in the Ni payload, a Elliptic Curve Diffie-Hellman value in the KEi payload, and the proposed Traffic Selectors for the proposed Child SA in the TSi and TSr payloads. The TS payloads MUST match exactly with the ones currently installed as a result of initial SA setup; this is further described in 2.9.2.

The notifications described in Section 1.3.1 may also be sent in a rekeying exchange. Usually, these will be the same notifications that were used in the original exchange.

The REKEY_SA notification MUST be included in a CREATE_CHILD_SA exchange if the purpose of the exchange is to replace an existing ESP SA. The SA being rekeyed is identified by the SPI field in the Notify payload; this is the SPI the exchange initiator would expect in inbound ESP packets. There is no data associated with this Notify message type. The Protocol ID field of the REKEY_SA notification is set to match the protocol of the SA we are rekeying (3 for ESP).

The CREATE_CHILD_SA response for rekeying a Child SA is:

```
                             <--  HDR, SK {SA, Nr, KEr,
                                      TSi, TSr}
```

The responder replies (using the same Message ID to respond) with the accepted offer in an SA payload, a nonce in the Nr payload, and a Elliptic Curve Diffie-Hellman value in the KEr payload.

All rekey exchanges MUST include KEi and KEr payloads.

The Traffic Selectors for traffic to be sent on that SA are specified in the TS payloads in the response, which may be a subset of what the initiator of the Child SA proposed.

1.4. The INFORMATIONAL Exchange

At various points during the operation of an IKE SA, peers may desire to convey control messages to each other regarding errors or notifications of certain events. To accomplish this, IKE defines an INFORMATIONAL exchange. INFORMATIONAL exchanges MUST ONLY occur after the initial exchanges and are cryptographically protected with the negotiated keys. Note that some informational messages, not exchanges, can be sent outside the context of an IKE SA. Section 2.21 also covers error messages in great detail.

Control messages that pertain to an IKE SA MUST be sent under that IKE SA. Control messages that pertain to Child SAs MUST be sent under the protection of the IKE SA that generated them (or its successor if the IKE SA was rekeyed).

Messages in an INFORMATIONAL exchange contain zero or more Notification, Delete, and Configuration payloads. The recipient of an INFORMATIONAL exchange request MUST send some response; otherwise, the sender will assume the message was lost in the network and will

retransmit it. That response MAY be an empty message. The request message in an INFORMATIONAL exchange MAY also contain no payloads. This is the expected way an endpoint can ask the other endpoint to verify that it is alive.

The INFORMATIONAL exchange is defined as:

```
Initiator                         Responder
-------------------------------------------------------------------
HDR, SK {[N,] [D,]
    [CP,] ...}  -->
                             <--  HDR, SK {[N,] [D,]
                                      [CP,] ...}
```

The processing of an INFORMATIONAL exchange is determined by its component payloads.

1.4.1. Deleting an SA with INFORMATIONAL Exchanges

ESP SAs always exist in pairs, with one SA in each direction. When an SA is closed, both members of the pair MUST be closed (that is, deleted). Each endpoint MUST close its incoming SAs and allow the other endpoint to close the other SA in each pair. To delete an SA, an INFORMATIONAL exchange with one or more Delete payloads is sent listing the SPIs (as they would be expected in the headers of inbound packets) of the SAs to be deleted. The recipient MUST close the designated SAs. Note that one never sends Delete payloads for the two sides of an SA in a single message. If there are many SAs to delete at the same time, one includes Delete payloads for the inbound half of each SA pair in the INFORMATIONAL exchange.

Normally, the response in the INFORMATIONAL exchange will contain Delete payloads for the paired SAs going in the other direction. There is one exception. If, by chance, both ends of a set of SAs independently decide to close them, each may send a Delete payload and the two requests may cross in the network. If a node receives a delete request for SAs for which it has already issued a delete request, it MUST delete the outgoing SAs while processing the request and the incoming SAs while processing the response. In that case, the responses MUST NOT include Delete payloads for the deleted SAs, since that would result in duplicate deletion and could in theory delete the wrong SA.

Similar to ESP SAs, IKE SAs are also deleted by sending an INFORMATIONAL exchange. Deleting an IKE SA implicitly closes any remaining Child SAs negotiated under it. The response to a request that deletes the IKE SA is an empty INFORMATIONAL response.

Half-closed ESP connections are anomalous, and a node with auditing capability should probably audit their existence if they persist. Note that this specification does not specify time periods, so it is up to individual endpoints to decide how long to wait. A node MAY refuse to accept incoming data on half-closed connections but MUST NOT unilaterally close them and reuse the SPIs. If connection state becomes sufficiently messed up, a node MAY close the IKE SA, as described above. It can then rebuild the SAs it needs on a clean base under a new IKE SA.

1.5. Informational Messages outside of an IKE SA

There are some cases in which a node receives a packet that it cannot process. Usually, in that case, it should silently drop received packets:

- If an ESP packet arrives with an unrecognized SPI.

- If an encrypted IKE request packet arrives on port 4500 with an unrecognized IKE SPI.

- If an IKE request packet arrives with a higher major version number than the implementation supports.

In the first case, if the receiving node has an active IKE SA to the IP address from whence the packet came, it MUST NOT reply.

1.6. Requirements Terminology

Definitions of the primitive terms in this document (such as Security Association or SA) can be found in [IPSECARCH]. It should be noted that parts of IKEv2 rely on some of the processing rules in [IPSECARCH], as described in various sections of this document.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [MUSTSHOULD].

1.7. Significant Differences between RFC 4306 and RFC 5996

1.8. Differences between RFC 5996 and This Document

2. IKE Protocol Details and Variations

IKE normally listens and sends on UDP port 500, though IKE messages may also be received on UDP port 4500 with a slightly different format (see Section 2.23). Since UDP is a datagram (unreliable) protocol, IKE includes in its definition recovery from transmission errors, including packet loss, packet replay, and packet forgery. IKE is designed to function so long as (1) at least one of a series of retransmitted packets reaches its destination before timing out; and (2) the channel is not so full of forged and replayed packets so

as to exhaust the network or CPU capacities of either endpoint. Even in the absence of those minimum performance requirements, IKE is designed to fail cleanly (as though the network were broken).

Although IKEv2 messages are intended to be short, they contain structures with no hard upper bound on size (in particular, digital certificates), and IKEv2 itself does not have a mechanism for fragmenting large messages. IP defines a mechanism for fragmentation of oversized UDP messages, but implementations vary in the maximum message size supported. Furthermore, use of IP fragmentation opens an implementation to denial-of-service (DoS) attacks [DOSUDPPROT]. Finally, some NAT and/or firewall implementations may block IP fragments.

All IKEv2 implementations MUST be able to send, receive, and process IKE messages that are up to 1280 octets long, and they SHOULD be able to send, receive, and process messages that are up to 3000 octets long. IKEv2 implementations need to be aware of the maximum UDP message size supported and MAY shorten messages by leaving out some certificates or cryptographic suite proposals if that will keep messages below the maximum.

The UDP payload of all packets containing IKE messages sent on port 4500 MUST begin with the prefix of four zeros; otherwise, the receiver won't know how to handle them.

2.1. Use of Retransmission Timers

All messages in IKE exist in pairs: a request and a response. The setup of an IKE SA normally consists of two exchanges. Once the IKE SA is set up, either end of the Security Association may initiate requests at any time, and there can be many requests and responses "in flight" at any given moment. But each message is labeled as either a request or a response, and for each exchange, one end of the Security Association is the initiator and the other is the responder.

For every pair of IKE messages, the initiator is responsible for retransmission in the event of a timeout. The responder MUST never retransmit a response unless it receives a retransmission of the request. In that event, the responder MUST ignore the retransmitted request except insofar as it causes a retransmission of the response. The initiator MUST remember each request until it receives the corresponding response. The responder MUST remember each response

until it receives a request whose sequence number is larger than or equal to the sequence number in the response plus its window size (see Section 2.3). In order to allow saving memory, responders are allowed to forget the response after a timeout of several minutes. If the responder receives a retransmitted request for which it has already forgotten the response, it MUST ignore the request (and not, for example, attempt constructing a new response).

IKE is a reliable protocol: the initiator MUST retransmit a request until it either receives a corresponding response or deems the IKE SA to have failed. In the latter case, the initiator discards all state associated with the IKE SA and any Child SAs that were negotiated using that IKE SA. A retransmission from the initiator MUST be bitwise identical to the original request. That is, everything starting from the IKE header (the IKE SA initiator's SPI onwards) must be bitwise identical; items before it (such as the IP and UDP headers) do not have to be identical.

Retransmissions of the IKE_SA_INIT request require some special handling. When a responder receives an IKE_SA_INIT request, it has to determine whether the packet is a retransmission belonging to an existing "half-open" IKE SA (in which case the responder retransmits the same response), or a new request (in which case the responder creates a new IKE SA and sends a fresh response), or it belongs to an existing IKE SA where the IKE_AUTH request has been already received (in which case the responder ignores it).

It is not sufficient to use the initiator's SPI and/or IP address to differentiate between these three cases because two different peers behind a single NAT could choose the same initiator SPI. Instead, a robust responder will do the IKE SA lookup using the whole packet, its hash, or the Ni payload.

The retransmission policy for one-way messages is somewhat different from that for regular messages. Because no acknowledgement is ever sent, there is no reason to gratuitously retransmit one-way messages. Given that all these messages are errors, it makes sense to send them only once per "offending" packet, and only retransmit if further offending packets are received. Still, it also makes sense to limit retransmissions of such error messages.

2.2. Use of Sequence Numbers for Message ID

Every IKE message contains a Message ID as part of its fixed header. This Message ID is used to match up requests and responses and to identify retransmissions of messages. Retransmission of a message MUST use the same Message ID as the original message.

The Message ID is a 32-bit quantity, which is zero for the IKE_SA_INIT messages (including retries of the message due to responses such as COOKIE and INVALID_KE_PAYLOAD), and incremented for each subsequent exchange. The Message ID is reset to zero in the new IKE SA after the IKE SA is rekeyed.

Each endpoint in the IKE Security Association maintains two "current" Message IDs: the next one to be used for a request it initiates and the next one it expects to see in a request from the other end. These counters increment as requests are generated and received. Responses always contain the same Message ID as the corresponding request. That means that after the initial exchange, each integer n may appear as the Message ID in four distinct messages: the nth request from the original IKE initiator, the corresponding response, the nth request from the original IKE responder, and the corresponding response. If the two ends make a very different number of requests, the Message IDs in the two directions can be very different. There is no ambiguity in the messages, however, because the Initiator and Response flags in the message header specify which of the four messages a particular one is.

Throughout this document, "initiator" refers to the party who initiated the exchange being described. The "original initiator" always refers to the party who initiated the exchange that resulted in the current IKE SA. In other words, if the "original responder" starts rekeying the IKE SA, that party becomes the "original initiator" of the new IKE SA.

Note that Message IDs are cryptographically protected and provide protection against message replays. In the unlikely event that Message IDs grow too large to fit in 32 bits, the IKE SA MUST be closed or rekeyed.

2.3. Window Size for Overlapping Requests

An implementation compliant with the profile MUST NOT emit SET_WINDOW_SIZE notifications and MUST NOT access and process multiple requests while it has a request outstanding. Such an implementation MUST also silently ignore all received SET_WINDOW_SIZE notification. Such requirements are expected to keep implementations compliant with the framework simpler while still compatible with pure

IKEv2 implementations.

The SET_WINDOW_SIZE notification asserts that the sending endpoint is capable of keeping state for multiple outstanding exchanges, permitting the recipient to send multiple requests before getting a response to the first. The data associated with a SET_WINDOW_SIZE notification MUST be 4 octets long and contain the big endian representation of the number of messages the sender promises to keep. The window size is always one until the initial exchanges complete.

An IKE endpoint MUST wait for a response to each of its messages before sending a subsequent message unless it has received a SET_WINDOW_SIZE Notify message from its peer informing it that the peer is prepared to maintain state for multiple outstanding messages in order to allow greater throughput.

After an IKE SA is set up, in order to maximize IKE throughput, an IKE endpoint MAY issue multiple requests before getting a response to any of them, up to the limit set by its peer's SET_WINDOW_SIZE. These requests may pass one another over the network. An IKE endpoint MUST be prepared to accept and process a request while it has a request outstanding in order to avoid a deadlock in this situation. An IKE endpoint may also accept and process multiple requests while it has a request outstanding.

An IKE endpoint MUST NOT exceed the peer's stated window size for transmitted IKE requests. In other words, if the responder stated its window size is N, then when the initiator needs to make a request X, it MUST wait until it has received responses to all requests up through request X-N. An IKE endpoint MUST keep a copy of (or be able to regenerate exactly) each request it has sent until it receives the corresponding response. An IKE endpoint MUST keep a copy of (or be able to regenerate exactly) the number of previous responses equal to its declared window size in case its response was lost and the initiator requests its retransmission by retransmitting the request.

An IKE endpoint supporting a window size greater than one ought to be capable of processing incoming requests out of order to maximize performance in the event of network failures or packet reordering.

The window size is normally a (possibly configurable) property of a particular implementation, and is not related to congestion control (unlike the window size in TCP, for example). In particular, what the responder should do when it receives a SET_WINDOW_SIZE notification containing a smaller value than is currently in effect is not defined. Thus, there is currently no way to reduce the window size of an existing IKE SA; you can only increase it. When rekeying an IKE SA, the new IKE SA starts with window size 1 until it is explicitly increased by sending a new SET_WINDOW_SIZE notification.

2.4. State Synchronization and Connection Timeouts

An IKE endpoint is allowed to forget all of its state associated with an IKE SA and the collection of corresponding Child SAs at any time. This is the anticipated behavior in the event of an endpoint crash and restart. It is important when an endpoint either fails or reinitializes its state that the other endpoint detect those conditions and not continue to waste network bandwidth by sending packets over discarded SAs and having them fall into a black hole.

The INITIAL_CONTACT notification asserts that this IKE SA is the only IKE SA currently active between the authenticated identities. It MAY be sent when an IKE SA is established after a crash, and the recipient MAY use this information to delete any other IKE SAs it has to the same authenticated identity without waiting for a timeout. The INITIAL_CONTACT notification, if sent, MUST be in the first IKE_AUTH request or response, not as a separate exchange afterwards; receiving parties MUST ignore it in other messages.

Since IKE is designed to operate in spite of DoS attacks from the network, an endpoint MUST NOT conclude that the other endpoint has failed based on any routing information (e.g., ICMP messages) or IKE messages that arrive without cryptographic protection (e.g., Notify messages complaining about unknown SPIs). An endpoint MUST conclude that the other endpoint has failed only when repeated attempts to contact it have gone unanswered for a timeout period or when a cryptographically protected INITIAL_CONTACT notification is received on a different IKE SA to the same authenticated identity. An endpoint should suspect that the other endpoint has failed based on routing information and initiate a request to see whether the other endpoint is alive. To check whether the other side is alive, IKE specifies an empty INFORMATIONAL request that (like all IKE requests) requires an acknowledgement (note that within the context of an IKE SA, an "empty" message consists of an IKE header followed by an Encrypted payload that contains no payloads). If a cryptographically protected (fresh, i.e., not retransmitted) message has been received from the other side recently, unprotected Notify messages MAY be ignored. Implementations MUST limit the rate at which they take actions based on unprotected messages.

The number of retries and length of timeouts are not covered in this specification because they do not affect interoperability. It is suggested that messages be retransmitted at least a dozen times over a period of at least several minutes before giving up on an SA, but

different environments may require different rules. To be a good network citizen, retransmission times MUST increase exponentially to avoid flooding the network and making an existing congestion situation worse. If there has only been outgoing traffic on all of the SAs associated with an IKE SA, it is essential to confirm liveness of the other endpoint to avoid black holes. If no cryptographically protected messages have been received on an IKE SA or any of its Child SAs recently, the system needs to perform a liveness check in order to prevent sending messages to a dead peer. (This is sometimes called "dead peer detection" or "DPD", although it is really detecting live peers, not dead ones.) Receipt of a fresh cryptographically protected message on an IKE SA or any of its Child SAs ensures liveness of the IKE SA and all of its Child SAs. Note that this places requirements on the failure modes of an IKE endpoint. An implementation needs to stop sending over any SA if some failure prevents it from receiving on all of the associated SAs. If a system creates Child SAs that can fail independently from one another without the associated IKE SA being able to send a delete message, then the system MUST negotiate such Child SAs using separate IKE SAs.

Note that with these rules, there is no reason to negotiate and agree upon an SA lifetime. If IKE presumes the partner is dead, based on repeated lack of acknowledgement to an IKE message, then the IKE SA and all Child SAs set up through that IKE SA are deleted.

An IKE endpoint may at any time delete inactive Child SAs to recover resources used to hold their state. If an IKE endpoint chooses to delete Child SAs, it MUST send Delete payloads to the other end notifying it of the deletion. It MAY similarly time out the IKE SA. Closing the IKE SA implicitly closes all associated Child SAs. In this case, an IKE endpoint SHOULD send a Delete payload indicating that it has closed the IKE SA unless the other endpoint is no longer responding.

2.5. Version Numbers and Forward Compatibility

This document describes version 2.0 of IKE, meaning the major version number is 2 and the minor version number is 0. This document is a replacement for [IKEV2].

The major version number should be incremented only if the packet formats or required actions have changed so dramatically that an older version node would not be able to interoperate with a newer version node if it simply ignored the fields it did not understand and took the actions specified in the older specification. The minor version number indicates new capabilities, and MUST be ignored by a node with a smaller minor version number, but used for informational purposes by the node with the larger minor version number. For example, it might indicate the ability to process a newly defined Notify message type. The node with the larger minor version number would simply note that its correspondent would not be able to understand that message and therefore would not send it.

If an endpoint supports major version n, and major version m, it MUST support all versions between n and m. If it receives a message with a major version that it supports, it MUST respond with that version number. In order to prevent two nodes from being tricked into corresponding with a lower major version number than the maximum that they both support, IKE has a flag that indicates that the node is capable of speaking a higher major version number.

Thus, the major version number in the IKE header indicates the version number of the message, not the highest version number that the transmitter supports. If the initiator is capable of speaking versions n, n+1, and n+2, and the responder is capable of speaking versions n and n+1, then they will negotiate speaking n+1, where the initiator will set a flag indicating its ability to speak a higher version. If they mistakenly (perhaps through an active attacker sending error messages) negotiate to version n, then both will notice that the other side can support a higher version number, and they MUST break the connection and reconnect using version n+1.

Also, for forward compatibility, all fields marked RESERVED MUST be set to zero by an implementation running version 2.0, and their content MUST be ignored by an implementation running version 2.0 ("Be conservative in what you send and liberal in what you receive" [IP]). In this way, future versions of the protocol can use those fields in a way that is guaranteed to be ignored by implementations that do not understand them. Similarly, payload types that are not defined are reserved for future use; implementations of a version where they are undefined MUST skip over those payloads and ignore their contents.

IKEv2 adds a "critical" flag to each payload header for further flexibility for forward compatibility. If the critical flag is set and the payload type is unrecognized, the message MUST be rejected and the response to the IKE request containing that payload MUST include a Notify payload UNSUPPORTED_CRITICAL_PAYLOAD, indicating an unsupported critical payload was included. In that Notify payload, the Notification Data contains the one-octet payload type. If the critical flag is not set and the payload type is unsupported, that payload MUST be ignored. Payloads sent in IKE response messages MUST NOT have the critical flag set. Note that the critical flag applies only to the payload type, not the contents. If the payload type is recognized, but the payload contains something that is not (such as an unknown transform inside an SA payload, or an unknown Notify Message Type inside a Notify payload), the critical flag is ignored.

Although new payload types may be added in the future and may appear interleaved with the fields defined in this specification, implementations MUST send the payloads defined in this specification in the order shown in the figures in Sections 1 and 2; implementations MUST reject as invalid a message with those payloads in any other order.

2.6. IKE SA SPIs and Cookies

The initial two eight-octet fields in the header, called the "IKE SPIs", are used as a connection identifier at the beginning of IKE packets. Each endpoint chooses one of the two SPIs and MUST choose them so as to be unique identifiers of an IKE SA. An SPI value of zero is special: it indicates that the remote SPI value is not yet known by the sender.

Incoming IKE packets MUST be mapped to an IKE SA only using the packet's SPI, not using (for example) the source IP address of the packet.

Unlike ESP where only the recipient's SPI appears in the header of a message, in IKE the sender's SPI is also sent in every message. Since the SPI chosen by the original initiator of the IKE SA is always sent first, an endpoint with multiple IKE SAs open that wants to find the appropriate IKE SA using the SPI it assigned must look at the Initiator flag in the header to determine whether it assigned the first or the second eight octets.

In the first message of an initial IKE exchange, the initiator will not know the responder's SPI value and will therefore set that field to zero. When the IKE_SA_INIT exchange does not result in the creation of an IKE SA due to INVALID_KE_PAYLOAD, NO_PROPOSAL_CHOSEN, or COOKIE, the responder's SPI will be zero also in the response message. However, if the responder sends a non-zero responder SPI, the initiator should not reject the response for only that reason.

The data associated with a COOKIE notification MUST be between 16 and 64 octets in length (inclusive), and its generation is described later in this section. If the IKE_SA_INIT response includes the COOKIE notification, the initiator MUST then retry the IKE_SA_INIT request, and include the COOKIE notification containing the received data as the first payload, and all other payloads unchanged. The initial exchange will then be as follows:

```
Initiator                         Responder
```

```
   --------------------------------------------------------------------
   HDR(A,0), SAi1, KEi, Ni, -->
                                <--  HDR(A,0), N(COOKIE)
   HDR(A,0), N(COOKIE), SAi1,
       KEi, Ni  -->
                                <--  HDR(A,B), SAr1, KEr,
                                         Nr, [CERTREQ]
   HDR(A,B), SK {IDi, [CERT,]
       [CERTREQ,] IDr, AUTH}  -->
                                <--  HDR(A,B), SK {IDr, [CERT,], AUTH}
```

The first two messages do not affect any initiator or responder state except for communicating the cookie. In particular, the message sequence numbers in the first four messages will all be zero and the message sequence numbers in the last two messages will be one. 'A' is the SPI assigned by the initiator, while 'B' is the SPI assigned by the responder.

An IKE implementation can implement its responder cookie generation in such a way as to not require any saved state to recognize its valid cookie when the second IKE_SA_INIT message arrives. The exact algorithms and syntax used to generate cookies do not affect interoperability and hence are not specified here. The following is an example of how an endpoint could use cookies to implement limited DoS protection.

A good way to do this is to set the responder cookie to be:

```
   Cookie = <VersionIDofSecret> | Hash(Ni | IPi | SPIi | <secret>)
```

where <secret> is a randomly generated secret known only to the responder and periodically changed and | indicates concatenation. <VersionIDofSecret> should be changed whenever <secret> is regenerated. The cookie can be recomputed when the IKE_SA_INIT arrives the second time and compared to the cookie in the received

message. If it matches, the responder knows that the cookie was generated since the last change to <secret> and that IPi must be the same as the source address it saw the first time. Incorporating SPIi into the calculation ensures that if multiple IKE SAs are being set up in parallel they will all get different cookies (assuming the initiator chooses unique SPIi's). Incorporating Ni in the hash ensures that an attacker who sees only message 2 can't successfully forge a message 3. Also, incorporating SPIi in the hash prevents an attacker from fetching one cookie from the other end, and then initiating many IKE_SA_INIT exchanges all with different initiator SPIs (and perhaps port numbers) so that the responder thinks that there are a lot of machines behind one NAT box that are all trying to connect.

If a new value for <secret> is chosen while there are connections in the process of being initialized, an IKE_SA_INIT might be returned with other than the current <VersionIDofSecret>. The responder in that case MAY reject the message by sending another response with a new cookie or it MAY keep the old value of <secret> around for a short time and accept cookies computed from either one. The responder should not accept cookies indefinitely after <secret> is changed, since that would defeat part of the DoS protection. The responder should change the value of <secret> frequently, especially if under attack.

When one party receives an IKE_SA_INIT request containing a cookie whose contents do not match the value expected, that party MUST ignore the cookie and process the message as if no cookie had been included; usually this means sending a response containing a new cookie. The initiator should limit the number of cookie exchanges it tries before giving up, possibly using exponential back-off. An attacker can forge multiple cookie responses to the initiator's IKE_SA_INIT message, and each of those forged cookie replies will cause two packets to be sent: one packet from the initiator to the responder (which will reject those cookies), and one response from responder to initiator that includes the correct cookie.

2.6.1. Interaction of COOKIE and INVALID_KE_PAYLOAD

When the initiator receives a cookie from the responder, the initiator MUST include the cookie in all subsequent retries.

As both peers support including the cookie in all retries, a slightly shorter exchange happen.

```
   Initiator                   Responder
   -----------------------------------------------------------
   HDR(A,0), SAi1, KEi, Ni -->
                           <-- HDR(A,0), N(COOKIE)
   HDR(A,0), N(COOKIE), SAi1, KEi, Ni  -->
                           <-- HDR(A,0), N(INVALID_KE_PAYLOAD)
   HDR(A,0), N(COOKIE), SAi1, KEi', Ni -->
                           <-- HDR(A,B), SAr1, KEr, Nr
```

Implementations MUST support this shorter exchange.

2.7. Cryptographic Algorithm Negotiation

The payload type known as "SA" indicates a proposal for a set of choices of IPsec protocols (IKE, ESP) for the SA as well as cryptographic algorithms associated with each protocol.

An SA payload consists of one or more proposals. Each proposal includes one protocol. Each protocol contains one or more transforms -- each specifying a cryptographic algorithm. Each transform contains zero or more attributes (attributes are needed only if the Transform ID does not completely specify the cryptographic algorithm).

This hierarchical structure was designed to efficiently encode proposals for cryptographic suites when the number of supported suites is large because multiple values are acceptable for multiple transforms. The responder MUST choose a single suite, which may be any subset of the SA proposal following the rules below.

Each proposal contains one protocol. If a proposal is accepted, the SA response MUST contain the same protocol. The responder MUST accept a single proposal or reject them all and return an error. The error is given in a notification of type NO_PROPOSAL_CHOSEN.

Each IPsec protocol proposal contains one or more transforms. Each transform contains a Transform Type. The accepted cryptographic suite MUST contain exactly one transform of each type included in the proposal. For example: if an ESP proposal includes transforms ENCR_AES w/keysize 128, ENCR_AES w/keysize 256, AUTH_HMAC_SHA2, and AUTH_AES_XCBC, the accepted suite MUST contain one of the ENCR_ transforms and one of the AUTH_ transforms. Thus, six combinations are acceptable.

If an initiator proposes both normal ciphers with integrity protection as well as combined-mode ciphers, then two proposals are needed. One of the proposals includes the normal ciphers with the integrity algorithms for them, and the other proposal includes all the combined-mode ciphers without the integrity algorithms (because combined-mode ciphers are not allowed to have any integrity algorithm other than "NONE").

2.8. Rekeying

IKE and ESP Security Associations use secret keys that should be used only for a limited amount of time and to protect a limited amount of data. This limits the lifetime of the entire Security Association. When the lifetime of a Security Association expires, the Security Association MUST NOT be used. If there is demand, new Security Associations MAY be established. Reestablishment of Security Associations to take the place of ones that expire is referred to as "rekeying".

The ability to rekey SAs without restarting the entire IKE SA MUST be supported. If an SA
has expired or is about to expire and rekeying attempts using the mechanisms described here fail, an implementation MUST close the IKE SA and any associated Child SAs and then MAY start new ones. Implementations MUST support in-place rekeying of SAs, since doing so offers better performance and is likely to reduce the number of packets lost during the transition.

To rekey a Child SA within an existing IKE SA, create a new, equivalent SA (see Section 2.17 below), and when the new one is established, delete the old one. Note that, when rekeying, the new Child SA MUST NOT have different Traffic Selectors and algorithms than the old one.

To rekey an IKE SA, establish a new equivalent IKE SA (see Section 2.18 below) with the peer to whom the old IKE SA is shared using a CREATE_CHILD_SA within the existing IKE SA. An IKE SA so created inherits all of the original IKE SA's Child SAs, and the new IKE SA is used for all control messages needed to maintain those Child SAs. After the new equivalent IKE SA is created, the initiator deletes the old IKE SA, and the Delete payload to delete itself MUST be the last request sent over the old IKE SA.

SAs should be rekeyed proactively, i.e., the new SA should be established before the old one expires and becomes unusable. Enough time should elapse between the time the new SA is established and the old one becomes unusable so that traffic can be switched over to the new SA.

In IKEv2, each end of the SA is responsible for enforcing its own lifetime policy on the SA and rekeying the SA when necessary. If the two ends have different lifetime policies, the end with the shorter lifetime will end up always being the one to request the rekeying. If an SA has been inactive for a long time and if an endpoint would not initiate the SA in the absence of traffic, the endpoint MAY choose to close the SA instead of rekeying it when its lifetime expires. It can also do so if there has been no traffic since the last time the SA was rekeyed.

Note that IKEv2 deliberately allows parallel SAs with the same Traffic Selectors between common endpoints. One of the purposes of this is to support traffic quality of service (QoS) differences among the SAs (see [DIFFSERVFIELD], [DIFFSERVARCH], and Section 4.1 of [DIFFTUNNEL]). Hence unlike IKEv1, the combination of the endpoints and the Traffic Selectors may not uniquely identify an SA between those endpoints, so the IKEv1 rekeying heuristic of deleting SAs on the basis of duplicate Traffic Selectors SHOULD NOT be used.

There are timing windows -- particularly in the presence of lost packets -- where endpoints may not agree on the state of an SA. The responder to a CREATE_CHILD_SA MUST be prepared to accept messages on an SA before sending its response to the creation request, so there is no ambiguity for the initiator. The initiator MAY begin sending on an SA as soon as it processes the response. The initiator,

however, cannot receive on a newly created SA until it receives and processes the response to its CREATE_CHILD_SA request. How, then, is the responder to know when it is OK to send on the newly created SA?

From a technical correctness and interoperability perspective, the responder MAY begin sending on an SA as soon as it sends its response to the CREATE_CHILD_SA request. In some situations, however, this could result in packets unnecessarily being dropped, so an implementation MUST defer such sending.

The responder can be assured that the initiator is prepared to receive messages on an SA if either (1) it has received a cryptographically valid message on the other half of the SA pair, or (2) the new SA rekeys an existing SA and it receives an IKE request to close the replaced SA. When rekeying an SA, the responder continues to send traffic on the old SA until one of those events occurs. When establishing a new SA, the responder MAY defer sending messages on a new SA until either it receives one or a timeout has occurred. If an initiator receives a message on an SA for which it has not received a response to its CREATE_CHILD_SA request, it interprets that as a likely packet loss and retransmits the CREATE_CHILD_SA request. An initiator MAY send a dummy ESP message on a newly created ESP SA if it has no messages queued in order to assure the responder that the initiator is ready to receive messages.

2.8.1. Simultaneous Child SA Rekeying

If the two ends have the same lifetime policies, it is possible that both will initiate a rekeying at the same time (which will result in redundant SAs). To reduce the probability of this happening, the timing of rekeying requests SHOULD be jittered (delayed by a random amount of time after the need for rekeying is noticed).

This form of rekeying may temporarily result in multiple similar SAs between the same pairs of nodes. When there are two SAs eligible to receive packets, a node MUST accept incoming packets through either SA. If redundant SAs are created through such a collision, the SA created with the lowest of the four nonces used in the two exchanges SHOULD be closed by the endpoint that created it. "Lowest" means an octet-by-octet comparison (instead of, for instance, comparing the nonces as large integers). In other words, start by comparing the first octet; if they're equal, move to the next octet, and so on. If you reach the end of one nonce, that nonce is the lower one. The node that initiated the surviving rekeyed SA should delete the replaced SA after the new one is established.

The following is an explanation on the impact this has on implementations. Assume that hosts A and B have an existing Child SA pair with SPIs (SPIa1,SPIb1), and both start rekeying it at the same time:

```
Host A                            Host B
-------------------------------------------------------------------
send req1: N(REKEY_SA,SPIa1),
    SA(..,SPIa2,..),Ni1,..  -->
                             <--  send req2: N(REKEY_SA,SPIb1),
                                      SA(..,SPIb2,..),Ni2
recv req2 <--
```

At this point, A knows there is a simultaneous rekeying happening. However, it cannot yet know which of the exchanges will have the lowest nonce, so it will just note the situation and respond as usual.

```
send resp2: SA(..,SPIa3,..),
     Nr1,..  -->
                             -->  recv req1
```

Now B also knows that simultaneous rekeying is going on. It responds as usual.

```
                             <--  send resp1: SA(..,SPIb3,..),
                                      Nr2,..
   recv resp1 <--
                             -->  recv resp2
```

At this point, there are three Child SA pairs between A and B (the old one and two new ones). A and B can now compare the nonces. Suppose that the lowest nonce was Nr1 in message resp2; in this case, B (the sender of req2) deletes the redundant new SA, and A (the node that initiated the surviving rekeyed SA), deletes the old one.

```
   send req3: D(SPIa1) -->
                              <--  send req4: D(SPIb2)
                              -->  recv req3
                              <--  send resp3: D(SPIb1)
   recv req4 <--
   send resp4: D(SPIa3) -->
```

The rekeying is now finished.

However, there is a second possible sequence of events that can happen if some packets are lost in the network, resulting in retransmissions. The rekeying begins as usual, but A's first packet (req1) is lost.

```
   Host A                             Host B
   -------------------------------------------------------------------
   send req1: N(REKEY_SA,SPIa1),
       SA(..,SPIa2,..),
       Ni1,..  -->  (lost)
                              <--  send req2: N(REKEY_SA,SPIb1),
                                      SA(..,SPIb2,..),Ni2
   recv req2 <--
   send resp2: SA(..,SPIa3,..),
       Nr1,.. -->
                              -->  recv resp2
                              <--  send req3: D(SPIb1)
   recv req3 <--
   send resp3: D(SPIa1) -->
                              -->  recv resp3
```

From B's point of view, the rekeying is now completed, and since it has not yet received A's req1, it does not even know that there was simultaneous rekeying. However, A will continue retransmitting the message, and eventually it will reach B.

```
   resend req1 -->
                                 -->  recv req1
```

To B, it looks like A is trying to rekey an SA that no longer exists; thus, B responds to the request with something non-fatal such as CHILD_SA_NOT_FOUND.

```
                                 <--  send resp1: N(CHILD_SA_NOT_FOUND)
   recv resp1 <--
```

When A receives this error, it already knows there was simultaneous rekeying, so it can ignore the error message.

2.8.2. Simultaneous IKE SA Rekeying

Probably the most complex case occurs when both peers try to rekey the IKE_SA at the same time. Basically, the text in Section 2.8 applies to this case as well; however, it is important to ensure that the Child SAs are inherited by the correct IKE_SA.

The case where both endpoints notice the simultaneous rekeying works the same way as with Child SAs. After the CREATE_CHILD_SA exchanges, three IKE SAs exist between A and B: the old IKE SA and two new IKE SAs. The new IKE SA containing the lowest nonce SHOULD be deleted by the node that created it, and the other surviving new IKE SA MUST inherit all the Child SAs.

In addition to normal simultaneous rekeying cases, there is a special case where one peer finishes its rekey before it even notices that other peer is doing a rekey. If only one peer detects a simultaneous rekey, redundant SAs are not created. In this case, when the peer that did not notice the simultaneous rekey gets the request to rekey the IKE SA that it has already successfully rekeyed, it SHOULD return TEMPORARY_FAILURE because it is an IKE SA that it is currently trying to close (whether or not it has already sent the delete notification for the SA). If the peer that did notice the simultaneous rekey gets the delete request from the other peer for the old IKE SA, it knows that the other peer did not detect the simultaneous rekey, and the first peer can forget its own rekey attempt.

```
   Host A                      Host B
   --------------------------------------------------------------------
   send req1:
        SA(..,SPIa1,..),Ni1,.. -->
                             <-- send req2: SA(..,SPIb1,..),Ni2,..
                             --> recv req1
                             <-- send resp1: SA(..,SPIb2,..),Nr2,..
   recv resp1 <--
   send req3: D() -->
                             --> recv req3
```

At this point, host B sees a request to close the IKE_SA. There's not much more to do than to reply as usual. However, at this point host B should stop retransmitting req2, since once host A receives resp3, it will delete all the state associated with the old IKE_SA and will not be able to reply to it.

```
                             <-- send resp3: ()
```

2.8.3. Rekeying the IKE SA versus Reauthentication

Rekeying the IKE SA and reauthentication are different concepts in IKEv2. Rekeying the IKE SA establishes new keys for the IKE SA and resets the Message ID counters, but it does not authenticate the parties again (no AUTH payload are involved).

Although rekeying the IKE SA may be important in some environments, reauthentication (the verification that the parties still have access to the long-term credentials) is often more important.

IKEv2 does not have any special support for reauthentication. Reauthentication is done by creating a new IKE SA from scratch (using IKE_SA_INIT/IKE_AUTH exchanges, without any REKEY_SA Notify payloads), creating new Child SAs within the new IKE SA (without REKEY_SA Notify payloads), and finally deleting the old IKE SA (which deletes the old Child SAs as well).

This means that reauthentication also establishes new keys for the IKE SA and Child SAs. Therefore, while rekeying can be performed more often than reauthentication, the situation where "authentication lifetime" is shorter than "key lifetime" does not make sense.

While creation of a new IKE SA can be initiated by either party (initiator or responder in the original IKE SA), the use of Configuration payloads means in practice that reauthentication has to be initiated by the same party as the original IKE SA. IKEv2 does not currently allow the responder to request reauthentication in this case; however, there are extensions that add this functionality such as [REAUTH].

2.9. Traffic Selector Negotiation

When an RFC4301-compliant IPsec subsystem receives an IP packet that matches a "protect" selector in its Security Policy Database (SPD), the subsystem protects that packet with IPsec. When no SA exists yet, it is the task of IKE to create it. Maintenance of a system's SPD is outside the scope of IKE, although some implementations might update their SPD in connection with the running of IKE (for an example scenario, see Section 1.1.3).

Traffic Selector (TS) payloads allow endpoints to communicate some of the information from their SPD to their peers. These must be communicated to IKE from the SPD (for example, the PF_KEY API [PFKEY] uses the SADB_ACQUIRE message). TS payloads specify the selection criteria for packets that will be forwarded over the newly set up SA.

This can serve as a consistency check in some scenarios to assure that the SPDs are consistent. In others, it guides the dynamic update of the SPD.

Two TS payloads appear in each of the messages in the exchange that creates a Child SA pair. Each TS payload contains one or more Traffic Selectors. Each Traffic Selector consists of an address range (IPv4 or IPv6), a port range, and an IP protocol ID.

The first of the two TS payloads is known as TSi (Traffic Selector-initiator). The second is known as TSr (Traffic Selector-responder). TSi specifies the source address of traffic forwarded from (or the destination address of traffic forwarded to) the initiator of the Child SA pair. TSr specifies the destination address of the traffic forwarded to (or the source address of the traffic forwarded from) the responder of the Child SA pair. For example, if the original initiator requests the creation of a Child SA pair, and wishes to tunnel all traffic from subnet 198.51.100.* on the initiator's side to subnet 192.0.2.* on the responder's side, the initiator would include a single Traffic Selector in each TS payload. TSi would specify the address range (198.51.100.0 - 198.51.100.255) and TSr would specify the address range (192.0.2.0 - 192.0.2.255). Assuming that proposal was acceptable to the responder, it would send identical TS payloads back.

IKEv2 allows the responder to choose a subset of the traffic proposed by the initiator. This could happen when the configurations of the two endpoints are being updated but only one end has received the new information. Since the two endpoints may be configured by different people, the incompatibility may persist for an extended period even in the absence of errors. It also allows for intentionally different configurations, as when one end is configured to tunnel all addresses and depends on the other end to have the up-to-date list.

When the responder chooses a subset of the traffic proposed by the initiator, it narrows the Traffic Selectors to some subset of the initiator's proposal (provided the set does not become the null set). If the type of Traffic Selector proposed is unknown, the responder ignores that Traffic Selector, so that the unknown type is not returned in the narrowed set.

To enable the responder to choose the appropriate range in this case, if the initiator has requested the SA due to a data packet, the initiator SHOULD include as the first Traffic Selector in each of TSi and TSr a very specific Traffic Selector including the addresses in the packet triggering the request. In the example, the initiator would include in TSi two Traffic Selectors: the first containing the address range (198.51.100.43 - 198.51.100.43) and the source port and

IP protocol from the packet and the second containing (198.51.100.0 - 198.51.100.255) with all ports and IP protocols. The initiator would similarly include two Traffic Selectors in TSr. If the initiator creates the Child SA pair not in response to an arriving packet, but rather, say, upon startup, then there may be no specific addresses the initiator prefers for the initial tunnel over any other. In that case, the first values in TSi and TSr can be ranges rather than specific values.

The responder performs the narrowing as follows:

- o If the responder's policy does not allow it to accept any part of the proposed Traffic Selectors, it responds with a TS_UNACCEPTABLE Notify message.
- o If the responder's policy allows the entire set of traffic covered by TSi and TSr, no narrowing is necessary, and the responder can return the same TSi and TSr values.
- o If the responder's policy allows it to accept the first selector of TSi and TSr, then the responder MUST narrow the Traffic Selectors to a subset that includes the initiator's first choices. In this example above, the responder might respond with TSi being

(198.51.100.43 - 198.51.100.43) with all ports and IP protocols.

o If the responder's policy does not allow it to accept the first selector of TSi and TSr, the responder narrows to an acceptable subset of TSi and TSr.

When narrowing is done, there may be several subsets that are acceptable but their union is not. In this case, the responder arbitrarily chooses one of them, and MAY include an ADDITIONAL_TS_POSSIBLE notification in the response. The ADDITIONAL_TS_POSSIBLE notification asserts that the responder narrowed the proposed Traffic Selectors but that other Traffic Selectors would also have been acceptable, though only in a separate SA. There is no data associated with this Notify type. This case will occur only when the initiator and responder are configured differently from one another. If the initiator and responder agree on the granularity of tunnels, the initiator will never request a tunnel wider than the responder will accept.

It is possible for the responder's policy to contain multiple smaller ranges, all encompassed by the initiator's Traffic Selector, and with the responder's policy being that each of those ranges should be sent over a different SA. Continuing the example above, the responder might have a policy of being willing to tunnel those addresses to and from the initiator, but might require that each address pair be on a

separately negotiated Child SA. If the initiator didn't generate its request based on the packet, but (for example) upon startup, there would not be the very specific first Traffic Selectors helping the responder to select the correct range. There would be no way for the responder to determine which pair of addresses should be included in this tunnel, and it would have to make a guess or reject the request with a SINGLE_PAIR_REQUIRED Notify message.

The SINGLE_PAIR_REQUIRED error indicates that a CREATE_CHILD_SA request is unacceptable because its sender is only willing to accept Traffic Selectors specifying a single pair of addresses. The requestor is expected to respond by requesting an SA for only the specific traffic it is trying to forward.

Few implementations will have policies that require separate SAs for each address pair. Because of this, if only some parts of the TSi and TSr proposed by the initiator are acceptable to the responder, responders SHOULD narrow the selectors to an acceptable subset rather than use SINGLE_PAIR_REQUIRED.

2.9.1. Traffic Selectors Violating Own Policy

When creating a new SA, the initiator needs to avoid proposing Traffic Selectors that violate its own policy. If this rule is not followed, valid traffic may be dropped. If you use decorrelated policies from [IPSECARCH], this kind of policy violations cannot happen.

This is best illustrated by an example. Suppose that host A has a policy whose effect is that traffic to 198.51.100.66 is sent via host B encrypted using AES, and traffic to all other hosts in 198.51.100.0/24 is also sent via B, but must use CAMELLIA. Suppose also that host B accepts any combination of AES and CAMELLIA.

If host A now proposes an SA that uses CAMELLIA, and includes TSr containing (198.51.100.0 - 198.51.100.255), this will be accepted by host B. Now, host B can also use this SA to send traffic from 198.51.100.66, but those packets will be dropped by A since it requires the use of AES for this traffic. Even if host A creates a new SA only for 198.51.100.66 that uses AES, host B may freely continue to use the first SA for the traffic. In this situation, when proposing the SA, host A should have followed its own policy, and included a TSr containing ((198.51.100.0 - 198.51.100.65), (198.51.100.67 - 198.51.100.255)) instead.

In general, if (1) the initiator makes a proposal "for traffic X (TSi/TSr), do SA", and (2) for some subset X' of X, the initiator does not actually accept traffic X' with SA, and (3) the initiator would be willing to accept traffic X' with some SA' (!=SA), valid traffic can be unnecessarily dropped since the responder can apply either SA or SA' to traffic X'.

2.9.2. Traffic Selectors in Rekeying

Rekeying is used to replace an existing Child SA with another. If the new SA would be allowed to have a narrower set of selectors than the original, traffic that was allowed on the old SA would be dropped in the new SA, thus violating the idea of "replacing". Thus, the new SA MUST NOT have narrower selectors than the original. If the rekeyed SA would ever need to have a narrower scope than the currently used SA, that would mean that the policy was changed in a way such that the currently used SA is against the policy. In that case, the SA should have been already deleted after the policy change took effect.

When the initiator attempts to rekey the Child SA, the proposed Traffic Selectors MUST be the same as the Traffic Selectors used in the old Child SA. That is, they would be the same as the currently active policy.

Because a rekeyed SA can never have a narrower scope than the one currently in use, there is no need for the selectors from the packet, so those selectors SHOULD NOT be sent.

2.10. Nonces

The IKE_SA_INIT messages each contain a nonce. These nonces are used as inputs to cryptographic functions. The CREATE_CHILD_SA request and the CREATE_CHILD_SA response also contain nonces. These nonces are used to add freshness to the key derivation technique used to obtain keys for Child SA, and to ensure creation of strong pseudorandom bits from the Elliptic Curve Diffie-Hellman key. Nonces used in IKEv2 MUST be randomly chosen, MUST be 128 bits in size.

2.11. Address and Port Agility

IKE runs over UDP ports 4500, and implicitly sets up ESP associations for the same IP addresses over which it runs. The IP addresses and ports in the outer header are, however, not themselves cryptographically protected, and IKE is designed to work even through Network Address Translation (NAT) boxes. An implementation MUST accept incoming requests even if the source port is not 500 or 4500, and MUST respond to the address and port from which the request was received. It MUST specify the address and port at which the request was received as the source address and port in the response. IKE functions identically over IPv4 or IPv6.

2.12. Reuse of Diffie-Hellman Exponentials

IKE generates keying material using an ephemeral Elliptic Curve Diffie-Hellman exchange in order to gain the property of "perfect forward secrecy". This means that once a connection is closed and its corresponding keys are forgotten, even someone who has recorded all of the data from the connection and gets access to all of the long-term keys of the two endpoints cannot reconstruct the keys used to protect the conversation without doing a brute force search of the session key space.

Achieving perfect forward secrecy requires that when a connection is closed, each endpoint MUST forget not only the keys used by the connection but also any information that could be used to recompute those keys.

Because computing Elliptic Curve Diffie-Hellman exponentials is computationally expensive, an endpoint may find it advantageous to reuse those exponentials for multiple connection setups. There are several reasonable strategies for doing this. An endpoint could choose a new exponential only periodically though this could result in less-than- perfect forward secrecy if some connection lasts for less than the lifetime of the exponential. Or it could keep track of which exponential was used for each connection and delete the information associated with the exponential only when some corresponding connection was closed. This would allow the exponential to be reused without losing perfect forward secrecy at the cost of maintaining more state.

Whether and when to reuse Elliptic Curve Diffie-Hellman exponentials are private decisions in the sense that they will not affect interoperability. An implementation that reuses exponentials MAY choose to remember the exponential used by the other endpoint on past exchanges and if one is reused to avoid the second half of the calculation. See [REUSE]

and [RFC6989] for a security analysis of this practice and for additional security considerations when reusing ephemeral Elliptic Curve Diffie-Hellman keys.

2.13. Generating Keying Material

In the context of the IKE SA, four cryptographic algorithms are negotiated: an encryption algorithm, an integrity protection algorithm, an Elliptic Curve Diffie-Hellman group, and a pseudorandom function (PRF). The PRF is used for the construction of keying material for all of the cryptographic algorithms used in both the IKE SA and the Child SAs.

We assume that each encryption algorithm and integrity protection algorithm uses a fixed-size key and that any randomly chosen value of that fixed size can serve as an appropriate key. For algorithms that accept a variable-length key, a fixed key size MUST be specified as part of the cryptographic transform negotiated (see Section 3.3.5 for the definition of the Key Length transform attribute). For integrity protection functions based on Hashed Message Authentication Code (HMAC), the fixed key size is the size of the output of the underlying hash function.

It is assumed that PRFs accept keys of any length, but have a preferred key size. The preferred key size MUST be used as the length of SK_d, SK_pi, and SK_pr (see Section 2.14). For PRFs based on the HMAC construction, the preferred key size is equal to the length of the output of the underlying hash function. Other types of PRFs MUST specify their preferred key size.

Keying material will always be derived as the output of the negotiated PRF algorithm. Since the amount of keying material needed may be greater than the size of the output of the PRF, the PRF is used iteratively. The term "prf+" describes a function that outputs a pseudorandom stream based on the inputs to a pseudorandom function called "prf".

In the following, | indicates concatenation. prf+ is defined as:

```
prf+ (K,S) = T1 | T2 | T3 | T4 | ...

where:
T1 = prf (K, S | 0x01)
T2 = prf (K, T1 | S | 0x02)
T3 = prf (K, T2 | S | 0x03)
T4 = prf (K, T3 | S | 0x04)
...
```

This continues until all the material needed to compute all required keys has been output from prf+. The keys are taken from the output string without regard to boundaries (e.g., if the required keys are a 256-bit Advanced Encryption Standard (AES) key and a 256-bit HMAC key, and the prf function generates 256 bits, the AES key will come from T1 and the beginning of T2, while the HMAC key will come from the rest of T2 and the beginning of T3).

The constant concatenated to the end of each prf function is a single octet. The prf+ function SHOULD NOT be used beyond 255 times the size of the prf function output. If ever this happens, the concatenated constant MUST be reset to the value of 1.

2.14. Generating Keying Material for the IKE SA

The shared keys are computed as follows. A quantity called SKEYSEED is calculated from the nonces exchanged during the IKE_SA_INIT exchange and the Elliptic Curve Diffie-Hellman shared secret established during that exchange. SKEYSEED is used to calculate seven other secrets: SK_d used for deriving new keys for the Child SAs established with this IKE SA; SK_ai and SK_ar used as a key to the integrity protection algorithm for authenticating the component messages of subsequent exchanges; SK_ei and SK_er used for encrypting (and of course decrypting) all subsequent exchanges; and SK_pi and SK_pr, which are used when generating an AUTH payload. The lengths of SK_d, SK_pi, and SK_pr MUST be the preferred key length of the PRF agreed upon.

SKEYSEED and its derivatives are computed as follows:

```
   SKEYSEED = prf(Ni | Nr, g^ir)

   {SK_d | SK_ai | SK_ar | SK_ei | SK_er | SK_pi | SK_pr}
                   = prf+ (SKEYSEED, Ni | Nr | SPIi | SPIr)
```

(indicating that the quantities SK_d, SK_ai, SK_ar, SK_ei, SK_er, SK_pi, and SK_pr are taken in order from the generated bits of the prf+). g^ir is the shared secret from the ephemeral Elliptic Curve an Diffie-Hellm exchange. g^ir is represented as a string of octets inan big endian

order padded with zeros if necessary to make it the length of the modulus. Ni and Nr are the nonces, stripped of any headers.

The two directions of traffic flow use different keys. The keys used to protect messages from the original initiator are SK_ai and SK_ei. The keys used to protect messages in the other direction are SK_ar and SK_er.

2.15. Authentication of the IKE SA

The peers are authenticated by having each sign (or MAC using a padded shared secret as the key, as described later in this section) a block of data. In these calculations, IDi' and IDr' are the entire ID payloads excluding the fixed header. For the responder, the octets to be signed start with the first octet of the first SPI in the header of the second message (IKE_SA_INIT response) and end with the last octet of the last payload in the second message. Appended to this (for the purposes of computing the signature) are the initiator's nonce Ni (just the value, not the payload containing it), and the value prf(SK_pr, IDr'). Note that neither the nonce Ni nor the value prf(SK_pr, IDr') are transmitted. Similarly, the initiator signs the first message (IKE_SA_INIT request), starting with the first octet of the first SPI in the header and ending with the last octet of the last payload. Appended to this (for purposes of computing the signature) are the responder's nonce Nr, and the value prf(SK_pi, IDi'). It is critical to the security of the exchange that each side sign the other side's nonce.

The initiator's signed octets can be described as:

```
   InitiatorSignedOctets = RealMessage1 | NonceRData | MACedIDForI
   GenIKEHDR = [ four octets 0 ] | RealIKEHDR
   RealIKEHDR =  SPIi | SPIr |  . . . | Length
   RealMessage1 = RealIKEHDR | RestOfMessage1
   NonceRPayload = PayloadHeader | NonceRData
   InitiatorIDPayload = PayloadHeader | RestOfInitIDPayload
   RestOfInitIDPayload = IDType | RESERVED | InitIDData
   MACedIDForI = prf(SK_pi, RestOfInitIDPayload)
```

The responder's signed octets can be described as:

```
   ResponderSignedOctets = RealMessage2 | NonceIData | MACedIDForR
   GenIKEHDR = [ four octets 0 ] | RealIKEHDR
   RealIKEHDR =  SPIi | SPIr |  . . . | Length
   RealMessage2 = RealIKEHDR | RestOfMessage2
   NonceIPayload = PayloadHeader | NonceIData
   ResponderIDPayload = PayloadHeader | RestOfRespIDPayload
   RestOfRespIDPayload = IDType | RESERVED | RespIDData
   MACedIDForR = prf(SK_pr, RestOfRespIDPayload)
```

Note that all of the payloads are included under the signature, including any payload types not defined in this document. If the first message of the exchange is sent multiple times (such as with a responder cookie and/or a different Elliptic Curve Diffie-Hellman group), it is the latest version of the message that is signed.

Optionally, messages 3 and 4 MAY include a certificate, or certificate chain providing evidence that the key used to compute a digital signature belongs to the name in the ID payload. The signature or MAC will be computed using algorithms dictated by the type of key used by the signer, and specified by the Auth Method field in the Authentication payload. There is no requirement that the initiator and responder sign with the same cryptographic algorithms. The choice of cryptographic algorithms depends on the type of key each has. In particular, the initiator may be using a shared key while the responder may have a public signature key and certificate. It will commonly be the case (but it is not required) that, if a shared secret is used for authentication, the same key is used in both directions.

Shared keys MUST NOT come from a user-chosen password without incorporating another source of randomness. It is typically insecure to use user-chosen passwords because they are unlikely to have sufficient unpredictability to resist dictionary attacks and these attacks are not prevented in this authentication method. (Applications using password-based authentication for bootstrapping and IKE SA should use the authentication method in Section 2.16, which is designed to prevent off-line dictionary attacks.) The pre-shared key needs to contain as much unpredictability as the strongest key being negotiated. In the case of a pre-shared key, the AUTH value is computed as:

```
For the initiator:
   AUTH = prf( prf(Shared Secret, "Key Pad for IKEv2"),
                    <InitiatorSignedOctets>)
```

```
For the responder:
   AUTH = prf( prf(Shared Secret, "Key Pad for IKEv2"),
                    <ResponderSignedOctets>)
```

where the string "Key Pad for IKEv2" is 17 ASCII characters without null termination. The shared secret can be variable length. The pad string is added so that if the shared secret is derived from a password, the IKE implementation need not store the password in cleartext, but rather can store the value prf(Shared Secret,"Key Pad for IKEv2"), which could not be used as a password equivalent for protocols other than IKEv2. As noted above, deriving the shared secret from a password is not secure. This construction is used because it is anticipated that people will do it anyway. The management interface by which the shared secret is provided MUST accept ASCII strings of at least 64 octets and MUST NOT add a null terminator before using them as shared secrets. It MUST also accept a hex encoding of the shared secret. The management interface MAY accept other encodings if the algorithm for translating the encoding to a binary string is specified.

2.16. Extensible Authentication Protocol Methods

2.17. Generating Keying Material for Child SAs

Keying material for Child SA created in CREATE_CHILD_SA exchanges under protection of a given IKE_SA is generated as follows:

```
KEYMAT = prf+(SK_d, g^ir (new) | Ni | Nr)
```

Where Ni and Nr are the nonces from the IKE_SA_INIT exchange if this request is the first Child SA created or the fresh Ni and Nr from the CREATE_CHILD_SA exchange if this is a subsequent creation.

g^ir (new) is the shared secret from the ephemeral Diffie-Hellman exchange of this CREATE_CHILD_SA exchange (represented as an octet string in big endian order padded with zeros in the high-order bits if necessary to make it the length of the modulus).

A single CREATE_CHILD_SA negotiation may result in multiple Security Associations. ESP SAs exist in pairs (one in each direction), so two SAs are created in a single Child SA negotiation for them. Keying material for each Child SA MUST be taken from the expanded KEYMAT using the following rules:

- o All keys for SAs carrying data from the initiator to the responder are taken before SAs going from the responder to the initiator.

- o If an IPsec protocol requires multiple keys, the order in which they are taken from the SA's keying material needs to be described in the protocol's specification. For ESP, [IPSECARCH] defines the order, namely: the encryption key MUST be taken from the first bits and the integrity key MUST be taken from the remaining bits.

Each cryptographic algorithm takes a fixed number of bits of keying material specified as part of the algorithm, or negotiated in SA payloads (see Section 2.13 for description of key lengths, and Section 3.3.5 for the definition of the Key Length transform attribute).

2.18. Rekeying IKE SAs Using a CREATE_CHILD_SA Exchange

The CREATE_CHILD_SA exchange can be used to rekey an existing IKE SA (see Sections 1.3.2 and 2.8). New initiator and responder SPIs are supplied in the SPI fields in the Proposal structures inside the Security Association (SA) payloads (not the SPI fields in the IKE header). The TS payloads are omitted when rekeying an IKE SA. SKEYSEED for the new IKE SA is computed using SK_d from the existing IKE SA as follows:

```
SKEYSEED = prf(SK_d (old), g^ir (new) | Ni | Nr)
```

where g^ir (new) is the shared secret from the ephemeral Diffie-Hellman exchange of this CREATE_CHILD_SA exchange (represented as an octet string in big endian order padded with zeros if necessary to make it the length of the modulus) and Ni and Nr are the two nonces stripped of any headers.

The old and new IKE SA may have selected a different PRF. Because the rekeying exchange belongs to the old IKE SA, it is the old IKE SA's PRF that is used to generate SKEYSEED.

The main reason for rekeying the IKE SA is to ensure that the compromise of old keying material does not provide information about the current keys, or vice versa. Therefore, implementations MUST perform a new Elliptic Curve Diffie-Hellman exchange when rekeying the IKE SA. In other words, an initiator MUST NOT propose the value "NONE" for the Elliptic Curve Diffie-Hellman transform, and a responder MUST NOT accept such a proposal. This means that a successful exchange rekeying the IKE SA always includes the KEi/KEr payloads.

The new IKE SA MUST reset its message counters to 0.

SK_d, SK_ai, SK_ar, SK_ei, and SK_er are computed from SKEYSEED as specified in Section 2.14, using SPIi, SPIr, Ni, and Nr from the new exchange, and using the new IKE SA's PRF.

2.19. Requesting an Internal Address on a Remote Network

Most commonly occurring in the endpoint-to-security-gateway scenario, an endpoint may need an IP address in the network protected by the security gateway and may need to have that address dynamically assigned. A request for such a temporary address can be included in any request to create a Child SA (including the implicit request in message 3) by including a CP payload. Note, however, it is usual to only assign one IP address during the IKE_AUTH exchange. That address persists at least until the deletion of the IKE SA.

This function provides address allocation to an IPsec Remote Access Client (IRAC) trying to tunnel into a network protected by an IPsec Remote Access Server (IRAS). Since the IKE_AUTH exchange creates an IKE SA and a Child SA, the IRAC MUST request the IRAS-controlled address (and optionally other information concerning the protected network) in the IKE_AUTH exchange. The IRAS may procure an address for the IRAC from any number of sources such as a DHCP/BOOTP (Bootstrap Protocol) server or its own address pool.

```
Initiator                         Responder
-------------------------------------------------------------------
 HDR, SK {IDi, [CERT,]
    [CERTREQ,] [IDr,] AUTH,
    CP(CFG_REQUEST), SAi2,
    TSi, TSr}  -->
                             <--  HDR, SK {IDr, [CERT,] AUTH,
                                      CP(CFG_REPLY), SAr2,
                                      TSi, TSr}
```

In all cases, the CP payload MUST be inserted before the SA payload. In variations of the protocol where there are multiple IKE_AUTH exchanges, the CP payloads MUST be inserted in the messages containing the SA payloads.

CP(CFG_REQUEST) MUST contain at least an INTERNAL_ADDRESS attribute (either IPv4 or IPv6) but MAY contain any number of additional attributes the initiator wants returned in the response.

For example, message from initiator to responder:

```
CP(CFG_REQUEST)=
  INTERNAL_ADDRESS()
TSi = (0, 0-65535, 0.0.0.0-255.255.255.255)
TSr = (0, 0-65535, 0.0.0.0-255.255.255.255)
```

NOTE: Traffic Selectors contain (protocol, port range, address range).

Message from responder to initiator:

```
CP(CFG_REPLY)=
  INTERNAL_ADDRESS(192.0.2.202)
  INTERNAL_NETMASK(255.255.255.0)
  INTERNAL_SUBNET(192.0.2.0/255.255.255.0)
TSi = (0, 0-65535, 192.0.2.202-192.0.2.202)
TSr = (0, 0-65535, 192.0.2.0-192.0.2.255)
```

All returned values will be implementation dependent. As can be seen in the above example, the IRAS MAY also send other attributes that were not included in CP(CFG_REQUEST) and MAY ignore the non-mandatory attributes that it does not support.

The responder MUST NOT send a CFG_REPLY without having first received a CP(CFG_REQUEST) from the initiator, because we do not want the IRAS to perform an unnecessary configuration lookup if the IRAC cannot process the REPLY.

In the case where the IRAS's configuration requires that CP be used for a given identity IDi, but IRAC has failed to send a CP(CFG_REQUEST), IRAS MUST fail the request, and terminate the Child SA creation with a FAILED_CP_REQUIRED error. The FAILED_CP_REQUIRED is not fatal to the IKE SA; it simply causes the Child SA creation to fail. The initiator can fix this by later starting a new Configuration payload request. There is no associated data in the FAILED_CP_REQUIRED error.

2.20. Requesting the Peer's Version

2.21. Error Handling

There are many kinds of errors that can occur during IKE processing. The general rule is that if a request is received that is badly formatted, or unacceptable for reasons of policy (such as no matching cryptographic algorithms), the response contains a Notify payload indicating the error. The decision whether or not to send such a response depends whether or not there is an authenticated IKE SA.

If there is an error parsing or processing a response packet, the implementation MUST not send back any error message because responses should not generate new requests (and a new request would be the only way to send back an error message). Such errors in parsing or processing response packets MUST cause the recipient to clean up the IKE state (for example, by sending a Delete for a bad SA).

Only authentication failures (AUTHENTICATION_FAILED) and malformed messages (INVALID_SYNTAX) lead to a deletion of the IKE SA without requiring an explicit INFORMATIONAL exchange carrying a Delete payload. Other error conditions MAY require such an exchange if policy dictates that this is needed.

2.21.1. Error Handling in IKE_SA_INIT

Errors that occur before a cryptographically protected IKE SA is established need to be handled very carefully. There is a trade-off between wanting to help the peer to diagnose a problem and thus responding to the error and wanting to avoid being part of a DoS attack based on forged messages.

In an IKE_SA_INIT exchange, any error notification causes the exchange to fail. Note that some error notifications such as COOKIE, INVALID_KE_PAYLOAD or INVALID_MAJOR_VERSION may lead to a subsequent successful exchange. Because all error notifications are completely unauthenticated, the recipient should continue trying for some time before giving up. The recipient MUST not immediately act based on the error notification unless corrective actions are defined in this specification, such as for COOKIE, INVALID_KE_PAYLOAD, and INVALID_MAJOR_VERSION.

2.21.2. Error Handling in IKE_AUTH

All errors that occur in an IKE_AUTH exchange, causing the authentication to fail for whatever reason (invalid shared secret, invalid ID, untrusted certificate issuer, revoked or expired certificate, etc.) MUST result in an AUTHENTICATION_FAILED notification. If the error occurred on the responder, the notification is returned in the protected response, and MUST be the only payload in that response. Although the IKE_AUTH messages are encrypted and integrity protected, if the peer receiving this notification has not authenticated the other end yet, that peer needs to treat the information with caution.

If the error occurs on the initiator, the notification MAY be returned in a separate INFORMATIONAL exchange, usually with no other payloads. This is an exception for the general rule of not starting new exchanges based on errors in responses.

Note, however, that request messages that contain an unsupported critical payload, or where the whole message is malformed (rather than just bad payload contents), MUST be rejected in their entirety, and MUST only lead to an UNSUPPORTED_CRITICAL_PAYLOAD or INVALID_SYNTAX Notification sent as a response. The receiver should not verify the payloads related to authentication in this case.

If authentication has succeeded in the IKE_AUTH exchange, the IKE SA is established; however, requesting configuration information may still fail. This failure does not automatically cause the IKE SA to be deleted. Specifically, a responder may include all the payloads associated with authentication

(IDr, CERT, and AUTH) while sending error notifications for the piggybacked exchanges (FAILED_CP_REQUIRED, NO_PROPOSAL_CHOSEN, and so on), and the initiator MUST NOT fail the authentication because of this. The initiator MAY, of course, for reasons of policy later delete such an IKE SA.

In an IKE_AUTH exchange, or in the INFORMATIONAL exchange immediately following it (in case an error happened when processing a response to IKE_AUTH), the UNSUPPORTED_CRITICAL_PAYLOAD, INVALID_SYNTAX, and AUTHENTICATION_FAILED notifications are the only ones to cause the IKE SA to be deleted or not created, without a Delete payload. Extension documents may define new error notifications with these semantics, but MUST NOT use them unless the peer has been shown to understand them, such as by using the Vendor ID payload.

2.21.3. Error Handling after IKE SA is Authenticated

After the IKE SA is authenticated, all requests having errors MUST result in a response notifying the other end of the error.

In normal situations, there should not be cases where a valid response from one peer results in an error situation in the other peer, so there should not be any reason for a peer to send error messages to the other end except as a response. Because sending such error messages as an INFORMATIONAL exchange might lead to further errors that could cause loops, such errors SHOULD NOT be sent. If errors are seen that indicate that the peers do not have the same state, it might be good to delete the IKE SA to clean up state and start over.

If a peer parsing a request notices that it is badly formatted (after it has passed the message authentication code checks and window checks) and it returns an INVALID_SYNTAX notification, then this error notification is considered fatal in both peers, meaning that the IKE SA is deleted without needing an explicit Delete payload.

2.21.4. Error Handling Outside IKE SA

A node needs to limit the rate at which it will send messages in response to unprotected messages.

If a node receives a message on UDP 4500 outside the context of an IKE SA known to it (and the message is not a request to start an IKE SA), this may be the result of a recent crash of the node. If the message is marked as a response, the node can audit the suspicious event but MUST NOT respond. If the message is marked as a request, the node can audit the suspicious event and MUST NOT send a response.

2.22. IPComp

2.23. NAT Traversal

Network Address Translation (NAT) gateways are a controversial subject. This section briefly describes what they are and how they are likely to act on IKE traffic. Many people believe that NATs are evil and that we should not design our protocols so as to make them work better. IKEv2 does indeed specify some unintuitive processing rules so that NATs are more likely to work.

NATs exist primarily because of the shortage of IPv4 addresses, though there are other rationales. IP nodes that are "behind" a NAT have IP addresses that are not globally unique, but rather are assigned from some space that is unique within the network behind the NAT but that are likely to be reused by nodes behind other NATs. Generally, nodes behind NATs can communicate with other nodes behind the same NAT and with nodes with globally unique addresses, but not with nodes behind other NATs. There are exceptions to that rule. When those nodes make connections to nodes on the real Internet, the NAT gateway "translates" the IP source address to an address that will be routed back to the gateway. Messages to the gateway from the Internet have their destination addresses "translated" to the internal address that will route the packet to the correct endnode.

NATs are designed to be "transparent" to endnodes. Neither software on the node behind the NAT nor the node on the Internet requires modification to communicate through the NAT. Achieving this transparency is more difficult with some protocols than with others. Protocols that include IP addresses of the endpoints within the payloads of the packet will fail unless the NAT gateway understands the protocol and modifies the internal references as well as those in the headers. Such knowledge is inherently unreliable, is a network layer violation, and often results in subtle problems.

Opening an IPsec connection through a NAT introduces special problems. In tunnel mode, there are routing problems because transparently translating the addresses of ESP packets requires special logic in the NAT and that logic is heuristic and unreliable in nature. For that reason, IKEv2 will use UDP encapsulation of IKE and ESP packets. This encoding is slightly less efficient but is easier for NATs to process. In addition, firewalls may be configured to pass UDP-encapsulated IPsec traffic but not plain, unencapsulated ESP or vice versa.

It is a common practice of NATs to translate TCP and UDP port numbers as well as addresses and use the port numbers of inbound packets to decide which internal node should get a given packet. For this reason, even though IKE packets MUST be sent to and from UDP port 4500, they MUST be accepted coming from any port and responses MUST be sent to the port from whence they came. This is because the ports may be modified as the packets pass through NATs. Similarly, IP addresses of the IKE endpoints are generally not included in the IKE payloads because the payloads are cryptographically protected and could not be transparently modified by NATs.

Port 4500 is reserved for UDP-encapsulated ESP and IKE. An IPsec endpoint MUST send all subsequent traffic from port 4500.

An initiator MUST use port 4500 for both IKE and ESP, regardless of whether or not there is a NAT, even at the beginning of IKE. UDP encapsulation MUST NOT be done on port 500. Implementations MUST NOT decide to not use UDP encapsulation for ESP irrespective of the choice made by the other side.

Support for NAT traversal is then not necessary and MUST NOT be implemented.

o To tunnel IKE packets over UDP port 4500, the IKE header has four octets of zeros prepended and the result immediately follows the UDP header. To tunnel ESP packets over UDP port 4500, the ESP header immediately follows the UDP header. Since the first four octets of the ESP header contain the SPI, and the SPI cannot validly be zero, it is always possible to distinguish ESP and IKE messages.

o There are cases where a NAT box decides to remove mappings that are still alive (for example, the keepalive interval is too long, or the NAT box is rebooted). This will be apparent to a host if it receives a packet whose integrity protection validates, but has

a different port, address, or both from the one that was associated with the SA in the validated packet. When such a validated packet is found, a host that does not support other methods of recovery such as IKEv2 Mobility and Multihoming (MOBIKE) [MOBIKE], and that is not behind a NAT, SHOULD send all packets (including retransmission packets) to the IP address and port in the validated packet, and SHOULD store this as the new address and port combination for the SA (that is, they SHOULD dynamically update the address). A host behind a NAT SHOULD NOT do this type of dynamic address update if a validated packet has different port and/or address values because it opens a possible DoS attack (such as allowing an attacker to break the connection with a single packet). Also, dynamic address update should only be done in response to a new packet; otherwise, an attacker can revert the addresses with old replayed packets. Because of this, dynamic updates can only be done safely if replay protection is enabled. When IKEv2 is used with MOBIKE, dynamically updating the addresses described above interferes with MOBIKE's way of recovering from the same situation. See Section 3.8 of [MOBIKE] for more information.

2.24. Explicit Congestion Notification (ECN)

IKEv2 requires that ECN be usable in the outer IP headers of all tunnel mode Child SAs created by IKEv2. Specifically, tunnel encapsulators and decapsulators for all tunnel mode SAs created by IKEv2 MUST support the ECN full-functionality option for tunnels specified in [ECN] and MUST implement the tunnel encapsulation and decapsulation processing specified in [IPSECARCH] to prevent discarding of ECN congestion indications.

2.25. Exchange Collisions

Because IKEv2 exchanges can be initiated by either peer, it is possible that two exchanges affecting the same SA partly overlap. This can lead to a situation where the SA state information is temporarily not synchronized, and a peer can receive a request that it cannot process in a normal fashion.

Obviously, using a window size greater than 1 leads to more complex situations, especially if requests are processed out of order. This section concentrates on problems that can arise even with a window size of 1, and recommends solutions.

A TEMPORARY_FAILURE notification MUST be sent when a peer receives a request that cannot be completed due to a temporary condition such as a rekeying operation. When a peer receives a TEMPORARY_FAILURE notification, it MUST NOT immediately retry the operation; it MUST wait so that the sender may complete whatever operation caused the temporary condition. The recipient MAY retry the request one or more times over a period of several minutes. If a peer continues to receive TEMPORARY_FAILURE on the same IKE SA after several minutes, it SHOULD conclude that the state information is out of sync and close the IKE SA.

A CHILD_SA_NOT_FOUND notification MUST be sent when a peer receives a request to rekey a Child SA that does not exist. The SA that the initiator attempted to rekey is indicated by the SPI field in the Notify payload, which is copied from the SPI field in the REKEY_SA notification. A peer that receives a CHILD_SA_NOT_FOUND notification SHOULD silently delete the Child SA (if it still exists) and send a request to create a new Child SA from scratch (if the Child SA does not yet exist).

2.25.1. Collisions while Rekeying or Closing Child SAs

If a peer receives a request to rekey a Child SA that it is currently trying to close, it MUST reply with TEMPORARY_FAILURE. If a peer receives a request to rekey a Child SA that it is currently rekeying, it MUST reply as usual, and SHOULD prepare to close redundant SAs later based on the nonces (see Section 2.8.1). If a peer receives a request to rekey a Child SA that does not exist, it MUST reply with CHILD_SA_NOT_FOUND.

If a peer receives a request to close a Child SA that it is currently trying to close, it MUST reply without a Delete payload (see Section 1.4.1). If a peer receives a request to close a Child SA that it is currently rekeying, it MUST reply as usual, with a Delete payload. If a peer receives a request to close a Child SA that does not exist, it MUST reply without a Delete payload.

If a peer receives a request to rekey the IKE SA, and it is currently creating, rekeying, or closing a Child SA of that IKE SA, it MUST reply with TEMPORARY_FAILURE.

2.25.2. Collisions while Rekeying or Closing IKE SAs

If a peer receives a request to rekey an IKE SA that it is currently rekeying, it MUST reply as usual, and SHOULD prepare to close redundant SAs and move inherited Child SAs later based on the nonces (see Section 2.8.2). If a peer receives a request to rekey an IKE SA that it is currently trying to close, it MUST reply with TEMPORARY_FAILURE.

If a peer receives a request to close an IKE SA that it is currently rekeying, it MUST reply as usual, and forget about its own rekeying request. If a peer receives a request to close an IKE SA that it is currently trying to close, it MUST reply as usual, and forget about its own close request.

If a peer receives a request to create or rekey a Child SA when it is currently rekeying the IKE SA, it MUST reply with TEMPORARY_FAILURE. If a peer receives a request to delete a Child SA when it is currently rekeying the IKE SA, it MUST reply as usual, with a Delete payload.

3. Header and Payload Formats

3.1. The IKE Header

IKE messages use UDP port 4500, with one IKE message per UDP datagram. Information from the beginning of the packet through the UDP header is largely ignored except that the IP addresses and UDP ports from the headers are reversed and used for return packets. When sent on UDP port 4500, IKE messages have prepended four octets of zeros. These four octets of zeros are not part of the IKE message and are not included in any of the length fields or checksums defined by IKE. Each IKE message begins with the IKE header, denoted HDR in this document. Following the header are one or more IKE payloads each identified by a Next Payload field in the preceding payload. Payloads are identified in the order in which they appear in an IKE message by looking in the Next Payload field in the IKE header, and subsequently according to the Next Payload field in the IKE payload itself until a Next Payload field of zero indicates that no payloads follow. If a payload of type "Encrypted" is found, that payload is decrypted and its contents parsed as additional payloads. An Encrypted payload MUST be the last payload in a packet and an Encrypted payload MUST NOT contain another Encrypted payload.

The responder's SPI in the header identifies an instance of an IKE Security Association. It is therefore possible for a single instance of IKE to multiplex distinct sessions with multiple peers, including multiple sessions per peer.

All multi-octet fields representing integers are laid out in big endian order (also known as "most significant byte first", or "network byte order").

   The format of the IKE header is shown in Figure 4.

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       IKE SA Initiator's SPI                  |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       IKE SA Responder's SPI                  |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  Next Payload | MjVer | MnVer | Exchange Type |     Flags     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          Message ID                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                            Length                             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

                      Figure 4: IKE Header Format

   o Initiator's SPI (8 octets) - A value chosen by the initiator to
      identify a unique IKE Security Association. This value MUST NOT
      be zero.

   o Responder's SPI (8 octets) - A value chosen by the responder to
      identify a unique IKE Security Association. This value MUST be
      zero in the first message of an IKE initial exchange (including
      repeats of that message including a cookie).

   o Next Payload (1 octet) - Indicates the type of payload that
      immediately follows the header. The format and value of each
      payload are defined below.

   o Major Version (4 bits) - Indicates the major version of the IKE
      protocol in use. Implementations based on this version of IKE
      MUST set the major version to 2. Implementations based on this
      document's version (version 2) of IKE MUST reject or ignore
      messages containing a version number greater than 2 with an
      INVALID_MAJOR_VERSION notification message as described in
      Section 2.5.

   o Minor Version (4 bits) - Indicates the minor version of the IKE
      protocol in use. Implementations based on this version of IKE
      MUST set the minor version to 0. They MUST ignore the minor
      version number of received messages.

o Exchange Type (1 octet) - Indicates the type of exchange being used. This constrains the payloads sent in each message in an exchange. The values in the following table are only current as of the publication date of RFC 4306. Other values may have been added since then or will be added after the publication of this document. Readers should refer to [IKEV2IANA] for the latest values.

Exchange Type	Value
IKE_SA_INIT	34
IKE_AUTH	35
CREATE_CHILD_SA	36
INFORMATIONAL	37

o Flags (1 octet) - Indicates specific options that are set for the message. Presence of options is indicated by the appropriate bit in the flags field being set. The bits are as follows:

```
  +-+-+-+-+-+-+-+-+
  |X|X|R|V|I|X|X|X|
  +-+-+-+-+-+-+-+-+
```

In the description below, a bit being 'set' means its value is '1', while 'cleared' means its value is '0'. 'X' bits MUST be cleared when sending and MUST be ignored on receipt.

* R (Response) - This bit indicates that this message is a response to a message containing the same Message ID. This bit MUST be cleared in all request messages and MUST be set in all responses. An IKE endpoint MUST NOT generate a response to a message that is marked as being a response (with one exception; see Section 2.21.2).

* V (Version) - This bit indicates that the transmitter is capable of speaking a higher major version number of the protocol than the one indicated in the major version number field. Implementations of IKEv2 MUST clear this bit when sending and MUST ignore it in incoming messages.

* I (Initiator) - This bit MUST be set in messages sent by the original initiator of the IKE SA and MUST be cleared in messages sent by the original responder. It is used by the recipient to determine which eight octets of the SPI were generated by the recipient. This bit changes to reflect who initiated the last rekey of the IKE SA.

- o Message ID (4 octets, unsigned integer) - Message identifier used to control retransmission of lost packets and matching of requests and responses. It is essential to the security of the protocol because it is used to prevent message replay attacks. See Sections 2.1 and 2.2.

- o Length (4 octets, unsigned integer) - Length of the total message (header + payloads) in octets.

3.2. Generic Payload Header

Each IKE payload defined in Sections 3.3 through 3.16 begins with a generic payload header, shown in Figure 5. Figures for each payload below will include the generic payload header, but for brevity, the description of each field will be omitted.

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Next Payload  |C|  RESERVED   |         Payload Length        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5: Generic Payload Header

The Generic Payload Header fields are defined as follows:

- o Next Payload (1 octet) - Identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0. This field provides a "chaining" capability whereby additional payloads can be added to a message by appending each one to the end of the message and setting the Next Payload field of the preceding payload to indicate the new payload's type. An Encrypted payload, which must always be the last payload of a message, is an exception. It contains data structures in the format of additional payloads. In the header of an Encrypted payload, the Next Payload field is set to the payload type of the first contained payload (instead of 0); conversely, the Next Payload field of the last contained payload is set to zero. The payload type values are listed here.

Next Payload Type	Notation	Value
No Next Payload		0

Security Association	SA	33
Key Exchange	KE	34
Identification - Initiator	IDi	35
Identification - Responder	IDr	36
Certificate	CERT	37
Certificate Request	CERTREQ	38
Authentication	AUTH	39
Nonce	Ni, Nr	40
Notify	N	41
Delete	D	42
Vendor ID	V	43
Traffic Selector - Initiator	TSi	44
Traffic Selector - Responder	TSr	45
Encrypted and Authenticated	SK	46
Configuration	CP	47

(Payload type values 1-32 should not be assigned in the future so that there is no overlap with the code assignments for IKEv1.)

o Critical (1 bit) - MUST be set to zero if the sender wants the recipient to skip this payload if it does not understand the payload type code in the Next Payload field of the previous payload. MUST be set to one if the sender wants the recipient to reject this entire message if it does not understand the payload type. MUST be ignored by the recipient if the recipient understands the payload type code. MUST be set to zero for payload types defined in this document. Note that the critical bit applies to the current payload rather than the "next" payload whose type code appears in the first octet. The reasoning behind not setting the critical bit for payloads defined in this document is that all implementations MUST understand all payload types defined in this document and therefore must ignore the critical bit's value. Skipped payloads are expected to have valid Next Payload and Payload Length fields. See Section 2.5 for more information on this bit.

o RESERVED (7 bits) - MUST be sent as zero; the field MUST be ignored on receipt.

o Payload Length (2 octets, unsigned integer) - Length in octets of the current payload, including the generic payload header.

Many payloads contain fields marked as "RESERVED". Some payloads in IKEv2 (and historically in IKEv1) are not aligned to 4-octet boundaries.

3.3. Security Association Payload

The Security Association payload, denoted SA in this document, is used to negotiate attributes of a Security Association. Assembly of Security Association payloads requires great peace of mind. An SA payload MAY contain multiple proposals. If there is more than one, they MUST be ordered from most preferred to least preferred. Each proposal contains a single IPsec protocol (ESP or IKE) and MAY contain multiple transforms, each transform MAY contain multiple attributes. When parsing an SA, an implementation MUST check that the total Payload Length is consistent with the payload's internal lengths and counts. Proposals, Transforms, and Attributes each have their own variable-length encodings. They are nested such that the Payload Length of an SA includes the combined contents of the SA, Proposal, Transform, and Attribute information. The length of a Proposal includes the lengths of all Transforms and Attributes it contains. The length of a Transform includes the lengths of all Attributes it contains.

The Proposal structure contains within it a Proposal Num and an IPsec protocol ID. Each structure MUST have a proposal number one (1) greater than the previous structure. The first Proposal in the initiator's SA payload MUST have a Proposal Num of one (1). One reason to use multiple proposals is to propose both standard crypto ciphers and combined-mode ciphers. Combined-mode ciphers include both a single encryption algorithm and MUST NOT offer an integrity algorithm set to "NONE". If an initiator wants to propose both combined-mode ciphers and normal ciphers, it must include two proposals: one will have all the combined-mode ciphers, and the other will have all

the normal ciphers with the integrity algorithms. For example, one such proposal would have two proposal structures. Proposal 1 is ESP with AES-256 in CTR mode, HMAC-SHA2-256-128 as the integrity algorithm, and 256-bit random ECP group or BrainpoolP256r1 as the Elliptic Curve Diffie-Hellman Group; Proposal 2 is AES-256 in GCM mode with an 16-octet Integrity Check Value (ICV) with 256-bit random ECP group or BrainpoolP256r1. Both proposals require the use of ESNs (Extended Sequence Numbers). This can be illustrated as:

```
   SA Payload
      |
      +--- Proposal #1 ( Proto ID = ESP(3), SPI size = 4,
      |     |            5 transforms,      SPI = 0x052357bb )
```

```
    |     |
    |     +-- Transform ENCR ( Name = ENCR_AES_CTR )
    |     |     +-- Attribute ( Key Length = 256 )
    |     +-- Transform INTEG ( Name = AUTH_HMAC_SHA2_256_128 )
    |     +-- Transform D-H ( Name = 256-bit random ECP group )
    |     +-- Transform D-H ( Name = brainpoolP256r1 )
    |     +-- Transform ESN ( Name = ESNs )
    |
    +--- Proposal #2 ( Proto ID = ESP(3), SPI size = 4,
          |            4 transforms,      SPI = 0x35a1d6f2 )
          |
          +-- Transform ENCR ( Name = AES-GCM with a 16 octet ICV )
          |     +-- Attribute ( Key Length = 256 )
          +-- Transform D-H ( Name = 256-bit random ECP group )
          +-- Transform D-H ( Name = brainpoolP256r1 )
          +-- Transform ESN ( Name = ESNs )
```

Each Proposal/Protocol structure is followed by one or more transform structures. The number of different transforms is generally determined by the Protocol. ESP has systematically four: ESN, an encryption algorithm, an integrity check algorithm and a Elliptic Curve Diffie-Hellman Group. IKE has also systematically four transforms: a Elliptic Curve Diffie-Hellman group, an integrity check algorithm, a PRF algorithm,

and an encryption algorithm. For each Protocol, the set of permissible transforms is assigned Transform ID numbers, which appear in the header of each transform.

If there are multiple transforms with the same Transform Type, the proposal is an OR of those transforms. If there are multiple transforms with different Transform Types, the proposal is an AND of the different groups.

A given transform MAY have one or more Attributes. Attributes are necessary when the transform can be used in more than one way, as when an encryption algorithm has a variable key size. The transform would specify the algorithm and the attribute would specify the key size. Most transforms do not have attributes. A transform MUST NOT have multiple attributes of the same type. To propose alternate values for an attribute (for example, multiple key sizes for the AES encryption algorithm), an implementation MUST include multiple transforms with the same Transform Type each with a single Attribute.

```
                      1                   2                   3
```

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                          <Proposals>                          ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: Security Association Payload

o Proposals (variable) - One or more proposal substructures.

The payload type for the Security Association payload is thirty-three (33).

3.3.1. Proposal Substructure

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Last Substruc |   RESERVED    |         Proposal Length       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Proposal Num  |  Protocol ID  |    SPI Size   |Num  Transforms|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                        SPI (variable)                         ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                        <Transforms>                           ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: Proposal Substructure

o Last Substruc (1 octet) - Specifies whether or not this is the last Proposal Substructure in the SA. This field has a value of 0 if this was the last Proposal Substructure, and a value of 2 if there are more Proposal Substructures.

o RESERVED (1 octet) - MUST be sent as zero; MUST be ignored on receipt.

o Proposal Length (2 octets, unsigned integer) - Length of this proposal, including all transforms and attributes that follow.

o Proposal Num (1 octet) - When a proposal is made, the first proposal in an SA payload MUST be 1, and subsequent proposals MUST be one more than the previous proposal (indicating an OR of the two proposals). When a proposal is accepted, the proposal number in the SA payload MUST match the number on the proposal sent that was accepted.

o Protocol ID (1 octet) - Specifies the IPsec protocol identifier for the current negotiation.

Protocol	Protocol ID
IKE	1
ESP	3

o SPI Size (1 octet) - For an initial IKE SA negotiation, this field MUST be zero; the SPI is obtained from the outer header. During subsequent negotiations, it is equal to the size, in octets, of the SPI of the corresponding protocol (8 for IKE, 4 for ESP).

o Num Transforms (1 octet) - Specifies the number of transforms in this proposal.

o SPI (variable) - The sending entity's SPI. Even if the SPI Size is not a multiple of 4 octets, there is no padding applied to the payload. When the SPI Size field is zero, this field is not present in the Security Association payload.

o Transforms (variable) - One or more transform substructures.

3.3.2. Transform Substructure

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Last Substruc |   RESERVED    |        Transform Length       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Transform Type |   RESERVED    |          Transform ID         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                      Transform Attributes                     ~
|                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 8: Transform Substructure

o Last Substruc (1 octet) - Specifies whether or not this is the last Transform Substructure in the Proposal. This field has a value of 0 if this was the last Transform Substructure, and a

value of 3 if there are more Transform Substructures.

o RESERVED - MUST be sent as zero; MUST be ignored on receipt.

o Transform Length - The length (in octets) of the Transform Substructure including Header and Attributes.

o Transform Type (1 octet) - The type of transform being specified in this transform. Different protocols support different Transform Types. For some protocols, some of the transforms may be optional. If a transform is optional and the initiator wishes to propose that the transform be omitted, no transform of the given type is included in the proposal. If the initiator wishes to make use of the transform optional to the responder, it includes a transform substructure with Transform ID = 0 as one of the options.

o Transform ID (2 octets) - The specific instance of the Transform Type being proposed.

The Transform Type values are listed below.

Description	Trans. Type	Used In
Encryption Algorithm (ENCR)	1	IKE and ESP
Pseudorandom Function (PRF)	2	IKE
Integrity Algorithm (INTEG)	3	IKE and ESP
Elliptic Curve Diffie-Hellman Group (D-H)	4	IKE and ESP
Extended Sequence Numbers (ESN)	5	ESP

For Transform Type 1 (Encryption Algorithm), the Transform IDs are listed below.

Name	Number	Defined In
ENCR_AES_CTR	13	[RFC3686]
AES-GCM with a 16 octet ICV	20	

For Transform Type 2 (Pseudorandom Function), the Transform IDs are listed below.

Name	Number	Defined In
PRF_HMAC_SHA2_256	5	[RFC4868]

For Transform Type 3 (Integrity Algorithm), defined Transform IDs are listed below.

Name	Number	Defined In
AUTH_HMAC_SHA2_256_128	12	[RFC4868]

For Transform Type 4 (Elliptic Curve Diffie-Hellman group), defined Transform IDs are listed below.

Name	Number	Defined In
256-bit random ECP group	19	[RFC6989]
brainpoolP256r1	28	[RFC6989]

Although ESP does not directly include a Diffie-Hellman exchange, a Elliptic Curve Diffie-Hellman group MUST be negotiated for the Child SA. This allows the peers to employ Elliptic Curve Diffie-Hellman in the CREATE_CHILD_SA exchange, providing perfect forward secrecy for the generated Child SA keys.

Elliptic curve groups need to have some additional tests performed on them to use them securely. See "Additional Diffie-Hellman Tests for IKEv2" ([RFC6989]) for more information.

For Transform Type 5 (Extended Sequence Numbers), defined Transform IDs are listed below.

Name	Number
Extended Sequence Numbers	1

A proposal containing a single ESN transform with value "1" means that using normal (non-extended) sequence numbers is not acceptable.

3.3.3. Valid Transform Types by Protocol

The number and type of transforms that accompany an SA payload are dependent on the protocol in the SA itself. An SA payload proposing the establishment of an SA has the following mandatory Transform Types. A compliant implementation MUST understand all mandatory for each protocol it supports (though it need not accept proposals with unacceptable suites).

Protocol	Mandatory Types
IKE	ENCR, PRF, INTEG, D-H
ESP	ENCR, ESN, INTEG, D-H

3.3.4. Mandatory Transform IDs

The implementation MUST reject SA proposals that are not authorized by these IKE suite controls. Cryptographic suites that are implemented MUST be configured as acceptable to local policy.

3.3.5. Transform Attributes

Each transform in a Security Association payload may include attributes that modify or complete the specification of the transform. The set of valid attributes depends on the transform.

Currently, only a single attribute type is defined: the Key Length
attribute is used by certain encryption transforms with variable-
length keys (see below for details).

The attributes are type/value pairs and are defined below.
Attributes can have a value with a fixed two-octet length or a
variable-length value. For the latter, the attribute is encoded as
type/length/value.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|A|       Attribute Type        |    AF=0  Attribute Length     |
|F|                             |    AF=1  Attribute Value      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   AF=0  Attribute Value                       |
|                   AF=1  Not Transmitted                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 9: Data Attributes

- o Attribute Format (AF) (1 bit) - Indicates whether the data attribute follows the Type/Length/Value (TLV) format or a shortened Type/Value (TV) format. If the AF bit is zero (0), then the attribute uses TLV format; if the AF bit is one (1), the TV format (with two-byte value) is used.

- o Attribute Type (15 bits) - Unique identifier for each type of attribute (see below).

- o Attribute Value (variable length) - Value of the attribute associated with the attribute type. If the AF bit is a zero (0), this field has a variable length defined by the Attribute Length field. If the AF bit is a one (1), the Attribute Value has a length of 2 octets.

The only currently defined attribute type (Key Length) is fixed
length. Attributes described as fixed length MUST NOT be encoded
using the variable-length encoding unless that length exceeds two
bytes. Variable-length attributes MUST NOT be encoded as
fixed-length even if their value can fit into two octets.

The values in the following table are only current as of the
publication date of RFC 4306. Other values may have been added since
then or will be added after the publication of this document.

Readers should refer to [IKEV2IANA] for the latest values.

Attribute Type	Value	Attribute Format
Key Length (in bits)	14	TV

Values 0-13 and 15-17 were used in a similar context in IKEv1, and should not be assigned except to matching values.

The Key Length attribute specifies the key length in bits (MUST use network byte order) for certain transforms as follows:

- o The Key Length attribute MUST NOT be used with transforms that use a fixed-length key. For example, all the Type 2 (Pseudorandom Function) and Type 3 (Integrity Algorithm) transforms specified in this document. It is recommended that future Type 2 or 3 transforms do not use this attribute.

- o Some transforms specify that the Key Length attribute MUST be always included (omitting the attribute is not allowed, and proposals not containing it MUST be traited as an error). For example, this includes ENCR_AES_CBC and ENCR_AES_CTR.

3.3.6. Attribute Negotiation

During Security Association negotiation initiators present offers to responders. Responders MUST select a single complete set of parameters from the offers (or reject all offers if none are acceptable). If there are multiple proposals, the responder MUST choose a single proposal. If the selected proposal has multiple transforms with the same type, the responder MUST choose a single one. Any attributes of a selected transform MUST be returned unmodified. The initiator of an exchange MUST check that the accepted offer is consistent with one of its proposals, and if not MUST terminate the exchange.

If the responder receives a proposal that contains a Transform Type it does not understand, or a proposal that is missing a mandatory Transform Type, it MUST consider this proposal unacceptable; however, other proposals in the same SA payload are processed as usual. Similarly, if the responder receives a transform that it does not understand, or one that contains a Transform Attribute it does not understand, it MUST consider this transform unacceptable; other transforms with the same Transform Type are processed as usual. This

allows new Transform Types and Transform Attributes to be defined in
the future.

Negotiating Elliptic Curve Diffie-Hellman groups presents some
special challenges. SA offers include proposed attributes and a
Elliptic Curve Diffie-Hellman public number (KE) in the same message.
In the initial exchange the initiator offers to use all required
Elliptic Curve Diffie-Hellman groups, it SHOULD pick the one the
responder is most likely to accept and

include a KE corresponding to that group. If the responder selects a
proposal using a different Elliptic Curve Diffie-Hellman group, the
responder will indicate the correct group in the response and the
initiator SHOULD pick an element of that group for its KE value when
retrying the first message. It MUST, however, continue to propose its
full supported set of groups in order to prevent a man-in-the-middle
downgrade attack.

3.4. Key Exchange Payload

The Key Exchange payload, denoted KE in this document, is used to
exchange Elliptic Curve Diffie-Hellman public numbers as part of a
Elliptic Curve Diffie-Hellman key exchange. The Key Exchange payload
consists of the IKE generic payload header followed by the Elliptic
Curve Diffie-Hellman public value itself.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   ECDH Group Num              |           RESERVED            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                       Key Exchange Data                       ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 10: Key Exchange Payload Format

A Key Exchange payload is constructed by copying one's Elliptic Curve
Diffie-Hellman public value into the "Key Exchange Data" portion of
the payload.

The Diffie-Hellman Group Num identifies the Elliptic Curve
Diffie-Hellman group in which the Key Exchange Data was computed (see
Section 3.3.2). This Diffie-Hellman Group Num MUST match a Elliptic

Curve Diffie-Hellman group specified in a proposal in the SA payload
that is sent in the same message, and SHOULD match the Elliptic Curve
Diffie-Hellman group in the first group in the first proposal, if
such exists. If none of the proposals in that SA payload specifies a
Elliptic Curve Diffie-Hellman group, the KE payload MUST NOT be

present. If the selected proposal uses a different Elliptic Curve
Diffie-Hellman group, the message MUST be rejected with a Notify
payload of type INVALID_KE_PAYLOAD. See also Sections 1.2 and 2.7.

The payload type for the Key Exchange payload is thirty-four (34).

3.5. Identification Payloads

The Identification payloads, denoted IDi and IDr in this document,
allow peers to assert an identity to one another. This identity may
be used for policy lookup, but does not necessarily have to match
anything in the CERT payload; both fields may be used by an
implementation to perform access control decisions. When using the
ID_IPV4_ADDR/ID_IPV6_ADDR identity types in IDi/IDr payloads, IKEv2
does not require this address to match the address in the IP header
of IKEv2 packets, or anything in the TSi/TSr payloads. The contents
of IDi/IDr are used purely to fetch the policy and authentication
data related to the other party.

The Peer Authorization Database (PAD) as described in RFC 4301
[IPSECARCH] describes the use of the ID payload in IKEv2 and provides
a formal model for the binding of identity to policy in addition to
providing services that deal more specifically with the details of
policy enforcement. The PAD is intended to provide a link between
the SPD and the IKE Security Association management. See
Section 4.4.3 of RFC 4301 for more details.

The Identification payload consists of the IKE generic payload header
followed by identification fields as follows:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   ID Type     |                 RESERVED                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                   Identification Data                         ~
```

```
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 11: Identification Payload Format

o ID Type (1 octet) - Specifies the type of Identification being used.

o RESERVED - MUST be sent as zero; MUST be ignored on receipt.

o Identification Data (variable length) - Value, as indicated by the Identification Type. The length of the Identification Data is computed from the size in the ID payload header.

The payload types for the Identification payload are thirty-five (35) for IDi and thirty-six (36) for IDr.

The following table lists the assigned semantics for the Identification Type field. The values in the following table are only current as of the publication date of RFC 4306. Other values may have been added since then or will be added after the publication of this document. Readers should refer to [IKEV2IANA] for the latest values.

```
ID Type                           Value
-----------------------------------------------------------------

ID_IPV4_ADDR                        1
   A single four (4) octet IPv4 address.

ID_FQDN                             2
   A fully-qualified domain name string.  An example of an ID_FQDN
   is "example.com".  The string MUST NOT contain any terminators
   (e.g., NULL, CR, etc.).  All characters in the ID_FQDN are ASCII;
   for an "internationalized domain name", the syntax is as defined
   in [IDNA], for example "xn--tmonesimerkki-bfbb.example.net".

ID_RFC822_ADDR                      3
   A fully-qualified RFC 822 email address string.  An example of a
   ID_RFC822_ADDR is "jsmith@example.com".  The string MUST NOT
   contain any terminators.  Because of [EAI], implementations would
   be wise to treat this field as UTF-8 encoded text, not as
   pure ASCII.

ID_IPV6_ADDR                        5
   A single sixteen (16) octet IPv6 address.
ID_DER_ASN1_DN                      9
   The binary Distinguished Encoding Rules (DER) encoding of an
   ASN.1 X.500 Distinguished Name [PKIX].

ID_DER_ASN1_GN                      10
   The binary DER encoding of an ASN.1 X.509 GeneralName [PKIX].
```

```
   ID_KEY_ID                           11
      An opaque octet stream that may be used to pass vendor-
      specific information necessary to do certain proprietary
      types of identification.
```

Two implementations will interoperate only if each can generate a type of ID acceptable to the other. To assure maximum interoperability, implementations MUST be configurable to send at least one of ID_IPV4_ADDR, ID_FQDN, ID_RFC822_ADDR, or ID_KEY_ID, and MUST be configurable to accept all of these four types. Implementations SHOULD be capable of generating and accepting all of these types. IPv6-capable implementations MUST additionally be configurable to accept ID_IPV6_ADDR. IPv6-only implementations MAY be configurable to send only ID_IPV6_ADDR instead of ID_IPV4_ADDR for IP addresses.

See "The Internet IP Security PKI Profile of IKEv1/ISAKMP, IKEv2, and PKIX" ([RFC4945]) for more information about matching Identification payloads and the contents of the PKIX Certificates.

3.6. Certificate Payload

The Certificate payload, denoted CERT in this document, provides a means to transport certificates or other authentication-related information via IKE. Certificate payloads SHOULD be included in an exchange if certificates are available to the sender. Note that the term "Certificate payload" is somewhat misleading, because not all authentication mechanisms use certificates and data other than certificates may be passed in this payload.

The Certificate payload is defined as follows:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Cert Encoding |                                               |
+-+-+-+-+-+-+-+-+                                               |
~                       Certificate Data                        ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 12: Certificate Payload Format

o Certificate Encoding (1 octet) - This field indicates the type of certificate or certificate-related information contained in the Certificate Data field.

Certificate Encoding	Value
X.509 Certificate - Signature	4
Certificate Revocation List (CRL)	7

o Certificate Data (variable length) - Actual encoding of certificate data. The type of certificate is indicated by the Certificate Encoding field.

The payload type for the Certificate payload is thirty-seven (37).

Specific syntax for some of the certificate type codes above is not defined in this document. The types whose syntax is defined in this document are:

o "X.509 Certificate - Signature" contains a DER-encoded X.509 certificate whose public key is used to validate the sender's AUTH payload. Note that with this encoding, if a chain of certificates needs to be sent, multiple CERT payloads are used, only the first of which holds the public key used to validate the sender's AUTH payload.

   o "Certificate Revocation List" contains a DER-encoded X.509
      certificate revocation list.

Implementations MUST be capable of being configured to send and accept up to four X.509 certificates in support of authentication. If multiple certificates are sent, the first certificate MUST contain the public key associated with the private key used to sign the AUTH payload. The other certificates MUST be sent in order. Self-signed (root) certificates MUST NOT be sent.

3.7. Certificate Request Payload

The Certificate Request payload, denoted CERTREQ in this document, provides a means to request preferred certificates via IKE and can appear in the IKE_INIT_SA response and/or the IKE_AUTH request. Certificate Request payloads MAY be included in an exchange when the sender needs to get the certificate of the receiver.

The Certificate Request payload is defined as follows:

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Next Payload  |C|  RESERVED   |         Payload Length        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Cert Encoding |                                               |
   +-+-+-+-+-+-+-+-+                                               |
   ~                    Certification Authority                    ~
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 13: Certificate Request Payload Format

   o Certificate Encoding (1 octet) - Contains an encoding of the type
      or format of certificate requested. Values are listed in
      Section 3.6.

   o Certification Authority (variable length) - Contains an encoding
      of an acceptable certification authority for the type of
      certificate requested.

The payload type for the Certificate Request payload is thirty-eight (38).

The Certificate Encoding field has the same values as those defined in Section 3.6. The Certification Authority field contains an indicator of trusted authorities for this certificate type. The Certification Authority value is a concatenated list of SHA-2 hashes of the public keys of trusted Certification Authorities (CAs). Each is encoded as the SHA-2 hash of the Subject Public Key Info element (see Section 4.1.2.7 of [PKIX]) from each Trust Anchor certificate. The 32-octet hashes are concatenated and included with no other formatting.

The contents of the Certification Authority field are defined only for X.509 certificates, which are types 4.

Note that the term "Certificate Request" is somewhat misleading, in that values other than certificates are defined in a "Certificate" payload and requests for those values can be present in a Certificate Request payload. The syntax of the Certificate Request payload in such cases is not defined in this document.

The Certificate Request payload is processed by inspecting the Cert Encoding field to determine whether the processor has any certificates of this type. If so, the Certification Authority field is inspected to determine if the processor has any certificates that can be validated up to one of the specified certification authorities. This can be a chain of certificates.

If an end-entity certificate exists that satisfies the criteria specified in the CERTREQ, a certificate or certificate chain SHOULD be sent back to the certificate requestor if the recipient of the CERTREQ:

- o is configured to use certificate authentication,
- o is allowed to send a CERT payload,
- o has matching CA trust policy governing the current negotiation, and
- o has at least one time-wise and usage-appropriate end-entity certificate chaining to a CA provided in the CERTREQ.

Certificate revocation checking must be considered during the chaining process used to select a certificate. Note that even if two peers are configured to use two different CAs, cross-certification relationships should be supported by appropriate selection logic.

The intent is not to prevent communication through the strict adherence of selection of a certificate based on CERTREQ, when an alternate certificate could be selected by the sender that would still enable the recipient to successfully validate and trust it through trust conveyed by cross-certification, CRLs, or other out-of-band configured means. Thus, the processing of a CERTREQ should be seen as a suggestion for a certificate to select, not a mandated one. If no certificates exist, then the CERTREQ is ignored. This is not an error condition of the protocol. There may be cases where there is a preferred CA sent in the CERTREQ, but an alternate might be acceptable (perhaps after prompting a human operator).

3.8. Authentication Payload

The Authentication payload, denoted AUTH in this document, contains data used for authentication purposes. The syntax of the Authentication Data varies according to the Auth Method as specified below.

The Authentication payload is defined as follows:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Auth Method   |                RESERVED                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                      Authentication Data                      ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 14: Authentication Payload Format

o Auth Method (1 octet) - Specifies the method of authentication used. The types of signatures are listed here.

Mechanism	Value

ECDSA on secp256r1 with SHA256	9
ECDSA on BrainpoolP256r1 with SHA256	214
ECSDSA on secp256r1 with SHA256	225
ECSDSA on BrainpoolP256r1 with SHA256	228

Shared Key Message Integrity Code   2
   Computed as specified in Section 2.15 using the shared key
   associated with the identity in the ID payload and the
   negotiated PRF.

- RESERVED - MUST be sent as zero; MUST be ignored on receipt.

- Authentication Data (variable length) - see Section 2.15.

The payload type for the Authentication payload is thirty-nine (39).

3.9. Nonce Payload

The Nonce payload, denoted as Ni and Nr in this document for the initiator's and responder's nonce, respectively, contains random data used to guarantee liveness during an exchange and protect against replay attacks.

The Nonce payload is defined as follows:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                            Nonce Data                         ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 15: Nonce Payload Format

   o Nonce Data (variable length) - Contains the random data generated
      by the transmitting entity.

   The payload type for the Nonce payload is forty (40).

   The size of the Nonce Data MUST be between 16 and 256 octets,
   inclusive. Nonce values MUST NOT be reused.

3.10. Notify Payload

   The Notify payload, denoted N in this document, is used to transmit
   informational data, such as error conditions and state transitions,
   to an IKE peer. A Notify payload may appear in a response message
   (usually specifying why a request was rejected), in an INFORMATIONAL
   exchange (to report an error not in an IKE request), or in any other
   message to indicate sender capabilities or to modify the meaning of
   the request.

   The Notify payload is defined as follows:

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Next Payload  |C|  RESERVED   |         Payload Length        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  Protocol ID  |   SPI Size    |      Notify Message Type      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   ~                Security Parameter Index (SPI)                 ~
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   ~                       Notification Data                       ~
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 16: Notify Payload Format

- o Protocol ID (1 octet) - If this notification concerns an existing SA whose SPI is given in the SPI field, this field indicates the type of that SA. For notifications concerning Child SAs, this field MUST contain (3) to indicate ESP. Of the notifications defined in this document, the SPI is included only with INVALID_SELECTORS, REKEY_SA, and CHILD_SA_NOT_FOUND. If the SPI field is empty, this field MUST be sent as zero and MUST be ignored on receipt.

- o SPI Size (1 octet) - Length in octets of the SPI as defined by the IPsec protocol ID or zero if no SPI is applicable. For a notification concerning the IKE SA, the SPI Size MUST be zero and the field must be empty.

- o Notify Message Type (2 octets) - Specifies the type of notification message.

- o SPI (variable length) - Security Parameter Index.

- o Notification Data (variable length) - Status or error data transmitted in addition to the Notify Message Type. Values for this field are type specific (see below).

The payload type for the Notify payload is forty-one (41).

3.10.1. Notify Message Types

Notification information can be error messages specifying why an SA could not be established. It can also be status data that a process managing an SA database wishes to communicate with a peer process.

The table below lists the notification messages and their corresponding values.

Types in the range 0 - 16383 are intended for reporting errors. An implementation receiving a Notify payload with one of these types that it does not recognize in a response MUST assume that the corresponding request has failed entirely. Unrecognized error types in a request and status types in a request or response MUST be ignored, and they should be logged.

Notify payloads with status types MAY be added to any message and MUST be ignored if not recognized. They are intended to indicate capabilities, and as part of SA negotiation, are used to negotiate non-cryptographic parameters.

More information on error handling can be found in Section 2.21.

The values in the following table are only current as of the publication date of RFC 4306, plus two error types added in this document. Other values may have been added since then or will be added after the publication of this document. Readers should refer to [IKEV2IANA] for the latest values.

NOTIFY messages: error types	Value
UNSUPPORTED_CRITICAL_PAYLOAD	1
See Section 2.5.	
INVALID_IKE_SPI	4
See Section 2.21.	
INVALID_MAJOR_VERSION	5
See Section 2.5.	
INVALID_SYNTAX	7
Indicates the IKE message that was received was invalid because some type, length, or value was out of range or because the request was rejected for policy reasons. To avoid a DoS attack using forged messages, this status MUST only be returned for and in an encrypted packet if the Message ID and cryptographic checksum were valid. To avoid leaking information to someone probing a node, this status MUST be sent in response to any error not covered by one of the other status types.	
INVALID_MESSAGE_ID	9
See Section 2.3.	
INVALID_SPI	11
See Section 1.5.	

```
   NO_PROPOSAL_CHOSEN                       14
       None of the proposed crypto suites was acceptable. This can be
       sent in any case where the offered proposals SA values are not
       acceptable for the responder. This can also be used as "generic"
       Child SA error when Child SA cannot be created for some other
       reason. See also Section 2.7.

   INVALID_KE_PAYLOAD                       17
       See Sections 1.2 and 1.3.

   AUTHENTICATION_FAILED                    24
       Sent in the response to an IKE_AUTH message when, for some
       reason, the authentication failed.  There is no associated
       data.  See also Section 2.21.2.

   SINGLE_PAIR_REQUIRED                     34
       See Section 2.9.

   NO_ADDITIONAL_SAS                        35
       See Section 1.3.

   INTERNAL_ADDRESS_FAILURE                 36
       See Section 3.15.4.

   FAILED_CP_REQUIRED                       37
       See Section 2.19.

   TS_UNACCEPTABLE                          38
       See Section 2.9.
```

```
   INVALID_SELECTORS                        39
       MAY be sent in an IKE INFORMATIONAL exchange when a node receives
       an ESP packet whose selectors do not match those of the SA
       on which it was delivered (and that caused the packet to be
       dropped).  The Notification Data contains the start of the
       offending packet (as in ICMP messages) and the SPI field of the
       notification is set to match the SPI of the Child SA.

   TEMPORARY_FAILURE                        43
       See Section 2.25.
```

CHILD_SA_NOT_FOUND See Section 2.25.	44

NOTIFY messages: status types	Value
INITIAL_CONTACT See Section 2.4.	16384
SET_WINDOW_SIZE See Section 2.3.	16385
ADDITIONAL_TS_POSSIBLE See Section 2.9.	16386
COOKIE See Section 2.6.	16390
REKEY_SA See Section 1.3.3.	16393
ESP_TFC_PADDING_NOT_SUPPORTED See Section 1.3.1.	16394
NON_FIRST_FRAGMENTS_ALSO See Section 1.3.1.	16395

3.11. Delete Payload

The Delete payload, denoted D in this document, contains a protocol-specific Security Association identifier that the sender has removed from its Security Association database and is, therefore, no longer valid. Figure 17 shows the format of the Delete payload. It is possible to send multiple SPIs in a Delete payload; however, each SPI MUST be for the same protocol. Mixing of protocol identifiers MUST NOT be performed in the Delete payload. It is permitted, however, to include multiple Delete payloads in a single INFORMATIONAL exchange where each Delete payload lists SPIs for a different protocol.

Deletion of the IKE SA is indicated by a protocol ID of 1 (IKE) but no SPIs. Deletion of a Child SA (ESP), will contain the IPsec protocol ID of that protocol (3 for ESP), and the SPI is the SPI the sending endpoint would expect in inbound ESP packets.

The Delete payload is defined as follows:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Protocol ID   |   SPI Size    |          Num of SPIs          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~               Security Parameter Index(es) (SPI)              ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 17: Delete Payload Format

o Protocol ID (1 octet) - Must be 1 for an IKE SA or 3 for ESP.

o SPI Size (1 octet) - Length in octets of the SPI as defined by the protocol ID. It MUST be zero for IKE (SPI is in message header) or four for ESP.

o Num of SPIs (2 octets, unsigned integer) - The number of SPIs contained in the Delete payload. The size of each SPI is defined by the SPI Size field.

o Security Parameter Index(es) (variable length) - Identifies the specific Security Association(s) to delete. The length of this field is determined by the SPI Size and Num of SPIs fields.

The payload type for the Delete payload is forty-two (42).

3.12. Vendor ID Payload

The Vendor ID payload, denoted V in this document, contains a vendor-defined constant. The constant is used by vendors to identify and recognize remote instances of their implementations. This mechanism allows a vendor to experiment with new features while maintaining backward compatibility.

A Vendor ID payload MAY announce that the sender is capable of accepting certain extensions to the protocol, or it MAY simply identify the implementation as an aid in debugging. A Vendor ID payload MUST NOT change the interpretation of any information defined in this specification (i.e., the critical bit MUST be set to 0).

Multiple Vendor ID payloads MAY be sent. An implementation is not
required to send any Vendor ID payload at all.

A Vendor ID payload may be sent as part of any message. Reception of
a familiar Vendor ID payload allows an implementation to make use of
private use numbers described throughout this document, such as
private payloads, private exchanges, private notifications, etc.
Unfamiliar Vendor IDs MUST be ignored.

Writers of documents who wish to extend this protocol MUST define a
Vendor ID payload to announce the ability to implement the extension
in the document. It is expected that documents that gain acceptance
and are standardized will be given "magic numbers" out of the Future
Use range by IANA, and the requirement to use a Vendor ID will go
away.

The Vendor ID payload fields are defined as follows:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                        Vendor ID (VID)                        ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 18: Vendor ID Payload Format

o Vendor ID (variable length) - It is the responsibility of the
   person choosing the Vendor ID to assure its uniqueness in spite of
   the absence of any central registry for IDs. A message digest of
   a long unique string MUST instead of the value of the string
   itself.

The payload type for the Vendor ID payload is forty-three (43).

3.13. Traffic Selector Payload

The Traffic Selector payload, denoted TS in this document, allows peers to identify packet flows for processing by IPsec security services. The Traffic Selector payload consists of the IKE generic payload header followed by individual Traffic Selectors as follows:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Number of TSs |                 RESERVED                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                       <Traffic Selectors>                     ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 19: Traffic Selectors Payload Format

o Number of TSs (1 octet) - Number of Traffic Selectors being provided.

o RESERVED - This field MUST be sent as zero and MUST be ignored on receipt.

o Traffic Selectors (variable length) - One or more individual Traffic Selectors.

The length of the Traffic Selector payload includes the TS header and all the Traffic Selectors.

The payload type for the Traffic Selector payload is forty-four (44) for addresses at the initiator's end of the SA and forty-five (45) for addresses at the responder's end.

There is no requirement that TSi and TSr contain the same number of individual Traffic Selectors. Thus, they are interpreted as follows: a packet matches a given TSi/TSr if it matches at least one of the individual selectors in TSi, and at least one of the individual selectors in TSr.

For instance, the following Traffic Selectors:

TSi = ((17, 100, 198.51.100.66-198.51.100.66),

```
       (17, 200, 198.51.100.66-198.51.100.66))
TSr = ((17, 300, 0.0.0.0-255.255.255.255),
       (17, 400, 0.0.0.0-255.255.255.255))
```

would match UDP packets from 198.51.100.66 to anywhere, with any of the four combinations of source/destination ports (100,300), (100,400), (200,300), and (200, 400).

Thus, some types of policies may require several Child SA pairs. For instance, a policy matching only source/destination ports (100,300) and (200,400), but not the other two combinations, cannot be negotiated as a single Child SA pair.

3.13.1. Traffic Selector

```
                         1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |   TS Type     |IP Protocol ID |       Selector Length         |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |           Start Port          |           End Port            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    ~                         Starting Address                      ~
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    ~                         Ending Address                        ~
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 20: Traffic Selector

o TS Type (one octet) - Specifies the type of Traffic Selector.

o IP protocol ID (1 octet) - Value specifying an associated IP protocol ID (such as UDP, TCP, and ICMP). A value of zero means that the protocol ID is not relevant to this Traffic Selector -- the SA can carry all protocols.

o Selector Length (2 octets, unsigned integer) - Specifies the length of this Traffic Selector substructure including the header.

o Start Port (2 octets, unsigned integer) - Value specifying the smallest port number allowed by this Traffic Selector. For protocols for which port is undefined (including protocol 0), or if all ports are allowed, this field MUST be zero. ICMP and ICMPv6 Type and Code values, as well as Mobile IP version 6 (MIPv6) mobility header (MH) Type values, are represented in this field as specified in Section 4.4.1.1 of [IPSECARCH]. ICMP Type and Code values are treated as a single 16-bit integer port number, with Type in the most significant eight bits and Code in the least significant eight bits. MIPv6 MH Type values are treated as a single 16-bit integer port number, with Type in the most significant eight bits and the least significant eight bits set to zero.

o End Port (2 octets, unsigned integer) - Value specifying the largest port number allowed by this Traffic Selector. For protocols for which port is undefined (including protocol 0), or if all ports are allowed, this field MUST be 65535. ICMP and ICMPv6 Type and Code values, as well as MIPv6 MH Type values, are represented in this field as specified in Section 4.4.1.1 of [IPSECARCH]. ICMP Type and Code values are treated as a single 16-bit integer port number, with Type in the most significant eight bits and Code in the least significant eight bits. MIPv6 MH Type values are treated as a single 16-bit integer port number, with Type in the most significant eight bits and the least significant eight bits set to zero.

o Starting Address - The smallest address included in this Traffic Selector (length determined by TS Type).

o Ending Address - The largest address included in this Traffic Selector (length determined by TS Type).

relire:

Systems that are complying with [IPSECARCH] that wish to indicate

"ANY" ports MUST set the start port to 0 and the end port to 65535; note that according to [IPSECARCH], "ANY" includes "OPAQUE". Systems working with [IPSECARCH] that wish to indicate "OPAQUE" ports, but not "ANY" ports, MUST set the start port to 65535 and the end port to 0.

The Traffic Selector types 7 and 8 can also refer to ICMP or ICMPv6 type and code fields, as well as MH Type fields for the IPv6 mobility header [MIPV6]. Note, however, that neither ICMP nor MIPv6 packets have separate source and destination fields. The method for specifying the Traffic Selectors for ICMP and MIPv6 is shown by example in Section 4.4.1.3 of [IPSECARCH].

The following table lists values for the Traffic Selector Type field and the corresponding Address Selector Data.

TS Type	Value
TS_IPV4_ADDR_RANGE	7

A range of IPv4 addresses, represented by two four-octet values. The first value is the beginning IPv4 address (inclusive) and the second value is the ending IPv4 address (inclusive). All addresses falling between the two specified addresses are considered to be within the list.

TS Type	Value
TS_IPV6_ADDR_RANGE	8

A range of IPv6 addresses, represented by two sixteen-octet values. The first value is the beginning IPv6 address (inclusive) and the second value is the ending IPv6 address

      (inclusive). All addresses falling between the two specified
      addresses are considered to be within the list.

3.14. Encrypted Payload

The Encrypted payload, denoted SK {...} in this document, contains other payloads in encrypted form. The Encrypted payload, if present in a message, MUST be the last payload in the message. Often, it is the only payload in the message. This payload is also called the "Encrypted and Authenticated" payload.

The algorithms for encryption and integrity protection are negotiated during IKE SA setup, and the keys are computed as specified in Sections 2.14 and 2.18.

This document specifies the cryptographic processing of Encrypted payloads using a block cipher in CBC mode and an integrity check algorithm that computes a fixed-length checksum over a variable size message. The design is modeled after the ESP algorithms described in RFCs 2104 [HMAC], 4303 [ESP], and 2451 [ESPCBC]. This document completely specifies the cryptographic processing of IKE data, but those documents should be consulted for design rationale. Future documents may specify the processing of Encrypted payloads for other types of transforms, such as counter mode encryption and authenticated encryption algorithms. Peers MUST NOT negotiate transforms for which no such specification exists.

When an authenticated encryption algorithm is used to protect the IKE SA, the construction of the Encrypted payload is different than what is described here. See [AEAD] for more information on authenticated encryption algorithms and their use in IKEv2.

The payload type for an Encrypted payload is forty-six (46). The Encrypted payload consists of the IKE generic payload header followed by individual fields as follows:

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Next Payload  |C|  RESERVED   |         Payload Length        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                     Initialization Vector                     |
   |         (length is block size for encryption algorithm)       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ~                    Encrypted IKE Payloads                     ~
   +               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |               |             Padding (0-255 octets)            |
   +-+-+-+-+-+-+-+                               +-+-+-+-+-+-+-+
   |                                             |  Pad Length   |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ~                    Integrity Checksum Data                    ~
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 21: Encrypted Payload Format

o Next Payload - The payload type of the first embedded payload. Note that this is an exception in the standard header format, since the Encrypted payload is the last payload in the message and therefore the Next Payload field would normally be zero. But because the content of this payload is embedded payloads and there was no natural place to put the type of the first one, that type is placed here.

o Payload Length - Includes the lengths of the header, initialization vector (IV), Encrypted IKE payloads, Padding, Pad Length, and Integrity Checksum Data.

o Initialization Vector - For CBC mode ciphers, the length of the initialization vector (IV) is equal to the block length of the underlying encryption algorithm. Senders MUST select a new unpredictable IV for every message; recipients MUST accept any value. The reader is encouraged to consult [MODES] for advice on IV generation. In particular, using the final ciphertext block of

the previous message is not considered unpredictable. For modes other than CBC, the IV format and processing is specified in the document specifying the encryption algorithm and mode.

o IKE payloads are as specified earlier in this section. This field is encrypted with the negotiated cipher.

o Padding MUST contain only null bytes, and MUST have a length that makes the combination of the payloads, the Padding, and the Pad Length to be a multiple of the encryption block size. This field is encrypted with the negotiated cipher.

o Pad Length is the length of the Padding field. The sender MUST set the Pad Length to the minimum value that makes the combination of the payloads, the Padding, and the Pad Length a multiple of the block size, but the recipient MUST accept any length that results in proper alignment. This field is encrypted with the negotiated cipher.

- o Integrity Checksum Data is the cryptographic checksum of the entire message starting with the Fixed IKE header through the Pad Length. The checksum MUST be computed over the encrypted message. Its length is determined by the integrity algorithm negotiated.

3.15. Configuration Payload

The Configuration payload, denoted CP in this document, is used to exchange configuration information between IKE peers. The exchange is for an IRAC to request an internal IP address from an IRAS and to exchange other information of the sort that one would acquire with Dynamic Host Configuration Protocol (DHCP) if the IRAC were directly connected to a LAN.

The Configuration payload is defined as follows:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C| RESERVED    |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   CFG Type    |                    RESERVED                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                   Configuration Attributes                    ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 22: Configuration Payload Format

The payload type for the Configuration payload is forty-seven (47).

- o CFG Type (1 octet) - The type of exchange represented by the Configuration Attributes.

CFG Type	Value
CFG_REQUEST	1
CFG_REPLY	2
CFG_SET	3
CFG_ACK	4

- o RESERVED (3 octets) - MUST be sent as zero; MUST be ignored on receipt.

- o Configuration Attributes (variable length) - These are type length

      value (TLV) structures specific to the Configuration payload and
      are defined below. There may be zero or more Configuration
      Attributes in this payload.

3.15.1. Configuration Attributes

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |R|         Attribute Type      |            Length             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   ~                             Value                             ~
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 23: Configuration Attribute Format

- Reserved (1 bit) - This bit MUST be set to zero and MUST be ignored on receipt.

- Attribute Type (15 bits) - A unique identifier for each of the Configuration Attribute Types.

- Length (2 octets, unsigned integer) - Length in octets of value.

- Value (0 or more octets) - The variable-length value of this Configuration Attribute. The following lists the attribute types.

The values in the following table are only current as of the publication date of RFC 4306 (except INTERNAL_ADDRESS_EXPIRY and INTERNAL_IP6_NBNS, which were removed by RFC 5996). Other values may have been added since then or will be added after the publication of this document. Readers should refer to [IKEV2IANA] for the latest values.

Attribute Type	Value	Multi-Valued	Length
INTERNAL_IP4_ADDRESS	1	YES*	0 or 4 octets
INTERNAL_IP4_NETMASK	2	NO	0 or 4 octets
INTERNAL_IP4_DNS	3	YES	0 or 4 octets
INTERNAL_IP4_NBNS	4	YES	0 or 4 octets
INTERNAL_IP4_DHCP	6	YES	0 or 4 octets
APPLICATION_VERSION	7	NO	0 or more
INTERNAL_IP6_ADDRESS	8	YES*	0 or 17 octets
INTERNAL_IP6_DNS	10	YES	0 or 16 octets
INTERNAL_IP6_DHCP	12	YES	0 or 16 octets

```
   INTERNAL_IP4_SUBNET      13     YES           0 or 8 octets
   SUPPORTED_ATTRIBUTES     14     NO            Multiple of 2
   INTERNAL_IP6_SUBNET      15     YES           17 octets
```

* These attributes may be multi-valued on return only if multiple values were requested.

o INTERNAL_IP4_ADDRESS, INTERNAL_IP6_ADDRESS - An address on the internal network, sometimes called a red node address or private address, and it MAY be a private address on the Internet. In a request message, the address specified is a requested address (or a zero-length address if no specific address is requested). If a specific address is requested, it likely indicates that a previous connection existed with this address and the requestor would like to reuse that address. With IPv6, a requestor MAY supply the low-order address octets it wants to use. Multiple internal addresses MAY be requested by requesting multiple internal address attributes. The responder MAY only send up to the number of addresses requested. The INTERNAL_IP6_ADDRESS is made up of two fields: the first is a 16-octet IPv6 address, and the second is a one-octet prefix-length as defined in [ADDRIPV6]. The requested address is valid as long as this IKE SA (or its rekeyed successors) requesting the address is valid. This is described in more detail in Section 3.15.3.

o INTERNAL_IP4_NETMASK - The internal network's netmask. Only one netmask is allowed in the request and response messages (e.g., 255.255.255.0), and it MUST be used only with an INTERNAL_IP4_ADDRESS attribute. INTERNAL_IP4_NETMASK in a CFG_REPLY means roughly the same thing as INTERNAL_IP4_SUBNET

containing the same information ("send traffic to these addresses through me"), but also implies a link boundary. For instance, the client could use its own address and the netmask to calculate the broadcast address of the link. An empty INTERNAL_IP4_NETMASK attribute can be included in a CFG_REQUEST to request this information (although the gateway can send the information even when not requested). Non-empty values for this attribute in a CFG_REQUEST do not make sense and thus MUST NOT be included.

o INTERNAL_IP4_DNS, INTERNAL_IP6_DNS - Specifies an address of a DNS server within the network. Multiple DNS servers MAY be requested. The responder MAY respond with zero or more DNS server attributes.

o INTERNAL_IP4_NBNS - Specifies an address of a NetBios Name Server (WINS) within the network. Multiple NBNS servers MAY be requested. The responder MAY respond with zero or more NBNS server attributes.

o INTERNAL_IP4_DHCP, INTERNAL_IP6_DHCP - Instructs the host to send any internal DHCP requests to the address contained within the attribute. Multiple DHCP servers MAY be requested. The responder MAY respond with zero or more DHCP server attributes.

o APPLICATION_VERSION - The version or application information of the IPsec host. This is a string of printable ASCII characters that is NOT null terminated.

o INTERNAL_IP4_SUBNET - The protected sub-networks that this edge-device protects. This attribute is made up of two fields: the first being an IP address and the second being a netmask. Multiple sub-networks MAY be requested. The responder MAY respond with zero or more sub-network attributes. This is discussed in more detail in Section 3.15.2.

o SUPPORTED_ATTRIBUTES - When used within a Request, this attribute MUST be zero-length and specifies a query to the responder to reply back with all of the attributes that it supports. The response contains an attribute that contains a set of attribute identifiers each in 2 octets. The length divided by 2 (octets) would state the number of supported attributes contained in the response.

o INTERNAL_IP6_SUBNET - The protected sub-networks that this edge-device protects. This attribute is made up of two fields: the first is a 16-octet IPv6 address, and the second is a one-octet prefix-length as defined in [ADDRIPV6]. Multiple sub-networks MAY be requested. The responder MAY respond with zero or more sub-network attributes. This is discussed in more detail in Section 3.15.2.

Note that no recommendations are made in this document as to how an implementation actually figures out what information to send in a response. That is, we do not recommend any specific method of an IRAS determining which DNS server should be returned to a requesting IRAC.

The CFG_REQUEST and CFG_REPLY pair allows an IKE endpoint to request information from its peer. If an attribute in the CFG_REQUEST Configuration payload is not zero-length, it is taken as a suggestion for that attribute. The CFG_REPLY Configuration payload MAY return that value, or a new one. It MAY also add new attributes and not include some requested ones. Unrecognized or unsupported attributes MUST be ignored in both requests and responses.

The CFG_SET and CFG_ACK pair allows an IKE endpoint to push configuration data to its peer. In this case, the CFG_SET Configuration payload contains attributes the initiator wants its peer to alter. The responder MUST return a Configuration payload if it accepted any of the configuration data, and the Configuration payload MUST contain the attributes that the responder accepted with zero-length data. Those attributes that it did not accept MUST NOT be in the CFG_ACK Configuration payload. If no attributes were accepted, the responder MUST return either an empty CFG_ACK payload or a response message without a CFG_ACK payload. There are currently no defined uses for the CFG_SET/CFG_ACK exchange, though they may be used in connection with extensions based on Vendor IDs. An implementation of this specification MAY ignore CFG_SET payloads.

3.15.2. Meaning of INTERNAL_IP4_SUBNET and INTERNAL_IP6_SUBNET

INTERNAL_IP4/6_SUBNET attributes can indicate additional subnets, ones that need one or more separate SAs, that can be reached through the gateway that announces the attributes. INTERNAL_IP4/6_SUBNET attributes may also express the gateway's policy about what traffic should be sent through the gateway; the client can choose whether other traffic (covered by TSr, but not in INTERNAL_IP4/6_SUBNET) is sent through the gateway or directly to the destination. Thus, traffic to the addresses listed in the INTERNAL_IP4/6_SUBNET attributes should be sent through the gateway that announces the attributes. If there are no existing Child SAs whose Traffic Selectors cover the address in question, new SAs need to be created.

For instance, if there are two subnets, 198.51.100.0/26 and 192.0.2.0/24, and the client's request contains the following:

```
CP(CFG_REQUEST) =
  INTERNAL_IP4_ADDRESS()
TSi = (0, 0-65535, 0.0.0.0-255.255.255.255)
TSr = (0, 0-65535, 0.0.0.0-255.255.255.255)
```

then a valid response could be the following (in which TSr and INTERNAL_IP4_SUBNET contain the same information):

```
CP(CFG_REPLY) =
  INTERNAL_IP4_ADDRESS(198.51.100.234)
  INTERNAL_IP4_SUBNET(198.51.100.0/255.255.255.192)
  INTERNAL_IP4_SUBNET(192.0.2.0/255.255.255.0)
TSi = (0, 0-65535, 198.51.100.234-198.51.100.234)
TSr = ((0, 0-65535, 198.51.100.0-198.51.100.63),
       (0, 0-65535, 192.0.2.0-192.0.2.255))
```

In these cases, the INTERNAL_IP4_SUBNET does not really carry any useful information.

A different possible response would have been this:

```
   CP(CFG_REPLY) =
     INTERNAL_IP4_ADDRESS(198.51.100.234)
     INTERNAL_IP4_SUBNET(198.51.100.0/255.255.255.192)
     INTERNAL_IP4_SUBNET(192.0.2.0/255.255.255.0)
   TSi = (0, 0-65535, 198.51.100.234-198.51.100.234)
   TSr = (0, 0-65535, 0.0.0.0-255.255.255.255)
```

That response would mean that the client can send all its traffic through the gateway, but the gateway does not mind if the client sends traffic not included by INTERNAL_IP4_SUBNET directly to the destination (without going through the gateway).

A different situation arises if the gateway has a policy that requires the traffic for the two subnets to be carried in separate SAs. Then a response like this would indicate to the client that if it wants access to the second subnet, it needs to create a separate SA:

```
   CP(CFG_REPLY) =
     INTERNAL_IP4_ADDRESS(198.51.100.234)
     INTERNAL_IP4_SUBNET(198.51.100.0/255.255.255.192)
     INTERNAL_IP4_SUBNET(192.0.2.0/255.255.255.0)
   TSi = (0, 0-65535, 198.51.100.234-198.51.100.234)
   TSr = (0, 0-65535, 198.51.100.0-198.51.100.63)
```

INTERNAL_IP4_SUBNET can also be useful if the client's TSr included only part of the address space. For instance, if the client requests the following:

```
   CP(CFG_REQUEST) =
     INTERNAL_IP4_ADDRESS()
   TSi = (0, 0-65535, 0.0.0.0-255.255.255.255)
   TSr = (0, 0-65535, 192.0.2.155-192.0.2.155)
```

then the gateway's response might be:

```
   CP(CFG_REPLY) =
     INTERNAL_IP4_ADDRESS(198.51.100.234)
     INTERNAL_IP4_SUBNET(198.51.100.0/255.255.255.192)
     INTERNAL_IP4_SUBNET(192.0.2.0/255.255.255.0)
   TSi = (0, 0-65535, 198.51.100.234-198.51.100.234)
   TSr = (0, 0-65535, 192.0.2.155-192.0.2.155)
```

Because the meaning of INTERNAL_IP4_SUBNET/INTERNAL_IP6_SUBNET in CFG_REQUESTs is unclear, they cannot be used reliably in CFG_REQUESTs.

3.15.3. Configuration Payloads for IPv6

The Configuration payloads for IPv6 are based on the corresponding IPv4 payloads, and do not fully follow the "normal IPv6 way of doing things". In particular, IPv6 stateless autoconfiguration or router advertisement messages are not used, neither is neighbor discovery. Note that there is an additional document that discusses IPv6 configuration in IKEv2, [IPV6CONFIG]. At the present time, it is an experimental document, but there is a hope that with more implementation experience, it will gain the same standards treatment as this document.

A client can be assigned an IPv6 address using the INTERNAL_IP6_ADDRESS Configuration payload. A minimal exchange might look like this:

```
   CP(CFG_REQUEST) =
     INTERNAL_IP6_ADDRESS()
     INTERNAL_IP6_DNS()
   TSi = (0, 0-65535, :: - FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF)
   TSr = (0, 0-65535, :: - FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF)

   CP(CFG_REPLY) =
     INTERNAL_IP6_ADDRESS(2001:DB8:0:1:2:3:4:5/64)
     INTERNAL_IP6_DNS(2001:DB8:99:88:77:66:55:44)
   TSi = (0, 0-65535, 2001:DB8:0:1:2:3:4:5 - 2001:DB8:0:1:2:3:4:5)
   TSr = (0, 0-65535, :: - FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF)
```

The client MAY send a non-empty INTERNAL_IP6_ADDRESS attribute in the CFG_REQUEST to request a specific address or interface identifier. The gateway first checks if the specified address is acceptable, and if it is, returns that one. If the address was not acceptable, the gateway attempts to use the interface identifier with some other prefix; if even that fails, the gateway selects another interface identifier.

The INTERNAL_IP6_ADDRESS attribute also contains a prefix length field. When used in a CFG_REPLY, this corresponds to the INTERNAL_IP4_NETMASK attribute in the IPv4 case.

Although this approach to configuring IPv6 addresses is reasonably simple, it has some limitations. IPsec tunnels configured using IKEv2 are not fully featured "interfaces" in the IPv6 addressing architecture sense [ADDRIPV6]. In particular, they do not necessarily have link-local addresses, and this may complicate the use of protocols that assume them, such as [MLDV2].

3.15.4. Address Assignment Failures

If the responder encounters an error while attempting to assign an IP address to the initiator during the processing of a Configuration payload, it responds with an INTERNAL_ADDRESS_FAILURE notification. The IKE SA is still created even if the initial Child SA cannot be created because of this failure. If this error is generated within an IKE_AUTH exchange, no Child SA will be created. However, there are some more complex error cases.

If the responder does not support Configuration payloads at all, it can simply ignore all Configuration payloads. This type of implementation never sends INTERNAL_ADDRESS_FAILURE notifications.

If the initiator requires the assignment of an IP address, it will treat a response without CFG_REPLY as an error.

The initiator may request a particular type of address (IPv4 or IPv6) that the responder does not support, even though the responder supports Configuration payloads. In this case, the responder simply ignores the type of address it does not support and processes the rest of the request as usual.

If the initiator requests multiple addresses of a type that the responder supports, and some (but not all) of the requests fail, the responder replies with the successful addresses only. The responder sends INTERNAL_ADDRESS_FAILURE only if no addresses can be assigned.

If the initiator does not receive the IP address(es) required by its policy, it MAY keep the IKE SA up and retry the Configuration payload as separate INFORMATIONAL exchange after suitable timeout, or it MAY tear down the IKE SA by sending a Delete payload inside a separate INFORMATIONAL exchange and later retry IKE SA from the beginning after some timeout. Such a timeout should not be too short (especially if the IKE SA is started from the beginning) because these error situations may not be able to be fixed quickly; the timeout should likely be several minutes. For example, an address shortage problem on the responder will probably only be fixed when more entries are returned to the address pool when other clients disconnect or when responder is reconfigured with larger address pool.

3.16. Extensible Authentication Protocol (EAP) Payload

4. Conformance Requirements

In order to assure that all implementations of IKEv2 can interoperate, there are "MUST support" requirements in addition to those listed elsewhere. Of course, IKEv2 is a security protocol, and one of its major functions is to allow only authorized parties to successfully complete establishment of SAs. So a particular implementation may be configured with any of a number of restrictions concerning algorithms and trusted authorities that will prevent universal interoperability.

To assure interoperability, all implementations MUST be capable of parsing all payload types (if only to skip over them) and to ignore payload types that it does not support unless the critical bit is set in the payload header. If the critical bit is set in an unsupported payload header, all implementations MUST reject the messages containing those payloads.

Every implementation MUST be capable of doing six-message IKE_SA_INIT, IKE_SA_INIT with a cookie, and IKE_AUTH exchanges establishing one SA (for IKE). Implementations MAY be initiate-only or respond- only if appropriate for their platform. Every implementation MUST be capable of responding to an INFORMATIONAL exchange, but a minimal implementation MAY respond to any request in the INFORMATIONAL exchange with an empty response (note that within the context of an IKE SA, an "empty" message consists of an IKE header followed by an Encrypted payload with no payloads contained in it). A minimal implementation MUST support the CREATE_CHILD_SA exchange only in so far as to recognize requests and reject them with a Notify payload of type NO_ADDITIONAL_SAS. A minimal implementation need not be able to initiate CREATE_CHILD_SA or INFORMATIONAL exchanges. When an SA expires (based on locally configured values of either lifetime or octets passed), an implementation MAY either try to renew it with a CREATE_CHILD_SA exchange or it MAY delete (close) the old SA and create a new one. A compliant implementation MUST be able to rekey any SA type. If the responder rejects the CREATE_CHILD_SA request with a NO_ADDITIONAL_SAS notification, the implementation MUST be capable of instead deleting the old SA and creating a new one.

Implementations are not required to support requesting temporary IP addresses or responding to such requests. If an implementation does support issuing such requests and its policy requires using temporary IP addresses, it MUST include a CP payload in the first message in the IKE_AUTH exchange containing at least a field of type INTERNAL_IP4_ADDRESS or INTERNAL_IP6_ADDRESS. All other fields are optional. If an implementation supports responding to such requests, it MUST parse the CP payload of type CFG_REQUEST in the first message in the IKE_AUTH exchange and recognize a field of type INTERNAL_IP4_ADDRESS or INTERNAL_IP6_ADDRESS. If it supports leasing an address of the appropriate type, it MUST return a CP payload of type CFG_REPLY containing an address of the requested type. The responder may include any other related attributes.

For an implementation to be called conforming to this specification, it MUST be possible to configure it to accept the following:

o Public Key Infrastructure using X.509 (PKIX) Certificates containing and signed by EC keys of size 256, 384 or 512 bits, where the ID passed is any of ID_KEY_ID, ID_FQDN, ID_RFC822_ADDR, or ID_DER_ASN1_DN.

o Shared key authentication where the ID passed is any of ID_KEY_ID, ID_FQDN, or ID_RFC822_ADDR.

o Authentication where the responder is authenticated using PKIX Certificates and the initiator is authenticated using shared key authentication.

5. Security Considerations

The IKE_SA_INIT and IKE_AUTH exchanges happen before the initiator has been authenticated. As a result, an implementation of this protocol needs to be completely robust when deployed on any insecure network. Implementation vulnerabilities, particularly DoS attacks, can be exploited by unauthenticated peers.

When using pre-shared keys, a critical consideration is how to assure the randomness of these secrets. The strongest practice is to ensure that any pre-shared key contain as much randomness as the strongest key being negotiated. Deriving a shared secret from a password, name, or other low-entropy source is not secure. These sources are subject to dictionary and social-engineering attacks, among others.

5.1. Traffic Selector Authorization

IKEv2 relies on information in the Peer Authorization Database (PAD) when determining what kind of Child SAs a peer is allowed to create. This process is described in Section 4.4.3 of [IPSECARCH]. When a peer requests the creation of a Child SA with some Traffic Selectors, the PAD must contain "Child SA Authorization Data" linking the identity authenticated by IKEv2 and the addresses permitted for Traffic Selectors.

For example, the PAD might be configured so that authenticated identity "sgw23.example.com" is allowed to create Child SAs for 192.0.2.0/24, meaning this security gateway is a valid "representative" for these addresses. Host-to-host IPsec requires similar entries, linking, for example, "fooserver4.example.com" with 198.51.100.66/32, meaning this identity is a valid "owner" or "representative" of the address in question.

As noted in [IPSECARCH], "It is necessary to impose these constraints on creation of child SAs to prevent an authenticated peer from spoofing IDs associated with other, legitimate peers". In the example given above, a correct configuration of the PAD prevents sgw23 from creating Child SAs with address 198.51.100.66, and prevents fooserver4 from creating Child SAs with addresses from 192.0.2.0/24.

It is important to note that simply sending IKEv2 packets using some particular address does not imply a permission to create Child SAs with that address in the Traffic Selectors. For example, even if sgw23 would be able to spoof its IP address as 198.51.100.66, it could not create Child SAs matching fooserver4's traffic.

The IKEv2 specification does not specify how exactly IP address assignment using Configuration payloads interacts with the PAD. Our interpretation is that when a security gateway assigns an address using Configuration payloads, it also creates a temporary PAD entry linking the authenticated peer identity and the newly allocated inner address.

It has been recognized that configuring the PAD correctly may be difficult in some environments. For instance, if IPsec is used between a pair of hosts whose addresses are allocated dynamically using DHCP, it is extremely difficult to ensure that the PAD

specifies the correct "owner" for each IP address. This would require a mechanism to securely convey address assignments from the DHCP server, and link them to identities authenticated using IKEv2.

Due to this limitation, some vendors have been known to configure their PADs to allow an authenticated peer to create Child SAs with Traffic Selectors containing the same address that was used for the IKEv2 packets. In environments where IP spoofing is possible (i.e., almost everywhere) this essentially allows any peer to create Child SAs with any Traffic Selectors. This is not an appropriate or secure configuration in most circumstances. See [H2HIPSEC] for an extensive discussion about this issue, and the limitations of host-to-host IPsec in general.

6. IANA Considerations

7. References

7.1. Normative References

[ADDRIPV6] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006, <http://www.rfc-editor.org/info/rfc4291>.

[AEAD]   Black, D. and D. McGrew, "Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol", RFC 5282, August 2008, <http://www.rfc-editor.org/info/rfc5282>.

[AESCMACPRF128]
Song, J., Poovendran, R., Lee, J., and T. Iwata, "The Advanced Encryption Standard-Cipher-based Message Authentication Code-Pseudo-Random Function-128 (AES-CMAC-PRF-128) Algorithm for the Internet Key Exchange Protocol (IKE)", RFC 4615, August 2006, <http://www.rfc-editor.org/info/rfc4615>.

[AESXCBCPRF128]
Hoffman, P., "The AES-XCBC-PRF-128 Algorithm for the Internet Key Exchange Protocol (IKE)", RFC 4434, February 2006, <http://www.rfc-editor.org/info/rfc4434>.

[ECN]   Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001, <http://www.rfc-editor.org/info/rfc3168>.

[ESPCBC]   Pereira, R. and R. Adams, "The ESP CBC-Mode Cipher Algorithms", RFC 2451, November 1998, <http://www.rfc-editor.org/info/rfc2451>.

[IKEV2IANA]
IANA, "Internet Key Exchange Version 2 (IKEv2) Parameters", <http://www.iana.org/assignments/ikev2-parameters/>.

[IPSECARCH]
Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005, <http://www.rfc-editor.org/info/rfc4301>.

[MUSTSHOULD]
Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[PKCS1] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003, <http://www.rfc-editor.org/info/rfc3447>.

[PKIX] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

[RFC4307] Schiller, J., "Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2)", RFC 4307, December 2005, <http://www.rfc-editor.org/info/rfc4307>.

[UDPENCAPS]
Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", RFC 3948, January 2005, <http://www.rfc-editor.org/info/rfc3948>.

7.2. Informative References

[ARCHGUIDEPHIL]
Bush, R. and D. Meyer, "Some Internet Architectural Guidelines and Philosophy", RFC 3439, December 2002, <http://www.rfc-editor.org/info/rfc3439>.

[ARCHPRINC]
Carpenter, B., "Architectural Principles of the Internet", RFC 1958, June 1996, <http://www.rfc-editor.org/info/rfc1958>.

[Clarif] Eronen, P. and P. Hoffman, "IKEv2 Clarifications and Implementation Guidelines", RFC 4718, October 2006, <http://www.rfc-editor.org/info/rfc4718>.

[DH] Diffie, W. and M. Hellman, "New Directions in Cryptography", IEEE Transactions on Information Theory, V.IT-22 n. 6, June 1977.

[DIFFSERVARCH]
Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, December 1998, <http://www.rfc-editor.org/info/rfc2475>.

[DIFFSERVFIELD]
Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998, <http://www.rfc-editor.org/info/rfc2474>.

[DIFFTUNNEL]
Black, D., "Differentiated Services and Tunnels", RFC 2983, October 2000, <http://www.rfc-editor.org/info/rfc2983>.

[DOI] Piper, D., "The Internet IP Security Domain of Interpretation for ISAKMP", RFC 2407, November 1998, <http://www.rfc-editor.org/info/rfc2407>.

[DOSUDPPROT]
Kaufman, C., Perlman, R., and B. Sommerfeld, "DoS protection for UDP-based protocols", ACM Conference on Computer and Communications Security, October 2003.

[EAI] Yang, A., Steele, S., and N. Freed, "Internationalized Email Headers", RFC 6532, February 2012, <http://www.rfc-editor.org/info/rfc6532>.

[ESP] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005, <http://www.rfc-editor.org/info/rfc4303>.

[H2HIPSEC] Aura, T., Roe, M., and A. Mohammed, "Experiences with Host-to-Host IPsec", 13th International Workshop on Security Protocols, Cambridge, UK, April 2005.

[HMAC] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997, <http://www.rfc-editor.org/info/rfc2104>.

[IDNA] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010, <http://www.rfc-editor.org/info/rfc5890>.

[IKEV2] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005, <http://www.rfc-editor.org/info/rfc4306>.

[IP] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981, <http://www.rfc-editor.org/info/rfc791>.

[IPV6CONFIG] Eronen, P., Laganier, J., and C. Madson, "IPv6 Configuration in Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 5739, February 2010, <http://www.rfc-editor.org/info/rfc5739>.

[MAILFORMAT] Resnick, P., Ed., "Internet Message Format", RFC 5322, October 2008, <http://www.rfc-editor.org/info/rfc5322>.

[MIPV6] Perkins, C., Johnson, D., and J. Arkko, "Mobility Support in IPv6", RFC 6275, July 2011, <http://www.rfc-editor.org/info/rfc6275>.

[MLDV2] Vida, R. and L. Costa, "Multicast Listener Discovery

Version 2 (MLDv2) for IPv6", RFC 3810, June 2004, <http://www.rfc-editor.org/info/rfc3810>.

[MOBIKE] Eronen, P., "IKEv2 Mobility and Multihoming Protocol (MOBIKE)", RFC 4555, June 2006, <http://www.rfc-editor.org/info/rfc4555>.

[MODES] Dworkin, M., "Recommendation for Block Cipher Modes of Operation", National Institute of Standards and Technology, NIST Special Publication 800-38A 2001 Edition, December 2001.

[NAI] Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The Network Access Identifier", RFC 4282, December 2005, <http://www.rfc-editor.org/info/rfc4282>.

[PFKEY] McDonald, D., Metz, C., and B. Phan, "PF_KEY Key Management API, Version 2", RFC 2367, July 1998, <http://www.rfc-editor.org/info/rfc2367>.

[REAUTH] Nir, Y., "Repeated Authentication in Internet Key Exchange (IKEv2) Protocol", RFC 4478, April 2006, <http://www.rfc-editor.org/info/rfc4478>.

[REUSE] Menezes, A. and B. Ustaoglu, "On Reusing Ephemeral Keys In Diffie-Hellman Key Agreement Protocols", December 2008, <http://www.cacr.math.uwaterloo.ca/techreports/2008/cacr2008-24.pdf>.

[RFC4945] Korver, B., "The Internet IP Security PKI Profile of IKEv1/ISAKMP, IKEv2, and PKIX", RFC 4945, August 2007, <http://www.rfc-editor.org/info/rfc4945>.

[RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 5996, September 2010, <http://www.rfc-editor.org/info/rfc5996>.

[RFC6989] Sheffer, Y. and S. Fluhrer, "Additional Diffie-Hellman Tests for the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 6989, July 2013, <http://www.rfc-editor.org/info/rfc6989>.

Appendix A. Summary of Changes from IKEv1

Appendix B. Diffie-Hellman Groups

↑

Appendix C. Exchanges and Payloads

This appendix contains a short summary of the IKEv2 exchanges, and what payloads can appear in which message. This appendix is purely informative; if it disagrees with the body of this document, the other text is considered correct.

Vendor ID (V) payloads may be included in any place in any message. This sequence here shows what are the most logical places for them.

C.1. IKE_SA_INIT Exchange

Reprendre dans le détails en fonction de ce qu'on décide (suppression du chiffrmeent opportuniste, etc)

```
   request             --> [N(COOKIE),]
                           SA, KE, Ni,
                           [V+][N+]

   normal response     <-- SA, KE, Nr,
   (no cookie)             [[N(HTTP_CERT_LOOKUP_SUPPORTED),] CERTREQ+,]
                           [N(CHILDLESS_IKEV2_SUPPORTED),]
                           [V+][N+]

   cookie response     <-- N(COOKIE),
                           [N(CHILDLESS_IKEV2_SUPPORTED),]
                           [V+][N+]

   different Diffie-   <-- N(INVALID_KE_PAYLOAD),
   Hellman group           [V+][N+]
   wanted
```

C.2. IKE_AUTH Exchange

```
   request             --> IDi, [CERT+,]
                           [N(INITIAL_CONTACT),]
                           [[N(HTTP_CERT_LOOKUP_SUPPORTED),] CERTREQ+,]
                           IDr,
                           AUTH,
                           [N(ESP_TFC_PADDING_NOT_SUPPORTED),]
                           [N(NON_FIRST_FRAGMENTS_ALSO),]
                           [V+][N+]
   response            <-- IDr, [CERT+,]
                           AUTH,
                           [N(ESP_TFC_PADDING_NOT_SUPPORTED),]
                           [N(NON_FIRST_FRAGMENTS_ALSO),]
                           [N(ADDITIONAL_TS_POSSIBLE),]
                           [V+][N+]

   error in Child SA  <--  IDr, [CERT+,]
   creation                AUTH,
                           N(error),
                           [V+][N+]
```

C.3. IKE_AUTH Exchange with EAP

C.4. CREATE_CHILD_SA Exchange for Creating or Rekeying Child SAs

```
   request             --> [N(REKEY_SA),]
                           [CP(CFG_REQUEST),]
                           [N(ESP_TFC_PADDING_NOT_SUPPORTED),]
                           [N(NON_FIRST_FRAGMENTS_ALSO),]
                           SA, Ni, KEi, TSi, TSr,
                           [V+][N+]

   normal              <-- [CP(CFG_REPLY),]
   response                [N(ESP_TFC_PADDING_NOT_SUPPORTED),]
                           [N(NON_FIRST_FRAGMENTS_ALSO),]
                           SA, Nr, KEr, TSi, TSr,
                           [N(ADDITIONAL_TS_POSSIBLE),]
                           [V+][N+]

   error case          <-- N(error)

   different Diffie-   <-- N(INVALID_KE_PAYLOAD),
   Hellman group           [V+][N+]
   wanted
```

C.5. CREATE_CHILD_SA Exchange for Rekeying the IKE SA

```
   request             --> SA, Ni, KEi,
                           [V+][N+]
   response            <-- SA, Nr, KEr,
                           [V+][N+]
```

C.6. INFORMATIONAL Exchange

```
   request             --> [N+,]
                           [D+,]
                           [CP(CFG_REQUEST)]

   response            <-- [N+,]
                           [D+,]
                           [CP(CFG_REPLY)]
```

Acknowledgements