Network Working Group
Internet-Draft
Expires: July 5, 2006

P. Hoffman
VPN Consortium
January 2006

Internet Key Exchange Protocol: IKEv2.1
draft-hoffman-ikev2-1-00.txt

Status of this Memo

Copyright Notice

Abstract

This document describes version 2.1 of the Internet Key Exchange (IKE) protocol. IKEv2.1 is heavily based on IKEv2 from RFC 4306 (edited by Charlie Kaufman), and includes all of the clarifications from the "IKEv2 Clarifications" document (edited by Pasi Eronen and Paul Hoffman). IKEv2.1 makes additional changes to those two documents in places where IKEv2 was unclear and the clarifications document did not commit to a particular protocol interpretation.

Table of Contents

1. Introduction

{{ An introduction to IKEv2.1 is given at the end of Section 1. It is put there (instead of here) to preserve the section numbering of the original IKEv2 document. }}

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.

Establishing this shared state in a manual fashion does not scale well. Therefore, a protocol to establish this state dynamically is needed. This memo describes such a protocol -- the Internet Key Exchange (IKE). This is version 2.1 of IKE. Version 1 of IKE was defined in RFCs 2407 [DOI], 2408 [ISAKMP], and 2409 [IKEV1]. IKEv2 was defined in [IKEV2]. This single document is intended to replace all three of those RFCs.

Definitions of the primitive terms in this document (such as Security Association or SA) can be found in [IPSECARCH]. {{ Clarif-7.2 }} It should be noted that parts of IKEv2 and IKEv2.1 rely on some of the processing rules in [IPSECARCH], as described in various sections of this document.

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/or Authentication Header (AH) [AH] 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. IKE can also negotiate use of IP Compression (IPComp) [IPCOMP] in connection with an ESP and/or AH SA. We call the IKE SA an "IKE_SA". The SAs for ESP and/or AH 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". We call the first messages establishing an IKE_SA IKE_SA_INIT and IKE_AUTH exchanges and subsequent IKE exchanges CREATE_CHILD_SA 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. In some scenarios, only a single CHILD_SA is needed between the IPsec endpoints, and therefore there would be no additional exchanges. Subsequent exchanges MAY be used to establish additional CHILD_SAs between the same authenticated pair of endpoints and to perform housekeeping functions.

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 request/response of an IKE session (IKE_SA_INIT) negotiates security parameters for the IKE_SA, sends nonces, and sends Diffie-Hellman values.

The second request/response (IKE_AUTH) transmits identities, proves knowledge of the secrets corresponding to the two identities, and sets up an SA for the first (and often only) AH and/or ESP CHILD_SA.

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 should errors occur are described in Section 2.21.

1.1. Usage Scenarios

IKE is expected to be used to negotiate ESP and/or AH SAs in a number of different scenarios, each with its own special requirements.

1.1.1. Security Gateway to Security Gateway Tunnel

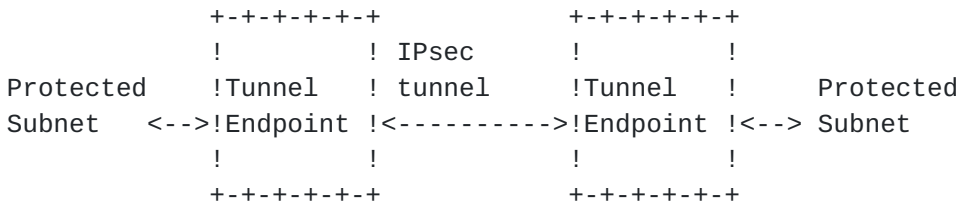

Figure 1: Security Gateway to Security Gateway Tunnel

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

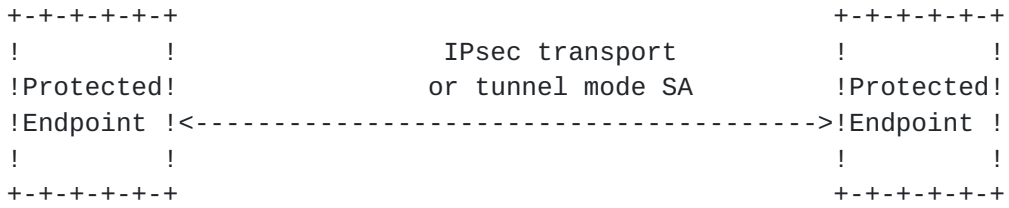

Figure 2: Endpoint to Endpoint

In this scenario, both endpoints of the IP connection implement IPsec, as required of hosts in [IPSECARCH]. Transport mode will commonly be used with no inner IP header. If there is an inner IP header, the inner addresses will be the same as the outer addresses. A single pair of addresses will be negotiated for packets to be protected by this SA. These endpoints MAY implement application layer access controls based on the IPsec authenticated identities of the participants. This scenario enables the end-to-end security that has been a guiding principle for the Internet since [ARCHPRINC], [TRANSPARENCY], and a method of limiting the inherent problems with complexity in networks noted by [ARCHGUIDEPHIL]. Although this scenario may not be fully applicable to the IPv4 Internet, it has been deployed successfully in specific scenarios within intranets using IKEv1. It should be more broadly enabled during the transition to IPv6 and with the adoption of IKEv2.

It is possible in this scenario that one or both of the protected endpoints will be behind a network address translation (NAT) node, in

which case the tunneled packets will have to be UDP encapsulated so that port numbers in the UDP headers can be used to identify individual endpoints "behind" the NAT (see Section 2.23).

1.1.3. Endpoint to Security Gateway Tunnel

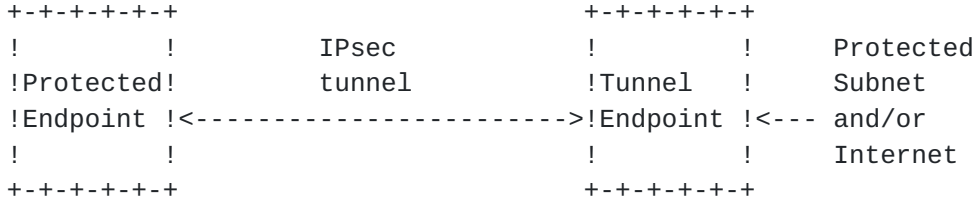

Figure 3: Endpoint to Security Gateway Tunnel

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. {{ Clarif-6.1 }} 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.

1.1.4. Other Scenarios

Other scenarios are possible, as are nested combinations of the above. One notable example combines aspects of 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 (known in IKEv1 as Phase 1). These initial exchanges normally consist of four messages, though in some scenarios that number can grow. 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 Diffie-Hellman exchange [DH].

The second pair of messages (IKE_AUTH) authenticate the previous messages, exchange identities and certificates, and establish the first CHILD_SA. Parts of these messages are encrypted and integrity protected with keys established through the IKE_SA_INIT exchange, so the identities are hidden from eavesdroppers and all fields in all the messages are authenticated.

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
E	Encrypted
EAP	Extensible Authentication
HDR	IKE Header
IDi	Identification - Initiator
IDr	Identification - Responder
KE	Key Exchange
Ni, Nr	Nonce
N	Notify
SA	Security Association
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], indicate that optionally a certificate request payload can be included.

{{ Clarif-7.10 }} Many payloads contain fields marked as "RESERVED" Some payloads in IKEv2 (and historically in IKEv1) are not aligned to 4-byte boundaries.

The initial exchanges are as follows:

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

HDR contains the Security Parameter Indexes (SPIs), version numbers, 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 Diffie-Hellman value. Ni is the initiator's nonce.

```
                              <--  HDR, SAr1, KEr, Nr, [CERTREQ]
```

The responder chooses a cryptographic suite from the initiator's offered choices and expresses that choice in the SAr1 payload, completes the 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 SKEYSEED, from which all keys are derived for that IKE_SA. All but the headers of all the messages that follow are encrypted and integrity protected. 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). 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 DH 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,] AUTH, SAi2,
    TSi, TSr}  -->
```

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 optional payload IDr enables the initiator to specify which of the responder's identities it wants to talk to. This is useful when the machine on which the responder is running is hosting multiple identities at the same IP address. The initiator begins negotiation of a CHILD_SA using the SAi2 payload. The final fields (starting with SAi2) are described in the description of the CREATE_CHILD_SA exchange.

```
                        <--  HDR, SK {IDr, [CERT,] AUTH,
                                SAr2, TSi, TSr}
```

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, and completes negotiation of a CHILD_SA with the additional fields described below in the CREATE_CHILD_SA exchange.

The recipients of messages 3 and 4 MUST verify that all signatures and MACs are computed correctly and that the names in the ID payloads correspond to the keys used to generate the AUTH payload.

{{ Clarif-4.2}} If creating the CHILD_SA during the IKE_AUTH exchange fails for some reason, the IKE_SA is still created as usual. The list of responses in the IKE_AUTH exchange that do not prevent an

IKE_SA from being set up include at least the following: NO_PROPOSAL_CHOSEN, TS_UNACCEPTABLE, SINGLE_PAIR_REQUIRED, INTERNAL_ADDRESS_FAILURE, and FAILED_CP_REQUIRED.

{{ Clarif-4.3 }} Note that IKE_AUTH messages do not contain KEi/KEr or Ni/Nr payloads. Thus, the SA payload in IKE_AUTH exchange cannot contain Transform Type 4 (Diffie-Hellman Group) with any other value than NONE. Implementations MUST leave the transform out entirely in this case.

1.3. The CREATE_CHILD_SA Exchange

{{ This is a heavy rewrite of most of this section. The major organization changes are described in Clarif-4.1 and Clarif-5.1. }}

The CREATE_CHILD_SA exchange is used to create new CHILD_SAs and to rekey both IKE_SAs and CHILD_SAs. This exchange consists of a single request/response pair, and some of its function was referred to as a phase 2 exchange in IKEv1. It MAY be initiated by either end of the IKE_SA after the initial exchanges are completed.

All messages following the initial exchange are cryptographically protected using the cryptographic algorithms and keys negotiated in the first two messages of the IKE exchange. These subsequent messages use the syntax of the Encrypted Payload described in Section 3.14. All subsequent messages included an Encrypted Payload, even if they are referred to in the text as "empty". For both messages in the CREATE_CHILD_SA, 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.

The CREATE_CHILD_SA is used for rekeying IKE_SAs and CHILD_SAs. 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 MAY optionally contain a KE payload for an additional 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 Diffie-Hellman value (if KE payloads are included in the CREATE_CHILD_SA exchange).

If a CREATE_CHILD_SA exchange includes a KEi payload, at least one of the SA offers MUST include the Diffie-Hellman group of the KEi. The Diffie-Hellman group of the KEi MUST be an element of the group the initiator expects the responder to accept (additional Diffie-Hellman groups can be proposed). If the responder rejects the Diffie-Hellman group of the KEi payload, the responder MUST reject the request and indicate its preferred Diffie-Hellman group in the INVALID_KE_PAYLOAD Notification payload. In the case of such a rejection, the CREATE_CHILD_SA exchange fails, and the initiator will probably retry the exchange with a Diffie-Hellman proposal and KEi in the group that the responder gave in the INVALID_KE_PAYLOAD.

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, optionally a 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, and a Diffie-Hellman value in the KEr payload if KEi was included in the request and the selected cryptographic suite includes that group.

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.

1.3.2. **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 Diffie-Hellman value in the KEi payload. New initiator and responder SPIs are supplied in the SPI fields.

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, and a Diffie-Hellman value in the KEr payload if the selected cryptographic suite includes that group.

The new IKE_SA has its message counters set to 0, regardless of what they were in the earlier IKE_SA. The window size starts at 1 for any new IKE_SA.

KEi and KEr are required for rekeying an IKE_SA.

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, SA, Ni, [KEi],
       TSi, TSr}   -->
```

The initiator sends SA offer(s) in the SA payload, a nonce in the Ni payload, optionally a Diffie-Hellman value in the KEi payload, and the proposed traffic selectors for the proposed CHILD_SA in the TSi and TSr payloads. When rekeying an existing CHILD_SA, the leading N payload of type REKEY_SA MUST be included and MUST give the SPI (as they would be expected in the headers of inbound packets) of the SAs being rekeyed.

The CREATE_CHILD_SA response for rekeying a CHILD_SA is:

```
                              <--  HDR, SK {SA, Nr, [KEr],
                                       Si, TSr}
```

The responder replies (using the same Message ID to respond) with the accepted offer in an SA payload, and a Diffie-Hellman value in the KEr payload if KEi was included in the request and the selected cryptographic suite includes that group.

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.

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 which generated them (or its successor if the IKE_SA was replaced for the purpose of rekeying).

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 (else the Sender will assume the message was lost in the network and will retransmit it). That response MAY be a message with no payloads. 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.

{{ Clarif-5.6 }} ESP and AH 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). When SAs are nested, as when data (and IP headers if in tunnel mode) are encapsulated first with IPComp, then with ESP, and finally with AH between the same pair of endpoints, all of the SAs MUST be deleted together. 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. {{ Clarif-5.7 }} Note that you never send delete payloads for the two sides of an SA in a single message. If you have many SAs to delete at the same time (such as for nested SAs), you include delete payloads for in inbound half of each SA in your Informational exchange.

Normally, the reply 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.

{{ Demoted the SHOULD }} Half-closed connections are anomalous and, and a node with auditing capability will probably audit their existence if they persist. Note that this specification nowhere specifies 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; doing so will implicitly close all SAs negotiated under it. It can then rebuild the SAs it needs on a clean base under a new IKE_SA. {{ Clarif-5.8 }} The response to a request that deletes the IKE_SA is an empty Informational response.

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.

1.5. Informational Messages outside of an IKE_SA

If an encrypted IKE packet arrives on port 500 or 4500 with an unrecognized SPI, it could be because the receiving node has recently crashed and lost state or because of some other system malfunction or attack. If the receiving node has an active IKE_SA to the IP address from whence the packet came, it MAY send a notification of the wayward packet over that IKE_SA in an INFORMATIONAL exchange. If it does not have such an IKE_SA, it MAY send an Informational message without cryptographic protection to the source IP address. Such a message is not part of an informational exchange, and the receiving node MUST NOT respond to it. Doing so could cause a message loop.

{{ Clarif-7.7 }} There are two cases when such a one-way notification is sent: INVALID_IKE_SPI and INVALID_SPI. These notifications are sent outside of an IKE_SA. Note that such notifications are explicitly not Informational exchanges; these are one-way messages

that must not be responded to. In case of INVALID_IKE_SPI, the message sent is a response message, and thus it is sent to the IP address and port from whence it came with the same IKE SPIs and the Message ID copied. In case of INVALID_SPI, however, there are no IKE SPI values that would be meaningful to the recipient of such a notification. Using zero values or random values are both acceptable.

1.6. Requirements Terminology

Keywords "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT" and "MAY" that appear in this document are to be interpreted as described in [MUSTSHOULD].

The term "Expert Review" is to be interpreted as defined in [IANACONS].

1.7. Introduction to IKEv2.1

IKEv2.1 is very similar to IKEv2. Most of the differences between this document at [IKEV2] are clarifications, mostly based on [Clarif]. The changes listed in that document were discussed in the IPsec Working Group and, after the Working Group was disbanded, on the IPsec mailing list. That document contains detailed explanations of areas that were unclear in IKEv2, and is thus useful to implementers of IKEv2 and IKEv2.1.

In the body of this document, notes that are enclosed in double curly braces {{ such as this }} point out changes from IKEv2. Changes that come from [Clarif] are marked with the section from that document, such as "{{ Clarif-2.10 }}".

This document also make the figures and references a bit more regular than in IKEv2.

IKEv2 developers have noted that the SHOULD-level requirements are often unclear in that they don't say when it is OK to not obey the requirements. They also have noted that there are MUST-level requirements that are not related to interoperability. This document has more explanation of some of these SHOULD-level requirements, and some SHOULD-level and MUST-level requirements have been changed to better match the definitions in [MUSTSHOULD]. All non-capitalized uses of the words SHOULD and MUST now mean their normal English sense, not the interoperability sense of [MUSTSHOULD].

IKEv2 (and IKEv1) developers have noted that there is a great deal of material in the tables of codes in Section 3.10. This leads to implementers not having all the needed information in the main body

of the docment. A later version of this document may move much of the material from those tables into the associated parts of the main body of the document.

A later version of this document will probably have all the {{ }} comments removed from the body of the document and instead appear in an appendix.

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, X.509 certificates), and IKEv2 itself does not have a mechanism for fragmenting large messages. IP defines a mechanism for fragmentation of oversize UDP messages, but implementations vary in the maximum message size supported. Furthermore, use of IP fragmentation opens an implementation to denial of service 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 bytes long, and they SHOULD be able to send, receive, and process messages that are up to 3000 bytes long. {{ Demoted the SHOULD }} 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. Use of the "Hash and URL" formats rather than including certificates in exchanges where possible can avoid most problems. {{ Demoted the SHOULD }} Implementations and configuration need to keep in mind, however, that if the URL lookups are possible only after the IPsec SA is established, recursion issues could prevent this technique from working.

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 request/response pairs. 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 request/response pair 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 triggers 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 the sequence number in the response plus its window size (see Section 2.3).

IKE is a reliable protocol, in the sense that the initiator MUST retransmit a request until either it receives a corresponding reply OR it deems the IKE security association to have failed and it discards all state associated with the IKE_SA and any CHILD_SAs negotiated using that IKE_SA.

{{ Clarif-7.5 }} All packets sent on port 4500 MUST begin with the prefix of four zeros; otherwise, the receiver won't know how to handle them.

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.

The Message ID is a 32-bit quantity, which is zero for the first IKE request in each direction. {{ Clarif-3.11 }} When the IKE_AUTH exchange does not use EAP, the IKE_SA initial setup messages will always be numbered 0 and 1. When EAP is used, each pair of messages have their message numbers incremented; the first pair of AUTH messages will have an ID of 1, the second will be 2, and so on.

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 very different numbers of requests, the Message IDs in the two directions can be very different. There is no ambiguity in the messages, however, because the (I)nitiator and (R)esponse bits in the message header specify which of the four messages a particular one is.

{{ Clarif-2.2 }} The Message ID for IKE_SA_INIT messages is always zero, including for retries of the message due to responses such as COOKIE and INVALID_KE_PAYLOAD.

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. Rekeying an IKE_SA resets the sequence numbers.

{{ Clarif-2.3 }} 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). It is not sufficient to use the initiator's SPI and/or IP address to differentiate between the two 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.

2.3. Window Size for Overlapping Requests

In order to maximize IKE throughput, an IKE endpoint MAY issue multiple requests before getting a response to any of them if the other endpoint has indicated its ability to handle such requests. For simplicity, an IKE implementation MAY choose to process requests strictly in order and/or wait for a response to one request before issuing another. Certain rules must be followed to ensure interoperability between implementations using different strategies.

After an IKE_SA is set up, either end can initiate one or more requests. 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. {{ Changed the SHOULD to MUST }} An IKE endpoint MUST be prepared to accept and process multiple requests while it has a

request outstanding.

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.

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 should be capable of processing incoming requests out of order to maximize performance in the event of network failures or packet reordering.

{{ Clarif-7.3 }} The window size is assumed to be 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, it is not defined what the responder should do when it receives a SET_WINDOW_SIZE notification containing a smaller value than is currently in effect. 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.

Since IKE is designed to operate in spite of Denial of Service (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. {{ Demoted the SHOULD }} 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 message 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 message has been received from the other side recently, unprotected notifications MAY be ignored. Implementations MUST limit the rate at which they take actions based on unprotected messages.

Numbers of retries and lengths 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, retranmission 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. 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 MUST NOT continue sending on any SA if some failure prevents it from receiving on all of the associated SAs. If CHILD_SAs can fail independently from one another without the associated IKE_SA being able to send a delete message, then they MUST be negotiated by separate IKE_SAs.

There is a Denial of Service attack on the initiator of an IKE_SA that can be avoided if the initiator takes the proper care. Since the first two messages of an SA setup are not cryptographically protected, an attacker could respond to the initiator's message before the genuine responder and poison the connection setup attempt. To prevent this, the initiator MAY be willing to accept multiple responses to its first message, treat each as potentially legitimate, respond to it, and then discard all the invalid half-open connections when it receives a valid cryptographically protected response to any

one of its requests. Once a cryptographically valid response is received, all subsequent responses should be ignored whether or not they are cryptographically valid.

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. {{ Clarified the SHOULD }} 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

{{ The version number is changed in the following paragraph, and the discussion of handling of multiple versions is also changed throughout the section. }}

This document describes version 2.1 of IKE, meaning the major version number is 2 and the minor version number is 1. It is likely that some implementations will want to support version 1.0 and version 2.0 and version 2.1, and in the future, other versions.

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 notification message. 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.

In the discussion of clarifications to IKEv2, it became clear that there was a need for additional "MUST" and "SHOULD" requirements. Some of those changes are reflected in IKEv2.1. Thus, the node with the higher version number may also need to note that its correspondent may not be following the same required actions, which could affect interoperability.

{{ Promoted the SHOULD }} If an endpoint receives a message with a higher major version number, it MUST drop the message and MUST send an unauthenticated notification message containing the highest version number it supports. 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 the 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.

Note that IKEv1 does not follow these rules, because there is no way in v1 of noting that you are capable of speaking a higher version number. So an active attacker can trick two v2-capable nodes into speaking v1. {{ Demoted the SHOULD }} When a v2-capable node negotiates down to v1, it should note that fact in its logs.

Also for forward compatibility, all fields marked RESERVED MUST be set to zero by an implementation running version 2.0 or later, and their content MUST be ignored by an implementation running version 2.0 or later ("Be conservative in what you send and liberal in what you receive"). 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. If the critical flag is not set and the payload type is unsupported, that payload MUST be ignored.

{{ Demoted the SHOULD }}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 Section 2 and implementations MAY reject as invalid a message with those payloads in any other order.

2.6. Cookies

The term "cookies" originates with Karn and Simpson [PHOTURIS] in Photuris, an early proposal for key management with IPsec, and it has persisted. The Internet Security Association and Key Management Protocol (ISAKMP) [ISAKMP] fixed message header includes two eight-octet fields titled "cookies", and that syntax is used by both IKEv1 and IKEv2 though in IKEv2 they are referred to as the IKE SPI and there is a new separate field in a Notify payload holding the cookie. The initial two eight-octet fields in the header are used as a connection identifier at the beginning of IKE packets. {{ Promoted the SHOULD }} 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 and indicates that the remote SPI value is not yet known by the sender.

Unlike ESP and AH 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 I(nitiator) Flag bit 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.

An expected attack against IKE is state and CPU exhaustion, where the target is flooded with session initiation requests from forged IP addresses. This attack can be made less effective if an implementation of a responder uses minimal CPU and commits no state to an SA until it knows the initiator can receive packets at the address from which it claims to be sending them. To accomplish this, a responder SHOULD -- when it detects a large number of half-open IKE_SAs -- reject initial IKE messages unless they contain a Notify payload of type COOKIE. {{ Clarified the SHOULD }} If the responder wants to set up an SA, it SHOULD instead send an unprotected IKE message as a response and include COOKIE Notify payload with the cookie data to be returned. Initiators who receive such responses MUST retry the IKE_SA_INIT with a Notify payload of type COOKIE

containing the responder supplied cookie 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,
    SAi2, TSi, TSr}  -->
                                <--  HDR(A,B), SK {IDr, [CERT,]
                                         AUTH, SAr2, TSi, TSr}
```

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.

{{ Clarif-2.1 }} Because the responder's SPI identifies security-related state held by the responder, and in this case no state is created, the responder sends a zero value for the responder's SPI.

{{ Demoted the SHOULD }} An IKE implementation should 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 they use 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 into the hash ensures that an attacker who sees only message 2 can't successfully forge a message 3.

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. {{ Demoted the SHOULD NOT }} The responder should not accept cookies indefinitely after <secret> is changed, since that would defeat part of the denial of service protection. {{ Demoted the SHOULD }} The responder should change the value of <secret> frequently, especially if under attack.

{{ Clarif-2.1 }} In addition to cookies, there are several cases where the IKE_SA_INIT exchange does not result in the creation of an IKE_SA (such as INVALID_KE_PAYLOAD or NO_PROPOSAL_CHOSEN). In such a case, sending a zero value for the Responder's SPI is correct. If the responder sends a non-zero responder SPI, the initiator should not reject the response for only that reason.

{{ Clarif-2.5 }} 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.

2.6.1. Interaction of COOKIE and INVALID_KE_PAYLOAD

{{ This section added by Clarif-2.4 }}

There are two common reasons why the initiator may have to retry the IKE_SA_INIT exchange: the responder requests a cookie or wants a different Diffie-Hellman group than was included in the KEi payload. If the initiator receives a cookie from the responder, the initiator needs to decide whether or not tp include the cookie in only the next retry of the IKE_SA_INIT request, or in all subsequent retries as well.

If the initiator includes the cookie only in the next retry, one additional roundtrip may be needed in some cases. An additional roundtrip is needed also if the initiator includes the cookie in all retries, but the responder does not support this. For instance, if

the responder includes the SAi1 and KEi payloads in cookie calculation, it will reject the request by sending a new cookie.

If both peers support including the cookie in all retries, a slightly shorter exchange can happen. Implementations MUST support this shorter exchange, but MUST NOT assume other implementations also supports 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, and/or AH) 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 or more protocols (usually one). 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 identifier 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 or more protocols. If a proposal is accepted, the SA response MUST contain the same protocols in the same order as the proposal. The responder MUST accept a single proposal or reject them all and return an error. (Example: if a single proposal contains ESP and AH and that proposal is accepted, both ESP and AH MUST be accepted. If ESP and AH are included in separate proposals, the responder MUST accept only one of them).

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_3DES, ENCR_AES w/keysize 128, ENCR_AES w/keysize 256, AUTH_HMAC_MD5, and AUTH_HMAC_SHA, the accepted suite MUST contain one of the ENCR_ transforms and one of the AUTH_ transforms. Thus, six combinations are acceptable.

Since the initiator sends its Diffie-Hellman value in the IKE_SA_INIT, it must guess the Diffie-Hellman group that the responder will select from its list of supported groups. If the initiator guesses wrong, the responder will respond with a Notify

payload of type INVALID_KE_PAYLOAD indicating the selected group. In this case, the initiator MUST retry the IKE_SA_INIT with the corrected Diffie-Hellman group. The initiator MUST again propose its full set of acceptable cryptographic suites because the rejection message was unauthenticated and otherwise an active attacker could trick the endpoints into negotiating a weaker suite than a stronger one that they both prefer.

2.8. Rekeying

{{ Demoted the SHOULD }} IKE, ESP, and AH 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".

To allow for minimal IPsec implementations, the ability to rekey SAs without restarting the entire IKE_SA is optional. An implementation MAY refuse all CREATE_CHILD_SA requests within an IKE_SA. 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. {{ Demoted the SHOULD }} Implementations should 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. 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. Use the new IKE_SA for all control messages needed to maintain the CHILD_SAs created by the old IKE_SA, and delete the old IKE_SA. The Delete payload to delete itself MUST be the last request sent over an IKE_SA.

{{ Demoted the SHOULD }} 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.

A difference between IKEv1 and IKEv2 is that in IKEv1 SA lifetimes were negotiated. 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 bundle 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. {{ Demoted the SHOULD }} It should 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.

{{ Demoted the SHOULD }} The node that initiated the surviving rekeyed SA should delete the replaced SA after the new one is established.

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 MAY want to 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 new SA, or (2) the new SA rekeys an existing SA and it receives an IKE request to close the replaced SA. {{ Clarif-5.10 }} When rekeying an SA, the responder SHOULD continue 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. {{ Demoted the SHOULD }} If an initiator receives a message on an SA for which it has not received a response

to its CREATE_CHILD_SA request, it should interpret that as a likely packet loss and retransmit the CREATE_CHILD_SA request. An initiator MAY send a dummy message on a newly created SA if it has no messages queued in order to assure the responder that the initiator is ready to receive messages.

{{ Clarif-5.9 }} Throughout this document, "initiator" refers to the party who initiated the exchange being described, and "original initiator" refers to the party who initiated the whole IKE_SA. The "original initiator" always refers to the party who initiated the exchange which resulted in the current IKE_SA. In other words, if the the "original responder" starts rekeying the IKE_SA, that party becomes the "original initiator" of the new IKE_SA.

2.8.1. Simultaneous CHILD_SA rekeying

{{ The first two paragraphs were moved, and the rest was added, based on Clarif-5.12 }}

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 though 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. {{ Clarif-5.11 }} "Lowest" means an octet-by-octet, lexicographical 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 following is an explanation on the impact this has on implementations. Assume that hosts A and B have an existing IPsec 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 going on. 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 resp4: 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 these 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 NO_PROPOSAL_CHOSEN.

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

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

2.8.2. Rekeying the IKE_SA Versus Reauthentication

{{ Added this section from Clarif-5.2 }}

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 or EAP payloads 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 EAP authentication and/or 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 is ongoing work to add this functionality [REAUTH].

2.9. Traffic Selector Negotiation

{{ Clarif-7.2 }} When an RFC4301-compliant IPsec subsystem receives an IP packet and 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 (see [PFKEY] for an example protocol), though 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. 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. In support of the scenario described in Section 1.1.3, an initiator may request that the responder assign an IP address and tell the initiator what it is. {{ Clarif-6.1 }} That request is done using configuration payloads, not traffic selectors. An address in a TSi payload in a response does not mean that the responder has assigned that address to the initiator: it only means that if packets matching these traffic selectors are sent by the initiator, IPsec processing

can be performed as agreed for this SA.

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.

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 request the creation of a CHILD_SA pair, and wishes to tunnel all traffic from subnet 192.0.1.* 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 (192.0.1.0 - 192.0.1.255) 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. (Note: The IP address range 192.0.2.* has been reserved for use in examples in RFCs and similar documents. This document needed two such ranges, and so also used 192.0.1.*. This should not be confused with any actual address.)

The responder is allowed to narrow the choices by selecting a subset of the traffic, for instance by eliminating or narrowing the range of one or more members of the set of traffic selectors, provided the set does not become the NULL set.

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 generated its request in response to an incoming packet from 192.0.1.43 to 192.0.2.123, 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 status of SINGLE_PAIR_REQUIRED.

{{ Clarif-4.11 }} Few implementations will have policies that require separate SAs for each address pair. Because of this, if only some part (or parts) of the TSi/TSr proposed by the initiator is (are) acceptable to the responder, responders SHOULD narrow TSi/TSr to an acceptable subset rather than use SINGLE_PAIR_REQUIRED.

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 (192.0.1.43 - 192.0.1.43) and the source port and IP protocol from the packet and the second containing (192.0.1.0 - 192.0.1.255) with all ports and IP protocols. The initiator would similarly include two traffic selectors in TSr.

If the responder's policy does not allow it to accept the entire set of traffic selectors in the initiator's request, but does allow him 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, the responder might respond with TSi being (192.0.1.43 - 192.0.1.43) with all ports and IP protocols.

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 MAY be ranges rather than specific values, and the responder chooses a subset of the initiator's TSi and TSr that are acceptable. If more than one subset is acceptable but their union is not, the responder MUST accept some subset and MAY include a Notify payload of type ADDITIONAL_TS_POSSIBLE to indicate that the initiator might want to try again. 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. {{ Demoted the SHOULD }} Such misconfigurations should be recorded in error logs.

{{ Clarif-4.10 }} A concise summary of the narrowing process is:

- If the responder's policy does not allow any part of the traffic covered by TSi/TSr, it responds with TS_UNACCEPTABLE.

- If the responder's policy allows the entire set of traffic covered by TSi/TSr, no narrowing is necessary, and the responder can

return the same TSi/TSr values.

- o Otherwise, narrowing is needed. If the responder's policy allows all traffic covered by TSi[1]/TSr[1] (the first traffic selectors in TSi/TSr) but not entire TSi/TSr, the responder narrows to an acceptable subset of TSi/TSr that includes TSi[1]/TSr[1].

- o If the responder's policy does not allow all traffic covered by TSi[1]/TSr[1], but does allow some parts of TSi/TSr, it narrows to an acceptable subset of TSi/TSr.

In the last two cases, there may be several subsets that are acceptable (but their union is not); in this case, the responder arbitrarily chooses one of them, and includes ADDITIONAL_TS_POSSIBLE notification in the response.

2.9.1. Traffic Selectors Violating Own Policy

{{ Clarif-4.12 }}

When creating a new SA, the initiator should not propose traffic selectors that violate its own policy. If this rule is not followed, valid traffic may be dropped.

This is best illustrated by an example. Suppose that host A has a policy whose effect is that traffic to 192.0.1.66 is sent via host B encrypted using AES, and traffic to all other hosts in 192.0.1.0/24 is also sent via B, but must use 3DES. Suppose also that host B accepts any combination of AES and 3DES.

If host A now proposes an SA that uses 3DES, and includes TSr containing (192.0.1.0-192.0.1.0.255), this will be accepted by host B. Now, host B can also use this SA to send traffic from 192.0.1.66, but those packets will be dropped by A since it requires the use of AES for those traffic. Even if host A creates a new SA only for 192.0.1.66 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 ((192.0.1.0-192.0.1.65),(192.0.1.67-192.0.1.255)) 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'.

2.10. 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 pseudo-random bits from the Diffie-Hellman key. Nonces used in IKEv2 MUST be randomly chosen, MUST be at least 128 bits in size, and MUST be at least half the key size of the negotiated prf. ("prf" refers to "pseudo-random function", one of the cryptographic algorithms negotiated in the IKE exchange.) {{ Clarif-7.4 }} However, the initiator chooses the nonce before the outcome of the negotiation is known. Because of that, the nonce has to be long enough for all the PRFs being proposed. If the same random number source is used for both keys and nonces, care must be taken to ensure that the latter use does not compromise the former.

2.11. Address and Port Agility

IKE runs over UDP ports 500 and 4500, and implicitly sets up ESP and AH associations for the same IP addresses it runs over. 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 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. In particular, it MUST forget the secrets used in the Diffie-Hellman calculation and any state that may persist in the state of a pseudo-random number generator that could be used to

recompute the Diffie-Hellman secrets.

Since the computing of 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.

Decisions as to whether and when to reuse Diffie-Hellman exponentials is a private decision in the sense that it 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.

2.13. Generating Keying Material

In the context of the IKE_SA, four cryptographic algorithms are negotiated: an encryption algorithm, an integrity protection algorithm, a Diffie-Hellman group, and a pseudo-random function (prf). The pseudo-random function 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. For algorithms for which not all values are valid keys (such as DES or 3DES with key parity), the algorithm by which keys are derived from arbitrary values MUST be specified by the cryptographic transform. 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. When the prf function takes a variable length key, variable length data, and produces a fixed-length output (e.g., when using HMAC), the formulas in this document apply. When the key for the prf function has fixed length, the data provided as a key is truncated or padded with zeros as necessary unless exceptional processing is explained following the formula.

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 algorithm, we will use the prf iteratively. We will use the terminology prf+ to describe the function that outputs a pseudo-random stream based on the inputs to a prf as follows: (where | indicates concatenation)

```
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)
```

continuing as needed to compute all required keys. 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 160-bit HMAC key, and the prf function generates 160 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 string feeding the prf is a single octet. prf+ in this document is not defined beyond 255 times the size of the prf output.

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 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.

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 Diffie-Hellman exchange. g^ir is represented as a string of octets in 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. If the negotiated prf takes a fixed-length key and the lengths of Ni and Nr do not add up to that length, half the bits must come from Ni and half from Nr, taking the first bits of each.

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. Each algorithm takes a fixed number of bits of keying material, which is specified as part of the algorithm. For integrity algorithms based on a keyed hash, the key size is always equal to the length of the output of the underlying hash function.

2.15. Authentication of the IKE_SA

When not using extensible authentication (see Section 2.16), the peers are authenticated by having each sign (or MAC using a shared secret as the key) a block of data. For the responder, the octets to be signed start with the first octet of the first SPI in the header of the second message and end with the last octet of the last payload in the second message. Appended to this (for 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') where IDr' is the responder's ID payload excluding the fixed header. Note that neither the nonce Ni nor the value prf(SK_pr,IDr') are transmitted. Similarly, the initiator signs the first message, 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'). In the above calculation, IDi' and IDr' are the entire ID payloads excluding the fixed header. It is critical to the security of the exchange that each side sign the other side's nonce.

{{ Clarif-3.1 }}

The initiator's signed octets can be described as:

```
InitiatorSignedOctets = RealMessage1 | NonceRData | MACedIDForI
GenIKEHDR = [ four octets 0 if using port 4500 ] | RealIKEHDR
RealIKEHDR =  SPIi | SPIr |  . . . | Length
RealMessage1 = RealIKEHDR | RestOfMessage1
NonceRPayload = PayloadHeader | NonceRData
InitiatorIDPayload = PayloadHeader | RestOfIDPayload
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 if using port 4500 ] | RealIKEHDR
RealIKEHDR =  SPIi | SPIr |  . . . | Length
RealMessage2 = RealIKEHDR | RestOfMessage2
NonceIPayload = PayloadHeader | NonceIData
ResponderIDPayload = PayloadHeader | RestOfIDPayload
RestOfRespIDPayload = IDType | RESERVED | InitIDData
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 twice (the second time with a responder cookie and/or a different Diffie-Hellman group), it is the second 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 that the same key is used in both directions. Note that it is a common but typically insecure practice to have a shared key derived solely from a user-chosen password without incorporating another source of randomness.

This is typically insecure because user-chosen passwords 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.) {{ Demoted the SHOULD }} 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:

```
AUTH = prf(prf(Shared Secret,"Key Pad for IKEv2"), <msg octets>)
```

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.

{{ Clarif-3.8 }} If the negotiated prf takes a fixed-size key, the shared secret MUST be of that fixed size. This requirement means that it is difficult to use these PRFs with shared key authentication because it limits the shared secrets that can be used. Thus, PRFs that require a fixed-size key SHOULD NOT be used with shared key authentication. For example, PRF_AES128_CBC [PRFAES128CBC] originally used fixed key sizes; that RFC has been updated to handle variable key sizes in [PRFAES128CBC-bis]. Note that Section 2.13 also contains text that is related to PRFs with fixed key size. However, the text in that section applies only to the prf+ construction.

2.16. Extensible Authentication Protocol Methods

In addition to authentication using public key signatures and shared secrets, IKE supports authentication using methods defined in RFC 3748 [EAP]. Typically, these methods are asymmetric (designed for a user authenticating to a server), and they may not be mutual. For this reason, these protocols are typically used to authenticate the initiator to the responder and MUST be used in conjunction with a public key signature based authentication of the responder to the initiator. These methods are often associated with mechanisms referred to as "Legacy Authentication" mechanisms.

While this memo references [EAP] with the intent that new methods can be added in the future without updating this specification, some simpler variations are documented here and in Section 3.16. [EAP] defines an authentication protocol requiring a variable number of messages. Extensible Authentication is implemented in IKE as additional IKE_AUTH exchanges that MUST be completed in order to initialize the IKE_SA.

An initiator indicates a desire to use extensible authentication by leaving out the AUTH payload from message 3. By including an IDi payload but not an AUTH payload, the initiator has declared an

identity but has not proven it. If the responder is willing to use an extensible authentication method, it will place an Extensible Authentication Protocol (EAP) payload in message 4 and defer sending SAr2, TSi, and TSr until initiator authentication is complete in a subsequent IKE_AUTH exchange. In the case of a minimal extensible authentication, the initial SA establishment will appear as follows:

```
Initiator                          Responder
-----------------------------------------------------------------
HDR, SAi1, KEi, Ni  -->
                                <--  HDR, SAr1, KEr, Nr, [CERTREQ]
HDR, SK {IDi, [CERTREQ,]
    [IDr,] SAi2,
    TSi, TSr}  -->
                                <--  HDR, SK {IDr, [CERT,] AUTH,
                                         EAP }
HDR, SK {EAP}  -->
                                <--  HDR, SK {EAP (success)}
HDR, SK {AUTH}  -->
                                <--  HDR, SK {AUTH, SAr2, TSi, TSr }
```

{{ Clarif-3.11 }} As described in Section 2.2, when EAP is used, each pair of IKE_SA initial setup messages will have their message numbers incremented; the first pair of AUTH messages will have an ID of 1, the second will be 2, and so on.

For EAP methods that create a shared key as a side effect of authentication, that shared key MUST be used by both the initiator and responder to generate AUTH payloads in messages 7 and 8 using the syntax for shared secrets specified in Section 2.15. The shared key from EAP is the field from the EAP specification named MSK. The shared key generated during an IKE exchange MUST NOT be used for any other purpose.

EAP methods that do not establish a shared key SHOULD NOT be used, as they are subject to a number of man-in-the-middle attacks [EAPMITM] if these EAP methods are used in other protocols that do not use a server-authenticated tunnel. Please see the Security Considerations section for more details. If EAP methods that do not generate a shared key are used, the AUTH payloads in messages 7 and 8 MUST be generated using SK_pi and SK_pr, respectively.

{{ Demoted the SHOULD }} The initiator of an IKE_SA using EAP needs to be capable of extending the initial protocol exchange to at least ten IKE_AUTH exchanges in the event the responder sends notification messages and/or retries the authentication prompt. Once the protocol exchange defined by the chosen EAP authentication method has successfully terminated, the responder MUST send an EAP payload

containing the Success message. Similarly, if the authentication method has failed, the responder MUST send an EAP payload containing the Failure message. The responder MAY at any time terminate the IKE exchange by sending an EAP payload containing the Failure message.

Following such an extended exchange, the EAP AUTH payloads MUST be included in the two messages following the one containing the EAP Success message.

{{ Clarif-3.5 }} When the initiator authentication uses EAP, it is possible that the contents of the IDi payload is used only for AAA routing purposes and selecting which EAP method to use. This value may be different from the identity authenticated by the EAP method. It is important that policy lookups and access control decisions use the actual authenticated identity. Often the EAP server is implemented in a separate AAA server that communicates with the IKEv2 responder. In this case, the authenticated identity has to be sent from the AAA server to the IKEv2 responder.

{{ Clarif-3.9 }} The information in Section 2.17 about PRFs with fixed-size keys also applies to EAP authentication. For instance, a PRF that requires a 128-bit key cannot be used with EAP because specifies that the MSK is at least 512 bits long.

2.17. Generating Keying Material for CHILD_SAs

A single CHILD_SA is created by the IKE_AUTH exchange, and additional CHILD_SAs can optionally be created in CREATE_CHILD_SA exchanges. Keying material for them is generated as follows:

KEYMAT = prf+(SK_d, 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.

For CREATE_CHILD_SA exchanges including an optional Diffie-Hellman exchange, the keying material is defined as:

KEYMAT = prf+(SK_d, 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 in the high-order bits if necessary to make it the length of the modulus).

A single CHILD_SA negotiation may result in multiple security associations. ESP and AH SAs exist in pairs (one in each direction),

and four SAs could be created in a single CHILD_SA negotiation if a combination of ESP and AH is being negotiated.

Keying material MUST be taken from the expanded KEYMAT in the following order:

- All keys for SAs carrying data from the initiator to the responder are taken before SAs going in the reverse direction.
- If multiple IPsec protocols are negotiated, keying material is taken in the order in which the protocol headers will appear in the encapsulated packet.
- If a single protocol has both encryption and authentication keys, the encryption key is taken from the first octets of KEYMAT and the authentication key is taken from the next octets.

Each cryptographic algorithm takes a fixed number of bits of keying material specified as part of the algorithm.

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 Section 2.8). {{ Clarif-5.3 }} 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.

{{ Clarif-5.5 }} 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. Note that this may not work if the new IKE_SA's PRF has a fixed key size because the output of the PRF may not be of the correct size.

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.

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.

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 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:

{{ Clarif-6.3 }}

```
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 IKE exchange with a FAILED_CP_REQUIRED error.

2.20. Requesting the Peer's Version

An IKE peer wishing to inquire about the other peer's IKE software version information MAY use the method below. This is an example of a configuration request within an INFORMATIONAL exchange, after the IKE_SA and first CHILD_SA have been created.

An IKE implementation MAY decline to give out version information prior to authentication or even after authentication to prevent trolling in case some implementation is known to have some security weakness. In that case, it MUST either return an empty string or no CP payload if CP is not supported.

```
Initiator                          Responder
-------------------------------------------------------------------
HDR, SK{CP(CFG_REQUEST)}  -->
                                <--  HDR, SK{CP(CFG_REPLY)}

CP(CFG_REQUEST)=
  APPLICATION_VERSION("")

CP(CFG_REPLY) APPLICATION_VERSION("foobar v1.3beta, (c) Foo Bar
  Inc.")
```

2.21. Error Handling

There are many kinds of errors that can occur during IKE processing. If a request is received that is badly formatted or unacceptable for reasons of policy (e.g., no matching cryptographic algorithms), the response MUST contain a Notify payload indicating the error. If an error occurs outside the context of an IKE request (e.g., the node is getting ESP messages on a nonexistent SPI), the node SHOULD initiate an INFORMATIONAL exchange with a Notify payload describing the problem.

Errors that occur before a cryptographically protected IKE_SA is established must be handled very carefully. There is a trade-off between wanting to be helpful in diagnosing a problem and responding to it and wanting to avoid being a dupe in a denial of service attack based on forged messages.

If a node receives a message on UDP port 500 or 4500 outside the context of an IKE_SA known to it (and not a request to start one), it may be the result of a recent crash of the node. If the message is marked as a response, the node MAY audit the suspicious event but MUST NOT respond. If the message is marked as a request, the node MAY audit the suspicious event and MAY send a response. If a response is sent, the response MUST be sent to the IP address and port from whence it came with the same IKE SPIs and the Message ID copied. The response MUST NOT be cryptographically protected and MUST contain a Notify payload indicating INVALID_IKE_SPI.

A node receiving such an unprotected Notify payload MUST NOT respond and MUST NOT change the state of any existing SAs. The message might be a forgery or might be a response the genuine correspondent was tricked into sending. {{ Demoted two SHOULDs }} A node should treat such a message (and also a network message like ICMP destination unreachable) as a hint that there might be problems with SAs to that IP address and should initiate a liveness test for any such IKE_SA. An implementation SHOULD limit the frequency of such tests to avoid being tricked into participating in a denial of service attack.

A node receiving a suspicious message from an IP address with which it has an IKE_SA MAY send an IKE Notify payload in an IKE INFORMATIONAL exchange over that SA. {{ Demoted the SHOULD }} The recipient MUST NOT change the state of any SAs as a result but may wish to audit the event to aid in diagnosing malfunctions. A node MUST limit the rate at which it will send messages in response to unprotected messages.

2.22. IPComp

Use of IP compression [IPCOMP] can be negotiated as part of the setup of a CHILD_SA. While IP compression involves an extra header in each packet and a compression parameter index (CPI), the virtual "compression association" has no life outside the ESP or AH SA that contains it. Compression associations disappear when the corresponding ESP or AH SA goes away. It is not explicitly mentioned in any DELETE payload.

Negotiation of IP compression is separate from the negotiation of cryptographic parameters associated with a CHILD_SA. A node requesting a CHILD_SA MAY advertise its support for one or more compression algorithms through one or more Notify payloads of type IPCOMP_SUPPORTED. The response MAY indicate acceptance of a single compression algorithm with a Notify payload of type IPCOMP_SUPPORTED. These payloads MUST NOT occur in messages that do not contain SA payloads.

Although there has been discussion of allowing multiple compression algorithms to be accepted and to have different compression algorithms available for the two directions of a CHILD_SA, implementations of this specification MUST NOT accept an IPComp algorithm that was not proposed, MUST NOT accept more than one, and MUST NOT compress using an algorithm other than one proposed and accepted in the setup of the CHILD_SA.

A side effect of separating the negotiation of IPComp from cryptographic parameters is that it is not possible to propose multiple cryptographic suites and propose IP compression with some of them but not others.

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 specify some unintuitive processing rules in order 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. If the connection runs in transport mode, changing the IP addresses on packets will cause the checksums to fail and the NAT cannot correct the checksums because they are cryptographically protected. Even in tunnel mode, there are routing problems because transparently translating the addresses of AH and ESP packets requires special logic in the NAT and that logic is heuristic and unreliable in nature. For that reason, IKEv2 can negotiate 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 IPsec traffic over UDP but not ESP/AH 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 from and to UDP port 500, 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. When working through a NAT, it is generally better to pass IKE packets over port 4500 because some older NATs handle IKE traffic on port 500 cleverly in an attempt to transparently establish IPsec connections between endpoints that don't handle NAT traversal themselves. Such NATs may interfere with the straightforward NAT traversal envisioned by this document. {{ Clarif-7.6 }} An IPsec endpoint that discovers a NAT

between it and its correspondent MUST send all subsequent traffic from port 4500, which NATs should not treat specially (as they might with port 500).

The specific requirements for supporting NAT traversal [NATREQ] are listed below. Support for NAT traversal is optional. In this section only, requirements listed as MUST apply only to implementations supporting NAT traversal.

- IKE MUST listen on port 4500 as well as port 500. IKE MUST respond to the IP address and port from which packets arrived.

- Both IKE initiator and responder MUST include in their IKE_SA_INIT packets Notify payloads of type NAT_DETECTION_SOURCE_IP and NAT_DETECTION_DESTINATION_IP. Those payloads can be used to detect if there is NAT between the hosts, and which end is behind the NAT. The location of the payloads in the IKE_SA_INIT packets are just after the Ni and Nr payloads (before the optional CERTREQ payload).

- If none of the NAT_DETECTION_SOURCE_IP payload(s) received matches the hash of the source IP and port found from the IP header of the packet containing the payload, it means that the other end is behind NAT (i.e., someone along the route changed the source address of the original packet to match the address of the NAT box). In this case, this end should allow dynamic update of the other ends IP address, as described later.

- If the NAT_DETECTION_DESTINATION_IP payload received does not match the hash of the destination IP and port found from the IP header of the packet containing the payload, it means that this end is behind a NAT. In this case, this end SHOULD start sending keepalive packets as explained in [UDPENCAPS].

- The IKE initiator MUST check these payloads if present and if they do not match the addresses in the outer packet MUST tunnel all future IKE and ESP packets associated with this IKE_SA over UDP port 4500.

- To tunnel IKE packets over UDP port 4500, the IKE header has four octets of zero 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 bytes of the ESP header contain the SPI, and the SPI cannot validly be zero, it is always possible to distinguish ESP and IKE messages.

- The original source and destination IP address required for the transport mode TCP and UDP packet checksum fixup (see [UDPENCAPS]) are obtained from the Traffic Selectors associated with the exchange. In the case of NAT traversal, the Traffic Selectors MUST contain exactly one IP address, which is then used as the original IP address.

- 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). To recover in these cases, hosts that are not behind a NAT SHOULD send all packets (including retransmission packets) to the IP address and port from the last valid authenticated packet from the other end (i.e., dynamically update the address). {{ Promoted the SHOULD }} A host behind a NAT MUST NOT do this because it opens a DoS attack possibility. Any authenticated IKE packet or any authenticated UDP-encapsulated ESP packet can be used to detect that the IP address or the port has changed.

Note that similar but probably not identical actions will likely be needed to make IKE work with Mobile IP, but such processing is not addressed by this document.

2.24. Explicit Congestion Notification (ECN)

When IPsec tunnels behave as originally specified in [IPSECARCH-OLD], ECN usage is not appropriate for the outer IP headers because tunnel decapsulation processing discards ECN congestion indications to the detriment of the network. ECN support for IPsec tunnels for IKEv1-based IPsec requires multiple operating modes and negotiation (see [ECN]). IKEv2 simplifies this situation by requiring that ECN be usable in the outer IP headers of all tunnel-mode IPsec 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.

3. Header and Payload Formats

3.1. The IKE Header

IKE messages use UDP ports 500 and/or 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 500, IKE messages begin immediately following the UDP header. When sent on UDP port 4500, IKE messages have prepended four octets of zero. These four octets of zero 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 memo. Following the header are one or more IKE payloads each identified by a "Next Payload" field in the preceding payload. Payloads are processed in the order in which they appear in an IKE message by invoking the appropriate processing routine according to 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 Recipient 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.

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

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

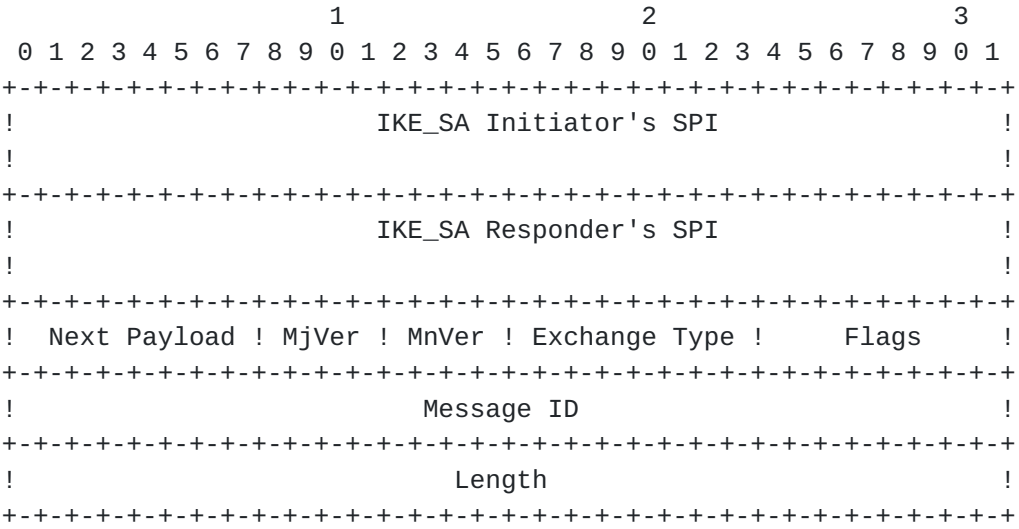

Figure 4: IKE Header Format

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.

- 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). {{ The phrase "and MUST NOT be zero in any other message" was removed; Clarif-2.1 }}

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

- 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 previous versions of IKE and ISAKMP MUST set the Major Version to 1. Implementations based on this version of IKE MUST reject or ignore messages containing a version number greater than 2.

- 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.

- Exchange Type (1 octet) - Indicates the type of exchange being used. This constrains the payloads sent in each message and orderings of messages in an exchange.

Exchange Type	Value
RESERVED	0-33
IKE_SA_INIT	34
IKE_AUTH	35
CREATE_CHILD_SA	36
INFORMATIONAL	37
RESERVED TO IANA	38-239
Reserved for private use	240-255

- Flags (1 octet) - Indicates specific options that are set for the message. Presence of options are indicated by the appropriate bit in the flags field being set. The bits are defined LSB first, so bit 0 would be the least significant bit of the Flags octet. In the description below, a bit being 'set' means its value is '1', while 'cleared' means its value is '0'.

  * X(reserved) (bits 0-2) - These bits MUST be cleared when sending and MUST be ignored on receipt.

  * I(nitiator) (bit 3 of Flags) - 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.

* V(ersion) (bit 4 of Flags) - 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.

* R(esponse) (bit 5 of Flags) - 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.

* X(reserved) (bits 6-7 of Flags) - These bits MUST be cleared when sending and MUST be ignored on receipt.

o Message ID (4 octets) - 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 Section 2.1 and Section 2.2.

o Length (4 octets) - Length of total message (header + payloads) in octets.

3.2. Generic Payload Header

Each IKE payload defined in Section 3.3 through Section 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.

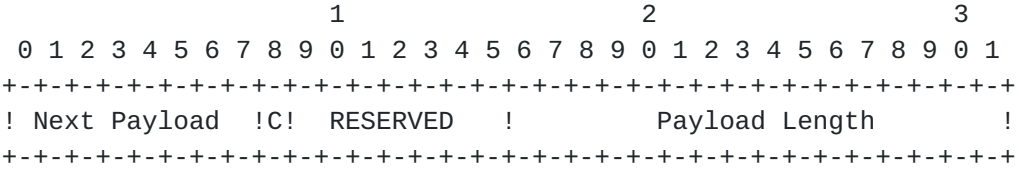

Figure 5: Generic Payload Header

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 it 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). The payload type values are:

Next Payload Type	Notation	Value
No Next Payload		0
RESERVED		1-32
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	E	46
Configuration	CP	47
Extensible Authentication	EAP	48
RESERVED TO IANA		49-127
PRIVATE USE		128-255

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

- 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.

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

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

3.3. Security Association Payload

The Security Association Payload, denoted SA in this memo, 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 may contain multiple IPsec protocols (where a protocol is IKE, ESP, or AH), each protocol MAY contain multiple transforms, and 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 syntax of Security Associations, Proposals, Transforms, and Attributes is based on ISAKMP; however the semantics are somewhat different. The reason for the complexity and the hierarchy is to allow for multiple possible combinations of algorithms to be encoded in a single SA. Sometimes there is a choice of multiple algorithms, whereas other times there is a combination of algorithms. For example, an initiator might want to propose using (AH w/MD5 and ESP w/3DES) OR (ESP w/MD5 and 3DES).

One of the reasons the semantics of the SA payload has changed from ISAKMP and IKEv1 is to make the encodings more compact in common cases.

The Proposal structure contains within it a Proposal # and an IPsec protocol ID. Each structure MUST have the same Proposal # as the previous one or be one (1) greater. The first Proposal MUST have a Proposal # of one (1). If two successive structures have the same Proposal number, it means that the proposal consists of the first

structure AND the second. So a proposal of AH AND ESP would have two proposal structures, one for AH and one for ESP and both would have Proposal #1. A proposal of AH OR ESP would have two proposal structures, one for AH with Proposal #1 and one for ESP with Proposal #2.

Each Proposal/Protocol structure is followed by one or more transform structures. The number of different transforms is generally determined by the Protocol. AH generally has a single transform: an integrity check algorithm. ESP generally has two: an encryption algorithm and an integrity check algorithm. IKE generally has four transforms: a Diffie-Hellman group, an integrity check algorithm, a prf algorithm, and an encryption algorithm. If an algorithm that combines encryption and integrity protection is proposed, it MUST be proposed as an encryption algorithm and an integrity protection algorithm MUST NOT be proposed. 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. For example, to propose ESP with (3DES or IDEA) and (HMAC_MD5 or HMAC_SHA), the ESP proposal would contain two Transform Type 1 candidates (one for 3DES and one for IDEA) and two Transform Type 2 candidates (one for HMAC_MD5 and one for HMAC_SHA). This effectively proposes four combinations of algorithms. If the initiator wanted to propose only a subset of those, for example (3DES and HMAC_MD5) or (IDEA and HMAC_SHA), there is no way to encode that as multiple transforms within a single Proposal. Instead, the initiator would have to construct two different Proposals, each with two transforms.

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), and implementation MUST include multiple Transforms with the same Transform Type each with a single Attribute.

Note that the semantics of Transforms and Attributes are quite different from those in IKEv1. In IKEv1, a single Transform carried multiple algorithms for a protocol with one carried in the Transform and the others carried in the Attributes.

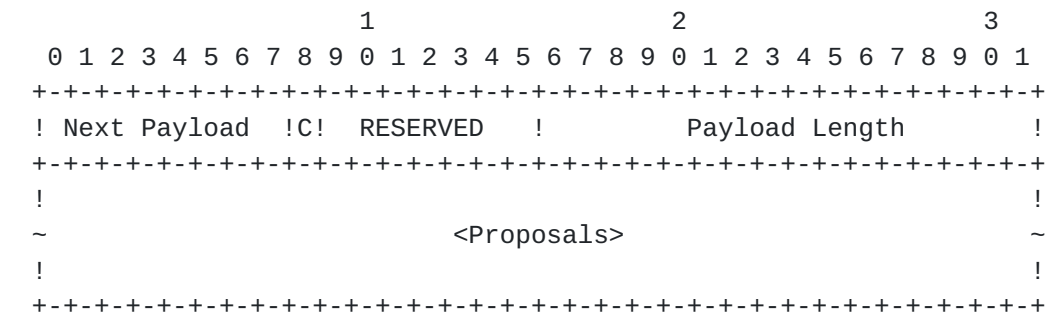

Figure 6: Security Association Payload

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

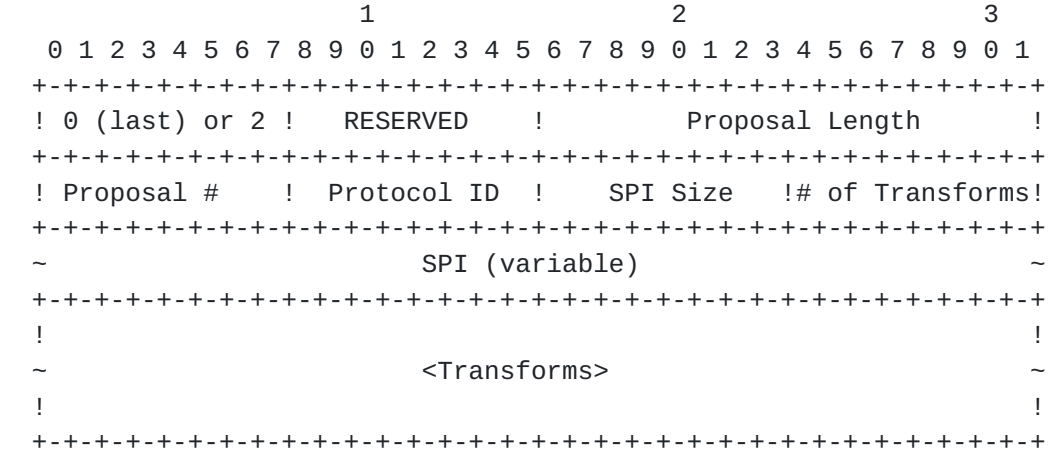

Figure 7: Proposal Substructure

o 0 (last) or 2 (more) (1 octet) - Specifies whether this is the last Proposal Substructure in the SA. This syntax is inherited from ISAKMP, but is unnecessary because the last Proposal could be identified from the length of the SA. The value (2) corresponds to a Payload Type of Proposal in IKEv1, and the first four octets of the Proposal structure are designed to look somewhat like the header of a Payload.

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

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

- Proposal # (1 octet) - When a proposal is made, the first proposal in an SA payload MUST be #1, and subsequent proposals MUST either be the same as the previous proposal (indicating an AND of the two proposals) or one more than the previous proposal (indicating an OR of the two proposals). When a proposal is accepted, all of the proposal numbers in the SA payload MUST be the same and MUST match the number on the proposal sent that was accepted.

- Protocol ID (1 octet) - Specifies the IPsec protocol identifier for the current negotiation. The defined values are:

Protocol	Protocol ID
RESERVED	0
IKE	1
AH	2
ESP	3
RESERVED TO IANA	4-200
PRIVATE USE	201-255

- 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 and AH).

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

- 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.

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

3.3.2. Transform Substructure

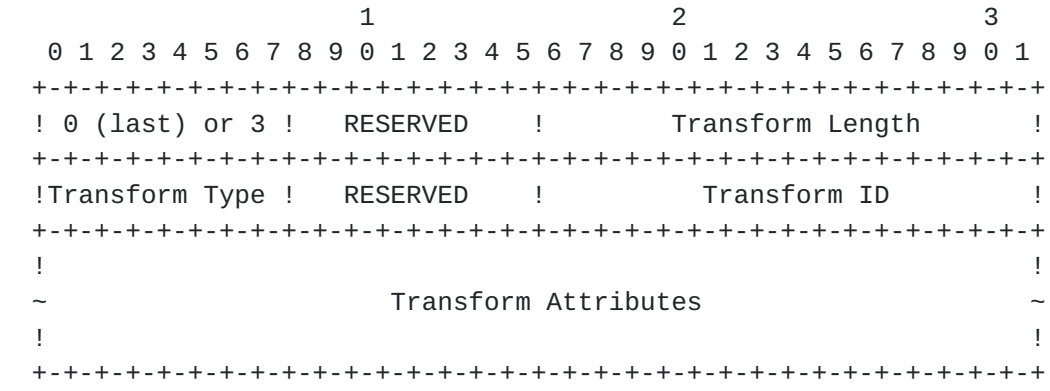

Figure 8: Transform Substructure

- 0 (last) or 3 (more) (1 octet) - Specifies whether this is the last Transform Substructure in the Proposal. This syntax is inherited from ISAKMP, but is unnecessary because the last Proposal could be identified from the length of the SA. The value (3) corresponds to a Payload Type of Transform in IKEv1, and the first four octets of the Transform structure are designed to look somewhat like the header of a Payload.

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

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

- 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.

- Transform ID (2 octets) - The specific instance of the transform type being proposed.

The tranform type values are:

Description	Trans. Type	Used In
RESERVED	0	
Encryption Algorithm (ENCR)	1	IKE and ESP
Pseudo-random Function (PRF)	2	IKE
Integrity Algorithm (INTEG)	3	IKE, AH, optional in ESP
Diffie-Hellman Group (D-H)	4	IKE, optional in AH & ESP
Extended Sequence Numbers (ESN)	5	AH and ESP
RESERVED TO IANA	6-240	
PRIVATE USE	241-255	

For Transform Type 1 (Encryption Algorithm), defined Transform IDs are:

Name	Number	Defined In
RESERVED	0	
ENCR_DES_IV64	1	(RFC1827)
ENCR_DES	2	(RFC2405), [DES]
ENCR_3DES	3	(RFC2451)
ENCR_RC5	4	(RFC2451)
ENCR_IDEA	5	(RFC2451), [IDEA]
ENCR_CAST	6	(RFC2451)
ENCR_BLOWFISH	7	(RFC2451)
ENCR_3IDEA	8	(RFC2451)
ENCR_DES_IV32	9	
RESERVED	10	
ENCR_NULL	11	(RFC2410)
ENCR_AES_CBC	12	(RFC3602)
ENCR_AES_CTR	13	(RFC3664)
RESERVED TO IANA	14-1023	
PRIVATE USE	1024-65535	

For Transform Type 2 (Pseudo-random Function), defined Transform IDs are:

Name	Number	Defined In
RESERVED	0	
PRF_HMAC_MD5	1	(RFC2104), [MD5]
PRF_HMAC_SHA1	2	(RFC2104), [SHA]
PRF_HMAC_TIGER	3	(RFC2104)
PRF_AES128_XCBC	4	(RFC3664)
RESERVED TO IANA	5-1023	
PRIVATE USE	1024-65535	

For Transform Type 3 (Integrity Algorithm), defined Transform IDs

are:

Name	Number	Defined In
NONE	0	
AUTH_HMAC_MD5_96	1	(RFC2403)
AUTH_HMAC_SHA1_96	2	(RFC2404)
AUTH_DES_MAC	3	
AUTH_KPDK_MD5	4	(RFC1826)
AUTH_AES_XCBC_96	5	(RFC3566)
RESERVED TO IANA	6-1023	
PRIVATE USE	1024-65535	

For Transform Type 4 (Diffie-Hellman Group), defined Transform IDs are:

Name	Number
NONE	0
Defined in Appendix B	1 - 2
RESERVED	3 - 4
Defined in [ADDGROUP]	5
RESERVED TO IANA	6 - 13
Defined in [ADDGROUP]	14 - 18
RESERVED TO IANA	19 - 1023
PRIVATE USE	1024-65535

For Transform Type 5 (Extended Sequence Numbers), defined Transform IDs are:

Name	Number
No Extended Sequence Numbers	0
Extended Sequence Numbers	1
RESERVED	2 - 65535

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 and optional transform types. A compliant implementation MUST understand all mandatory and optional types for each protocol it supports (though it need not accept proposals with unacceptable suites). A proposal MAY omit the optional types if the only value for them it will accept is NONE.

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

3.3.4. Mandatory Transform IDs

The specification of suites that MUST and SHOULD be supported for interoperability has been removed from this document because they are likely to change more rapidly than this document evolves.

An important lesson learned from IKEv1 is that no system should only implement the mandatory algorithms and expect them to be the best choice for all customers. For example, at the time that this document was written, many IKEv1 implementers were starting to migrate to AES in Cipher Block Chaining (CBC) mode for Virtual Private Network (VPN) applications. Many IPsec systems based on IKEv2 will implement AES, additional Diffie-Hellman groups, and additional hash algorithms, and some IPsec customers already require these algorithms in addition to the ones listed above.

It is likely that IANA will add additional transforms in the future, and some users may want to use private suites, especially for IKE where implementations should be capable of supporting different parameters, up to certain size limits. In support of this goal, all implementations of IKEv2 SHOULD include a management facility that allows specification (by a user or system administrator) of Diffie-Hellman (DH) parameters (the generator, modulus, and exponent lengths and values) for new DH groups. Implementations SHOULD provide a management interface through which these parameters and the associated transform IDs may be entered (by a user or system administrator), to enable negotiating such groups.

All implementations of IKEv2 MUST include a management facility that enables a user or system administrator to specify the suites that are acceptable for use with IKE. Upon receipt of a payload with a set of transform IDs, the implementation MUST compare the transmitted transform IDs against those locally configured via the management controls, to verify that the proposed suite is acceptable based on local policy. The implementation MUST reject SA proposals that are not authorized by these IKE suite controls. Note that cryptographic suites that MUST be implemented need not 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. These attributes are type/value pairs and are defined below. For example, if an encryption algorithm has a variable-length key, the key length to be used may be specified as an attribute. 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.

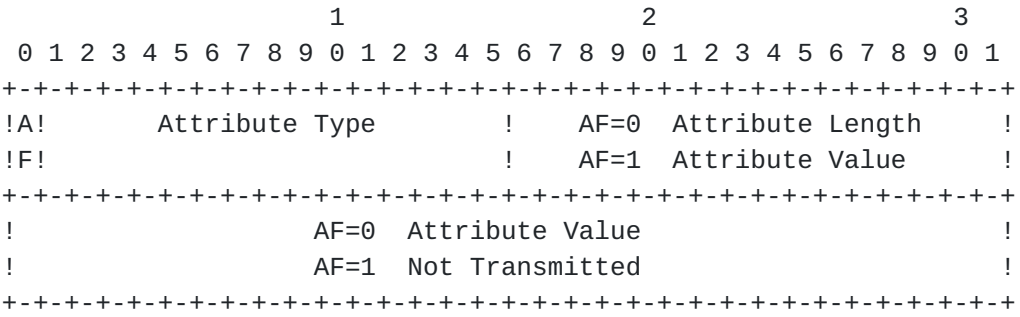

Figure 9: Data Attributes

- Attribute Type (2 octets) - Unique identifier for each type of attribute (see below). The most significant bit of this field is the Attribute Format bit (AF). It indicates whether the data attributes follow the Type/Length/Value (TLV) format or a shortened Type/Value (TV) format. If the AF bit is zero (0), then the Data Attributes are of the Type/Length/Value (TLV) form. If the AF bit is a one (1), then the Data Attributes are of the Type/Value form.

- Attribute Length (2 octets) - Length in octets of the Attribute Value. When the AF bit is a one (1), the Attribute Value is only 2 octets and the Attribute Length field is not present.

- 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.

- Key Length - When using an Encryption Algorithm that has a variable-length key, this attribute specifies the key length in bits (MUST use network byte order). This attribute MUST NOT be used when the specified Encryption Algorithm uses a fixed-length key.

Note that only a single attribute type (Key Length) is defined, and it is fixed length. The variable-length encoding specification is included only for future extensions. {{ Clarif-7.11 removed the sentence that listed, incorrectly, the algorithms defined in the document that accept attributes. }}

Attributes described as basic MUST NOT be encoded using the variable-length encoding. Variable-length attributes MUST NOT be encoded as basic even if their value can fit into two octets. NOTE: This is a change from IKEv1, where increased flexibility may have simplified the composer of messages but certainly complicated the parser.

Attribute Type	Value	Attribute Format
RESERVED	0-13	
Key Length (in bits)	14	TV
RESERVED	15-17	
RESERVED TO IANA	18-16383	
PRIVATE USE	16384-32767	

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

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 number and return all of the Proposal substructures with that Proposal number. If there are 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 that response MUST be rejected.

Negotiating Diffie-Hellman groups presents some special challenges. SA offers include proposed attributes and a Diffie-Hellman public number (KE) in the same message. If in the initial exchange the initiator offers to use one of several 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 guess turns out to be wrong, 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 SHOULD, however, continue to propose its full supported set of groups in order to prevent a man-in-the-middle downgrade attack.

Implementation Note:

Certain negotiable attributes can have ranges or could have multiple acceptable values. These include the key length of a variable key length symmetric cipher. To further interoperability and to support upgrading endpoints independently, implementers of this protocol SHOULD accept values that they deem to supply greater security. For instance, if a peer is configured to accept a variable-length cipher with a key length of X bits and is offered that cipher with a larger key length, the implementation SHOULD accept the offer if it supports use of the longer key.

Support of this capability allows an implementation to express a concept of "at least" a certain level of security-- "a key length of _at least_ X bits for cipher Y".

3.4. Key Exchange Payload

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

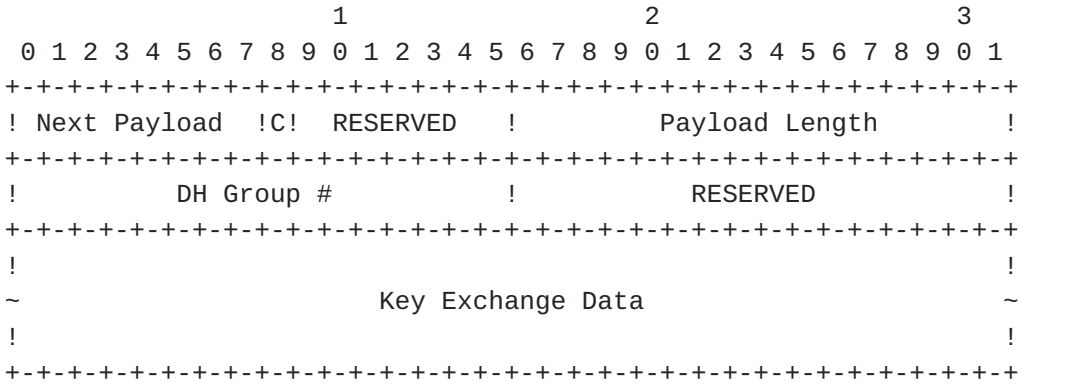

Figure 10: Key Exchange Payload Format

A key exchange payload is constructed by copying one's Diffie-Hellman public value into the "Key Exchange Data" portion of the payload. The length of the Diffie-Hellman public value MUST be equal to the length of the prime modulus over which the exponentiation was performed, prepending zero bits to the value if necessary.

The DH Group # identifies the Diffie-Hellman group in which the Key Exchange Data was computed (see Section 3.3.2). If the selected proposal uses a different Diffie-Hellman group, the message MUST be rejected with a Notify payload of type INVALID_KE_PAYLOAD.

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 memo, 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. {{ Clarif-7.1 }} 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 is used purely to fetch the policy and authentication data related to the other party.

NOTE: In IKEv1, two ID payloads were used in each direction to hold Traffic Selector (TS) information for data passing over the SA. In IKEv2, this information is carried in TS payloads (see Section 3.13).

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

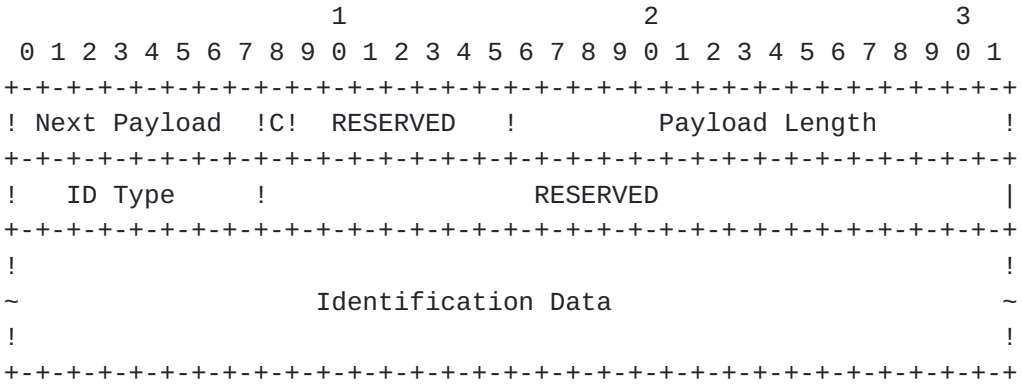

Figure 11: Identification Payload Format

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

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

- 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 values for the Identification Type field:

ID Type	Value
RESERVED	0
ID_IPV4_ADDR	1
	A single four (4) octet IPv4 address.
ID_FQDN	2
	A fully-qualified domain name string. An example of a ID_FQDN is, "example.com". The string MUST not contain any terminators (e.g., NULL, CR, etc.).
ID_RFC822_ADDR	3
	A fully-qualified RFC822 email address string, An example of a ID_RFC822_ADDR is, "jsmith@example.com". The string MUST not contain any terminators.
RESERVED TO IANA	4
ID_IPV6_ADDR	5
	A single sixteen (16) octet IPv6 address.
RESERVED TO IANA	6 - 8
ID_DER_ASN1_DN	9
	The binary Distinguished Encoding Rules (DER) encoding of an ASN.1 X.500 Distinguished Name [X.501].
ID_DER_ASN1_GN	10
	The binary DER encoding of an ASN.1 X.500 GeneralName [X.509].
ID_KEY_ID	11
	An opaque octet stream which may be used to pass vendor-specific information necessary to do certain proprietary types of identification.
RESERVED TO IANA	12-200
PRIVATE USE	201-255

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 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.

{{ Clarif-3.4 }} EAP [EAP] does not mandate the use of any particular type of identifier, but often EAP is used with Network Access Identifiers (NAIs) defined in [NAI]. Although NAIs look a bit like email addresses (e.g., "joe@example.com"), the syntax is not exactly the same as the syntax of email address in [MAILFORMAT]. For those NAIs that include the realm component, the ID_RFC822_ADDR identification type SHOULD be used. Responder implementations should not attempt to verify that the contents actually conform to the exact syntax given in [MAILFORMAT], but instead should accept any reasonable-looking NAI. For NAIs that do not include the realm component,the ID_KEY_ID identification type SHOULD be used.

3.6. Certificate Payload

The Certificate Payload, denoted CERT in this memo, 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 unless the peer has indicated an ability to retrieve this information from elsewhere using an HTTP_CERT_LOOKUP_SUPPORTED Notify payload. 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:

Figure 12: Certificate Payload Format

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

Certificate Encoding	Value
RESERVED	0
PKCS #7 wrapped X.509 certificate	1
PGP Certificate	2
DNS Signed Key	3
X.509 Certificate - Signature	4
Kerberos Token	6
Certificate Revocation List (CRL)	7
Authority Revocation List (ARL)	8
SPKI Certificate	9
X.509 Certificate - Attribute	10
Raw RSA Key	11
Hash and URL of X.509 certificate	12
Hash and URL of X.509 bundle	13
RESERVED to IANA	14 - 200
PRIVATE USE	201 - 255

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

- X.509 Certificate - Signature (4) contains a DER encoded X.509 certificate whose public key is used to validate the sender's AUTH payload.

- Certificate Revocation List (7) contains a DER encoded X.509 certificate revocation list.

- {{ Added "DER-encoded RSAPublicKey structure" from Clarif-3.7 }} Raw RSA Key (11) contains a PKCS #1 encoded RSA key, that is, a DER-encoded RSAPublicKey structure (see [RSA] and [PKCS1]).

- Hash and URL encodings (12-13) allow IKE messages to remain short by replacing long data structures with a 20 octet SHA-1 hash (see [SHA]) of the replaced value followed by a variable-length URL that resolves to the DER encoded data structure itself. This improves efficiency when the endpoints have certificate data cached and makes IKE less subject to denial of service attacks that become easier to mount when IKE messages are large enough to require IP fragmentation [DOSUDPPROT].

Use the following ASN.1 definition for an X.509 bundle:

```
CertBundle
  { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0)
    id-mod-cert-bundle(34) }

DEFINITIONS EXPLICIT TAGS ::=
BEGIN

IMPORTS
  Certificate, CertificateList
  FROM PKIX1Explicit88
     { iso(1) identified-organization(3) dod(6)
       internet(1) security(5) mechanisms(5) pkix(7)
       id-mod(0) id-pkix1-explicit(18) } ;

CertificateOrCRL ::= CHOICE {
  cert [0] Certificate,
  crl  [1] CertificateList }

CertificateBundle ::= SEQUENCE OF CertificateOrCRL

END
```

Implementations MUST be capable of being configured to send and accept up to four X.509 certificates in support of authentication, and also MUST be capable of being configured to send and accept the first two Hash and URL formats (with HTTP URLs). Implementations SHOULD be capable of being configured to send and accept Raw RSA keys. If multiple certificates are sent, the first certificate MUST contain the public key used to sign the AUTH payload. The other certificates may be sent in any order.

{{ Clarif-3.7 }} Because the contents and use of some of the certificate types are not defined, they SHOULD NOT be used. In specific, implementations SHOULD NOT use the following types unless they are later defined in a standards-track document:

PKCS #7 wrapped X.509 certificate	1
PGP Certificate	2
DNS Signed Key	3
Kerberos Token	6
SPKI Certificate	9

3.7. Certificate Request Payload

The Certificate Request Payload, denoted CERTREQ in this memo, 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. If multiple CAs are trusted and the cert encoding does not allow a list, then multiple Certificate Request payloads SHOULD be transmitted.

The Certificate Request Payload is defined as follows:

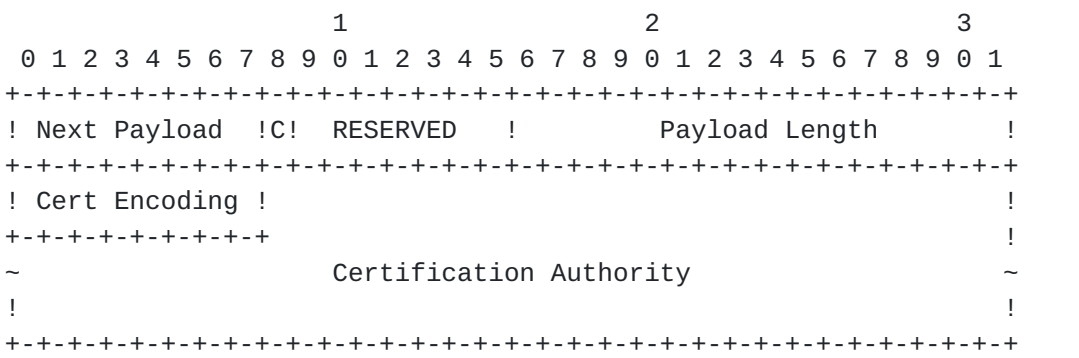

Figure 13: Certificate Request Payload Format

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

- 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-1 hashes of the public keys of trusted Certification Authorities (CAs). Each is encoded as the SHA-1 hash of the Subject Public Key Info element (see section 4.1.2.7 of [PKIX]) from each Trust Anchor certificate. The twenty-octet hashes are concatenated and included with no other formatting.

{{ Clarif-3.7 }} The contents of the "Certification Authority" field are defined only for X.509 certificates, which are types 4, 10, 12,

and 13. Other values SHOULD NOT be used until standards-track specifications that specify their use are published.

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:

- is configured to use certificate authentication,
- is allowed to send a CERT payload,
- has matching CA trust policy governing the current negotiation, and
- 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 memo, 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:

Figure 14: Authentication Payload Format

o Auth Method (1 octet) - Specifies the method of authentication used. Values defined are:

  * RSA Digital Signature (1) - Computed as specified in Section 2.15 using an RSA private key over a PKCS#1 padded hash (see [RSA] and [PKCS1]). {{ Clarif-3.2 }} To promote interoperability, implementations that support this type SHOULD support signatures that use SHA-1 as the hash function and SHOULD use SHA-1 as the default hash function when generating signatures. {{ Clarif-3.3 }} A newer version of PKCS#1 (v2.1) defines two different encoding methods (ways of "padding the hash") for signatures. However, IKEv2 and this document point specifically to the PKCS#1 v2.0 which has only one encoding method for signatures (EMSA-PKCS1- v1_5).

  * 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 function

  * DSS Digital Signature (3) - Computed as specified in Section 2.15 using a DSS private key (see [DSS]) over a SHA-1 hash.

  * The values 0 and 4-200 are reserved to IANA. The values 201-255 are available for private use.

o 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 Ni and Nr in this memo 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:

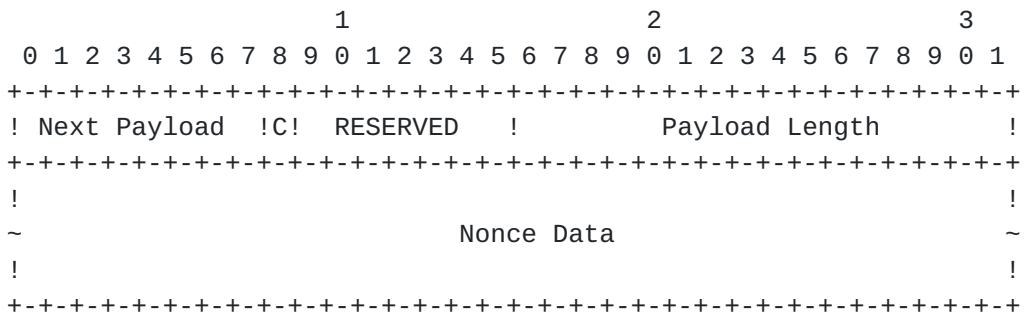

Figure 15: Nonce Payload Format

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 a Nonce 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:

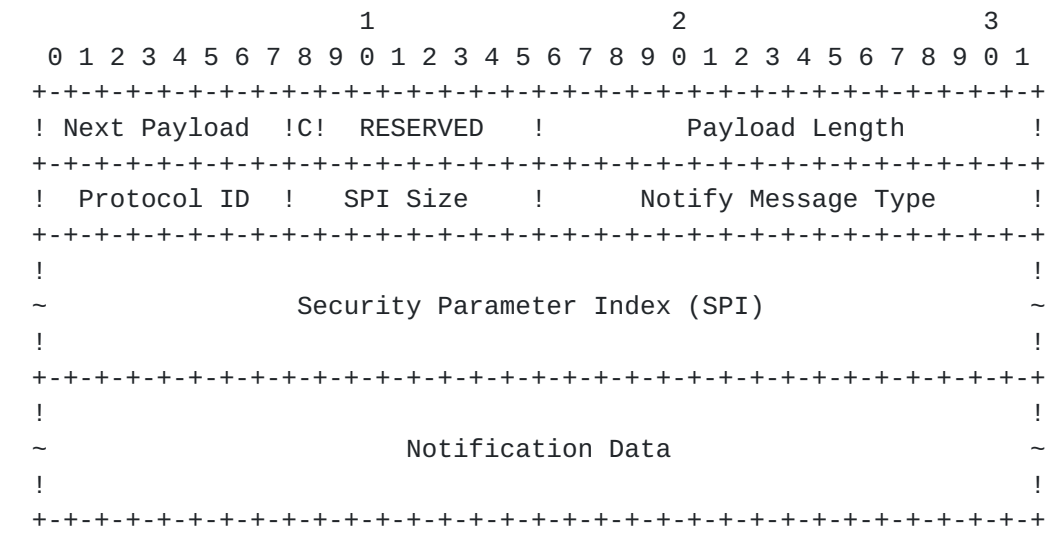

Figure 16: Notify Payload Format

- Protocol ID (1 octet) - If this notification concerns an existing SA, this field indicates the type of that SA. For IKE_SA notifications, this field MUST be one (1). For notifications concerning IPsec SAs this field MUST contain either (2) to indicate AH or (3) to indicate ESP. {{ Clarif-7.8 }} For notifications that do not relate to an existing SA, this field MUST be sent as zero and MUST be ignored on receipt; this is only true for the INVALID_SELECTORS and REKEY_SA notifications. . All other values for this field are reserved to IANA for future assignment.

- 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.

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

- SPI (variable length) - Security Parameter Index.

- Notification Data (variable length) - Informational 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. The number of different error statuses was greatly reduced from IKEv1 both for simplification and to avoid giving configuration information to probers.

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. {{ Demoted the SHOULD }} 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.

NOTIFY messages: error types	Value
RESERVED	0
UNSUPPORTED_CRITICAL_PAYLOAD	1
INVALID_IKE_SPI	4
INVALID_MAJOR_VERSION	5
INVALID_SYNTAX	7

Sent if the payload has the "critical" bit set and the payload type is not recognized. Notification Data contains the one-octet payload type.

Indicates an IKE message was received with an unrecognized destination SPI. This usually indicates that the recipient has rebooted and forgotten the existence of an IKE_SA.

Indicates the recipient cannot handle the version of IKE specified in the header. The closest version number that the recipient can support will be in the reply header.

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 denial of service attack using forged messages, this status may 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. {{ Demoted the SHOULD }} To aid debugging, more detailed error

information should be written to a console or log.

INVALID_MESSAGE_ID 9
Sent when an IKE message ID outside the supported window is received. This Notify MUST NOT be sent in a response; the invalid request MUST NOT be acknowledged. Instead, inform the other side by initiating an INFORMATIONAL exchange with Notification data containing the four octet invalid message ID. Sending this notification is optional, and notifications of this type MUST be rate limited.

INVALID_SPI 11
MAY be sent in an IKE INFORMATIONAL exchange when a node receives an ESP or AH packet with an invalid SPI. The Notification Data contains the SPI of the invalid packet. This usually indicates a node has rebooted and forgotten an SA. If this Informational Message is sent outside the context of an IKE_SA, it should only be used by the recipient as a "hint" that something might be wrong (because it could easily be forged).

NO_PROPOSAL_CHOSEN 14
None of the proposed crypto suites was acceptable.

INVALID_KE_PAYLOAD 17
The D-H Group # field in the KE payload is not the group # selected by the responder for this exchange. There are two octets of data associated with this notification: the accepted D-H Group # in big endian order.

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

SINGLE_PAIR_REQUIRED 34
This 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.

NO_ADDITIONAL_SAS 35
This error indicates that a CREATE_CHILD_SA request is unacceptable because the responder is unwilling to accept any more CHILD_SAs on this IKE_SA. Some minimal implementations may only accept a single CHILD_SA setup in the context of an initial IKE exchange and reject any subsequent attempts to add more.

INTERNAL_ADDRESS_FAILURE 36

Indicates an error assigning an internal address (i.e., INTERNAL_IP4_ADDRESS or INTERNAL_IP6_ADDRESS) during the processing of a Configuration Payload by a responder. If this error is generated within an IKE_AUTH exchange, no CHILD_SA will be created.

FAILED_CP_REQUIRED 37
Sent by responder in the case where CP(CFG_REQUEST) was expected but not received, and so is a conflict with locally configured policy. There is no associated data.

TS_UNACCEPTABLE 38
Indicates that none of the addresses/protocols/ports in the supplied traffic selectors is acceptable.

INVALID_SELECTORS 39
MAY be sent in an IKE INFORMATIONAL exchange when a node receives an ESP or AH 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 IPsec SA.

RESERVED TO IANA 40-8191

PRIVATE USE 8192-16383

NOTIFY messages: status types	Value

INITIAL_CONTACT 16384
This 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. This notification MUST NOT be sent by an entity that may be replicated (e.g., a roaming user's credentials where the user is allowed to connect to the corporate firewall from two remote systems at the same time). {{ Clarif-7.9 }} The INITIAL_CONTACT notification, if sent, MUST be in the first IKE_AUTH request, not as a separate exchange afterwards.

SET_WINDOW_SIZE 16385
This 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. Window size is always one until the initial exchanges complete.

ADDITIONAL_TS_POSSIBLE 16386

This notification asserts that the sending endpoint narrowed the proposed traffic selectors but that other traffic selectors would also have been acceptable, though only in a separate SA (see section 2.9). There is no data associated with this Notify type. It may be sent only as an additional payload in a message including accepted TSs.

IPCOMP_SUPPORTED 16387

This notification may be included only in a message containing an SA payload negotiating a CHILD_SA and indicates a willingness by its sender to use IPComp on this SA. The data associated with this notification includes a two-octet IPComp CPI followed by a one-octet transform ID optionally followed by attributes whose length and format are defined by that transform ID. A message proposing an SA may contain multiple IPCOMP_SUPPORTED notifications to indicate multiple supported algorithms. A message accepting an SA may contain at most one.

The transform IDs currently defined are:

Name	Number	Defined In
RESERVED	0	
IPCOMP_OUI	1	
IPCOMP_DEFLATE	2	RFC 2394
IPCOMP_LZS	3	RFC 2395
IPCOMP_LZJH	4	RFC 3051
RESERVED TO IANA	5-240	
PRIVATE USE	241-255	

NAT_DETECTION_SOURCE_IP 16388

This notification is used by its recipient to determine whether the source is behind a NAT box. The data associated with this notification is a SHA-1 digest of the SPIs (in the order they appear in the header), IP address, and port on which this packet was sent. There MAY be multiple Notify payloads of this type in a message if the sender does not know which of several network attachments will be used to send the packet. The recipient of this notification MAY compare the supplied value to a SHA-1 hash of the SPIs, source IP address, and port, and if they don't match it SHOULD enable NAT traversal (see section 2.23). Alternately,

it MAY reject the connection attempt if NAT traversal is not supported.

NAT_DETECTION_DESTINATION_IP 16389

This notification is used by its recipient to determine whether it is behind a NAT box. The data associated with this notification is a SHA-1 digest of the SPIs (in the order they appear in the header), IP address, and port to which this packet was sent. The recipient of this notification MAY compare the supplied value to a hash of the SPIs, destination IP address, and port, and if they don't match it SHOULD invoke NAT traversal (see section 2.23). If they don't match, it means that this end is behind a NAT and this end SHOULD start sending keepalive packets as defined in [UDPENCAPS]. Alternately, it MAY reject the connection attempt if NAT traversal is not supported.

COOKIE 16390

This notification MAY be included in an IKE_SA_INIT response. It indicates that the request should be retried with a copy of this notification as the first payload. This notification MUST be included in an IKE_SA_INIT request retry if a COOKIE notification was included in the initial response. The data associated with this notification MUST be between 1 and 64 octets in length (inclusive).

USE_TRANSPORT_MODE 16391

This notification MAY be included in a request message that also includes an SA payload requesting a CHILD_SA. It requests that the CHILD_SA use transport mode rather than tunnel mode for the SA created. If the request is accepted, the response MUST also include a notification of type USE_TRANSPORT_MODE. If the responder declines the request, the CHILD_SA will be established in tunnel mode. If this is unacceptable to the initiator, the initiator MUST delete the SA. Note: Except when using this option to negotiate transport mode, all CHILD_SAs will use tunnel mode.

Note: The ECN decapsulation modifications specified in [IPSECARCH] MUST be performed for every tunnel mode SA created by IKEv2.

HTTP_CERT_LOOKUP_SUPPORTED 16392

This notification MAY be included in any message that can include a CERTREQ payload and indicates that the sender is capable of looking up certificates based on an HTTP-based URL (and hence presumably would prefer to receive certificate specifications in that format).

REKEY_SA 16393

This notification MUST be included in a CREATE_CHILD_SA exchange if the purpose of the exchange is to replace an existing ESP or AH SA. The SPI field identifies the SA being rekeyed. {{ Clarif-5.4 }} The SPI placed in the REKEY_SA notification is the SPI the exchange initiator would expect in inbound ESP or AH packets. There is no data.

ESP_TFC_PADDING_NOT_SUPPORTED 16394
This notification asserts that the sending endpoint will NOT accept packets that contain Flow Confidentiality (TFC) padding. {{ Clarif-4.5 }} The scope of this message is a single CHILD_SA, and thus this notification is included in messages containing an SA payload negotiating a CHILD_SA. If neither endpoint accepts TFC padding, this notification SHOULD be included in both the request proposing an SA and the response accepting it. If this notification is included in only one of the messages, TFC padding can still be sent in the other direction.

NON_FIRST_FRAGMENTS_ALSO 16395
Used for fragmentation control. See [IPSECARCH] for explanation. {{ Clarif-4.6 }} Sending non-first fragments 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 peer rejects this proposal, the peer only omits NON_FIRST_FRAGMENTS_ALSO notification from the response, but does not reject the whole CHILD_SA creation.

RESERVED TO IANA 16396-40959

PRIVATE USE 40960-65535

3.11. Delete Payload

The Delete Payload, denoted D in this memo, 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, such as ESP or AH, will contain the IPsec protocol ID of that protocol (2 for AH, 3 for ESP), and the SPI is the SPI the sending endpoint would expect in inbound ESP or AH

packets.

The Delete Payload is defined as follows:

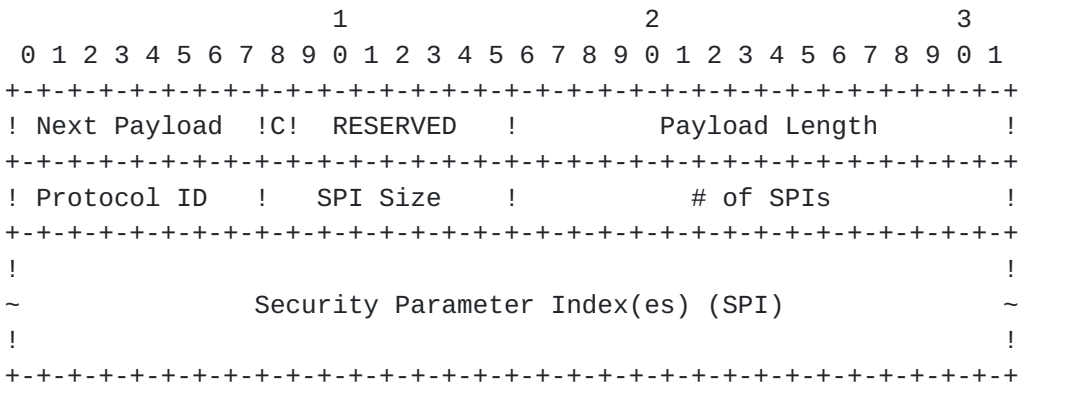

Figure 17: Delete Payload Format

- Protocol ID (1 octet) - Must be 1 for an IKE_SA, 2 for AH, or 3 for ESP.

- 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 AH and ESP.

- # of SPIs (2 octets) - The number of SPIs contained in the Delete payload. The size of each SPI is defined by the SPI Size field.

- 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 # 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 memo, 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 to 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 memo-- private payloads, private exchanges, private notifications, etc. Unfamiliar Vendor IDs MUST be ignored.

Writers of Internet-Drafts who wish to extend this protocol MUST define a Vendor ID payload to announce the ability to implement the extension in the Internet-Draft. It is expected that Internet-Drafts 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:

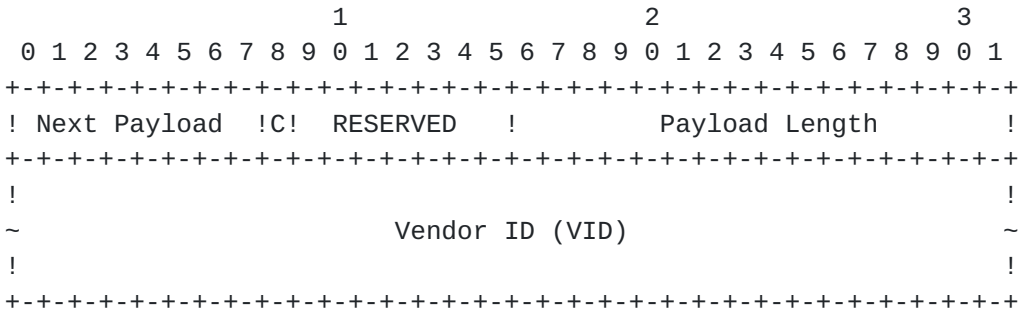

Figure 18: Vendor ID Payload Format

- 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. Good practice is to include a company name, a person name, or some such. If you want to show off, you might include the latitude and longitude and time where you were when you chose the ID and some random input. A message digest of a long unique string is preferable to the long unique 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 memo, 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:

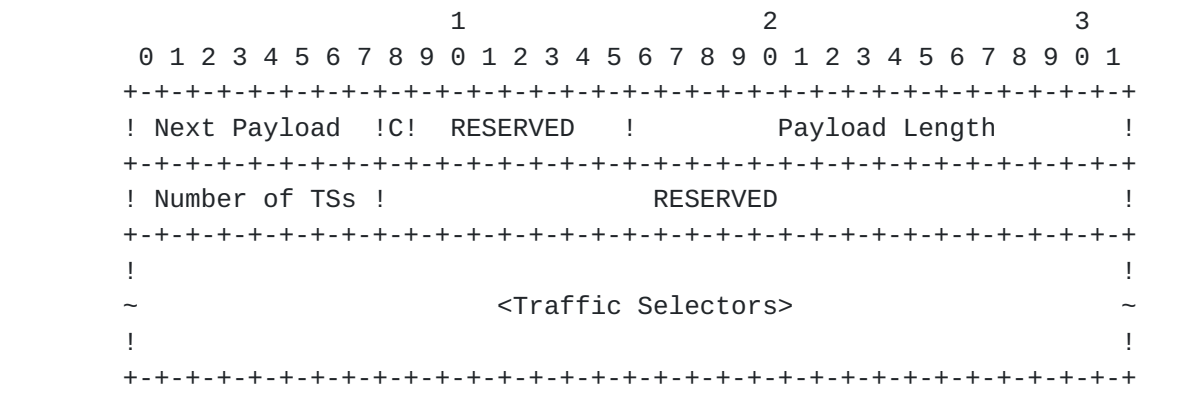

Figure 19: Traffic Selectors Payload Format

- Number of TSs (1 octet) - Number of traffic selectors being provided.

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

- 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.

{{ Clarif-4.7 }} 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, 192.0.1.66-192.0.1.66),
       (17, 200, 192.0.1.66-192.0.1.66))
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 192.0.1.66 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

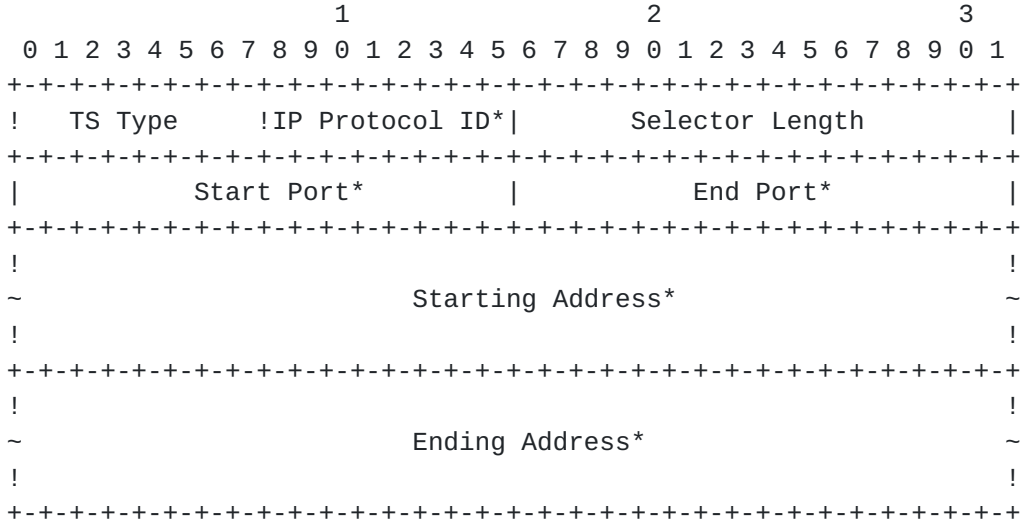

Figure 20: Traffic Selector

*Note: All fields other than TS Type and Selector Length depend on the TS Type. The fields shown are for TS Types 7 and 8, the only two values currently defined.

- TS Type (one octet) - Specifies the type of traffic selector.

- IP protocol ID (1 octet) - Value specifying an associated IP protocol ID (e.g., UDP/TCP/ICMP). A value of zero means that the protocol ID is not relevant to this traffic selector-- the SA can carry all protocols.

- Selector Length - Specifies the length of this Traffic Selector Substructure including the header.

- Start Port (2 octets) - Value specifying the smallest port number allowed by this Traffic Selector. For protocols for which port is undefined, or if all ports are allowed, this field MUST be zero. For the ICMP protocol, the two one-octet fields Type and Code are treated as a single 16-bit integer (with Type in the most significant eight bits and Code in the least significant eight bits) port number for the purposes of filtering based on this field.

- End Port (2 octets) - Value specifying the largest port number allowed by this Traffic Selector. For protocols for which port is undefined, or if all ports are allowed, this field MUST be 65535. For the ICMP protocol, the two one-octet fields Type and Code are treated as a single 16-bit integer (with Type in the most significant eight bits and Code in the least significant eight bits) port number for the purposed of filtering based on this field.

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

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

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.

{{ Added from Clarif-4.8 }} The traffic selector types 7 and 8 can also refer to ICMP type and code fields. Note, however, that ICMP packets do not have separate source and destination port fields. The method for specifying the traffic selectors for ICMP is shown by example in Section 4.4.1.3 of [IPSECARCH].

{{ Added from Clarif-4.9 }} Traffic selectors can use IP Protocol ID 135 to match the IPv6 mobility header [MIPV6]. This document does not specify how to represent the "MH Type" field in traffic selectors, although it is likely that a different document will specify this in the future. Note that [IPSECARCH] says that the IPv6 mobility header (MH) message type is placed in the most significant eight bits of the 16-bit local port selector. The direction semantics of TSi/TSr port fields are the same as for ICMP.

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

TS Type	Value
RESERVED	0-6
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_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.

RESERVED TO IANA	9-240
PRIVATE USE	241-255

3.14. Encrypted Payload

The Encrypted Payload, denoted SK{...} or E in this memo, 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.

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

The encryption and integrity protection algorithms are 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. We require a block cipher with a fixed block size and an integrity check algorithm that computes a fixed-length checksum over a variable size message.

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:

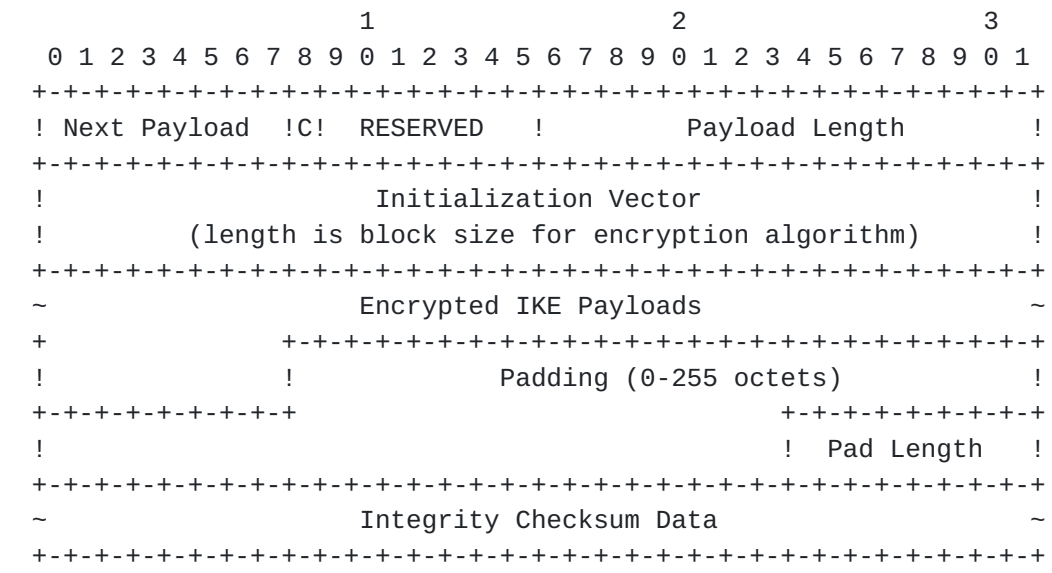

Figure 21: Encrypted Payload Format

- 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.

- Payload Length - Includes the lengths of the header, IV, Encrypted IKE Payloads, Padding, Pad Length, and Integrity Checksum Data.

- Initialization Vector - A randomly chosen value whose length is equal to the block length of the underlying encryption algorithm. Recipients MUST accept any value. Senders SHOULD either pick this value pseudo-randomly and independently for each message or use the final ciphertext block of the previous message sent. Senders MUST NOT use the same value for each message, use a sequence of values with low hamming distance (e.g., a sequence number), or use ciphertext from a received message.

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

- Padding MAY contain any value chosen by the sender, 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.

- Pad Length is the length of the Padding field. The sender SHOULD 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.

- 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.

Configuration payloads are of type CFG_REQUEST/CFG_REPLY or CFG_SET/CFG_ACK (see CFG Type in the payload description below). CFG_REQUEST and CFG_SET payloads may optionally be added to any IKE request. The IKE response MUST include either a corresponding CFG_REPLY or CFG_ACK or a Notify payload with an error type indicating why the request could not be honored. An exception is that a minimal implementation MAY ignore all CFG_REQUEST and CFG_SET payloads, so a response message without a corresponding CFG_REPLY or CFG_ACK MUST be accepted as an indication that the request was not supported.

"CFG_REQUEST/CFG_REPLY" 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. Requestors MUST ignore returned attributes that they do not recognize.

Some attributes MAY be multi-valued, in which case multiple attribute values of the same type are sent and/or returned. Generally, all values of an attribute are returned when the attribute is requested. For some attributes (in this version of the specification only internal addresses), multiple requests indicates a request that multiple values be assigned. For these attributes, the number of values returned SHOULD NOT exceed the number requested.

If the data type requested in a CFG_REQUEST is not recognized or not

supported, the responder MUST NOT return an error type but rather MUST either send a CFG_REPLY that MAY be empty or a reply not containing a CFG_REPLY payload at all. Error returns are reserved for cases where the request is recognized but cannot be performed as requested or the request is badly formatted.

"CFG_SET/CFG_ACK" 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 it 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 minimal implementation of this specification MAY ignore CFG_SET payloads.

{{ Demoted the SHOULD }} Extensions via the CP payload should not be used for general purpose management. Its main intent is to provide a bootstrap mechanism to exchange information within IPsec from IRAS to IRAC. While it MAY be useful to use such a method to exchange information between some Security Gateways (SGW) or small networks, existing management protocols such as DHCP [DHCP], RADIUS [RADIUS], SNMP, or LDAP [LDAP] should be preferred for enterprise management as well as subsequent information exchanges.

The Configuration Payload is defined as follows:

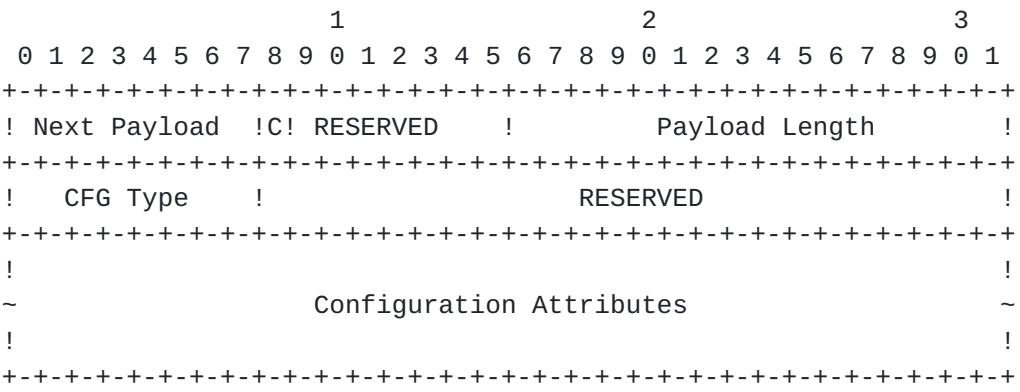

Figure 22: Configuration Payload Format

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

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

CFG Type	Value
RESERVED	0
CFG_REQUEST	1
CFG_REPLY	2
CFG_SET	3
CFG_ACK	4
RESERVED TO IANA	5-127
PRIVATE USE	128-255

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

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

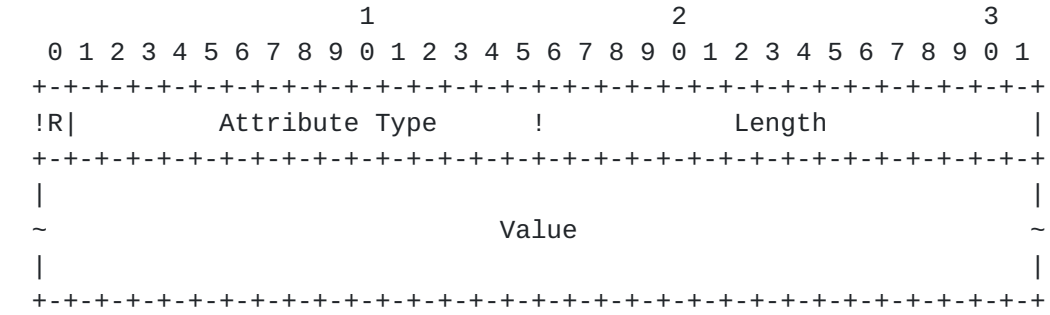

Figure 23: Configuration Attribute Format

- 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) - Length in octets of Value.

- Value (0 or more octets) - The variable-length value of this Configuration Attribute. The following attribute types have been defined:

Attribute Type	Value	Multi-Valued	Length
RESERVED	0		
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_ADDRESS_EXPIRY	5	NO	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
RESERVED	9		
INTERNAL_IP6_DNS	10	YES	0 or 16 octets
INTERNAL_IP6_NBNS	11	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
RESERVED TO IANA	16-16383		
PRIVATE USE	16384-32767		

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

- INTERNAL_IP4_ADDRESS, INTERNAL_IP6_ADDRESS - An address on the internal network, sometimes called a red node address or private address and MAY be a private address on the Internet. {{ Clarif-6.3}} 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 bytes 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 until the expiry time defined with the INTERNAL_ADDRESS_EXPIRY attribute or there are no IKE_SAs between the peers.

- INTERNAL_IP4_NETMASK - The internal network's netmask. Only one netmask is allowed in the request and reply messages (e.g., 255.255.255.0), and it MUST be used only with an

INTERNAL_IP4_ADDRESS attribute. {{ Clarif-6.5 }} 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.

- 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.

- INTERNAL_IP4_NBNS, INTERNAL_IP6_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. {{ Clarif-6.7 }} NetBIOS is not defined for IPv6; therefore, INTERNAL_IP6_NBNS SHOULD NOT be used.

- INTERNAL_ADDRESS_EXPIRY - Specifies the number of seconds that the host can use the internal IP address. The host MUST renew the IP address before this expiry time. Only one of these attributes MAY be present in the reply. {{ Clarif-6.8 }} Expiry times and explicit renewals are primarily useful in environments like DHCP, where the server cannot reliably know when the client has gone away. However, in IKEv2, this is known, and the gateway can simply free the address when the IKE_SA is deleted. Further, supporting renewals is not mandatory. Thus INTERNAL_ADDRESS_EXPIRY attribute MUST NOT be used.

- 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.

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

- 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.

- 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.

- 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.

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

3.15.2. Meaning of INTERNAL_IP4_SUBNET/INTERNAL_IP6_SUBNET

{{ Section added based on Clarif-6.4 }}

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 IPsec SAs whose traffic selectors cover the address in question, new SAs need to be created.

For instance, if there are two subnets, 192.0.1.0/26 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(192.0.1.234)
  INTERNAL_IP4_SUBNET(192.0.1.0/255.255.255.192)
  INTERNAL_IP4_SUBNET(192.0.2.0/255.255.255.0)
TSi = (0, 0-65535, 192.0.1.234-192.0.1.234)
TSr = ((0, 0-65535, 192.0.1.0-192.0.1.63),
       (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 reply would have been this:

```
CP(CFG_REPLY) =
  INTERNAL_IP4_ADDRESS(192.0.1.234)
  INTERNAL_IP4_SUBNET(192.0.1.0/255.255.255.192)
  INTERNAL_IP4_SUBNET(192.0.2.0/255.255.255.0)
TSi = (0, 0-65535, 192.0.1.234-192.0.1.234)
TSr = (0, 0-65535, 0.0.0.0-255.255.255.255)
```

That reply 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(192.0.1.234)
  INTERNAL_IP4_SUBNET(192.0.1.0/255.255.255.192)
  INTERNAL_IP4_SUBNET(192.0.2.0/255.255.255.0)
TSi = (0, 0-65535, 192.0.1.234-192.0.1.234)
TSr = (0, 0-65535, 192.0.1.0-192.0.1.63)
```

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 reply might be:

```
CP(CFG_REPLY) =
  INTERNAL_IP4_ADDRESS(192.0.1.234)
  INTERNAL_IP4_SUBNET(192.0.1.0/255.255.255.192)
  INTERNAL_IP4_SUBNET(192.0.2.0/255.255.255.0)
TSi = (0, 0-65535, 192.0.1.234-192.0.1.234)
TSr = (0, 0-65535, 192.0.2.155-192.0.2.155)
```

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

3.15.3. Configuration payloads for IPv6

{{ Added this section from Clarif-6.6 }}

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.

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 [IPV6ADDR]. 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

{{ Added this section from Clarif-6.9 }}

If the responder encounters an error while attempting to assign an IP address to the initiator, it responds with an INTERNAL_ADDRESS_FAILURE notification. 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.

3.16. Extensible Authentication Protocol (EAP) Payload

The Extensible Authentication Protocol Payload, denoted EAP in this memo, allows IKE_SAs to be authenticated using the protocol defined in RFC 3748 [EAP] and subsequent extensions to that protocol. The full set of acceptable values for the payload is defined elsewhere, but a short summary of RFC 3748 is included here to make this document stand alone in the common cases.

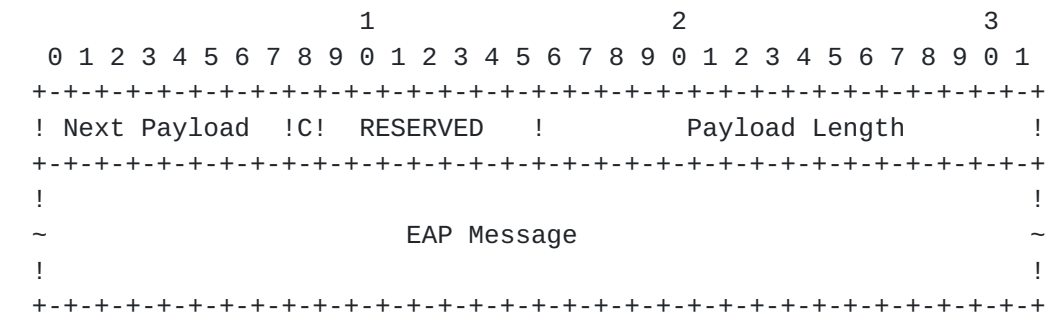

Figure 24: EAP Payload Format

The payload type for an EAP Payload is forty eight (48).

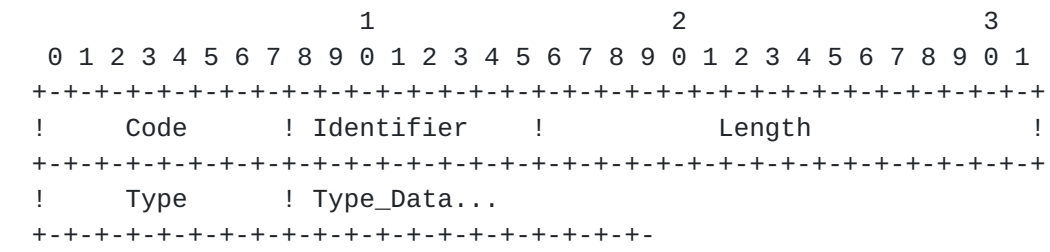

Figure 25: EAP Message Format

- Code (1 octet) indicates whether this message is a Request (1), Response (2), Success (3), or Failure (4).

- Identifier (1 octet) is used in PPP to distinguish replayed messages from repeated ones. Since in IKE, EAP runs over a reliable protocol, it serves no function here. In a response message, this octet MUST be set to match the identifier in the corresponding request. In other messages, this field MAY be set to any value.

- Length (2 octets) is the length of the EAP message and MUST be four less than the Payload Length of the encapsulating payload.

- Type (1 octet) is present only if the Code field is Request (1) or Response (2). For other codes, the EAP message length MUST be four octets and the Type and Type_Data fields MUST NOT be present. In a Request (1) message, Type indicates the data being requested. In a Response (2) message, Type MUST either be Nak or match the type of the data requested. The following types are defined in RFC 3748:

1 Identity
2 Notification
3 Nak (Response Only)
4 MD5-Challenge
5 One-Time Password (OTP)
6 Generic Token Card

- Type_Data (Variable Length) varies with the Type of Request and the associated Response. For the documentation of the EAP methods, see [EAP].

{{ Demoted the SHOULD NOT and SHOULD }} Note that since IKE passes an indication of initiator identity in message 3 of the protocol, the responder should not send EAP Identity requests. The initiator may, however, respond to such requests if it receives them.

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.

IKEv2 is designed to permit minimal implementations that can interoperate with all compliant implementations. There are a series of optional features that can easily be ignored by a particular implementation if it does not support that feature. Those features include:

- Ability to negotiate SAs through a NAT and tunnel the resulting ESP SA over UDP.

- Ability to request (and respond to a request for) a temporary IP address on the remote end of a tunnel.

- Ability to support various types of legacy authentication.

- Ability to support window sizes greater than one.

- Ability to establish multiple ESP and/or AH SAs within a single IKE_SA.

o Ability to rekey SAs.

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 four-message IKE_SA_INIT and IKE_AUTH exchanges establishing two SAs (one for IKE, one for ESP and/or AH). 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 INFORMATIONAL message with an empty INFORMATIONAL reply (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 MAY 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), and 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. 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, it MUST include a CP payload in message 3 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 message 3 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. {{ Demoted the SHOULD }} The responder may include any other related attributes.

A minimal IPv4 responder implementation will ignore the contents of the CP payload except to determine that it includes an INTERNAL_IP4_ADDRESS attribute and will respond with the address and other related attributes regardless of whether the initiator requested them.

A minimal IPv4 initiator will generate a CP payload containing only an INTERNAL_IP4_ADDRESS attribute and will parse the response ignoring attributes it does not know how to use. {{ Clarif-6.8 removes the sentence about processing INTERNAL_ADDRESS_EXPIRY. }} Minimal initiators need not be able to request lease renewals and minimal responders need not respond to them.

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

- PKIX Certificates containing and signed by RSA keys of size 1024 or 2048 bits, where the ID passed is any of ID_KEY_ID, ID_FQDN, ID_RFC822_ADDR, or ID_DER_ASN1_DN.

- Shared key authentication where the ID passes is any of ID_KEY_ID, ID_FQDN, or ID_RFC822_ADDR.

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

5. Security Considerations

While this protocol is designed to minimize disclosure of configuration information to unauthenticated peers, some such disclosure is unavoidable. One peer or the other must identify itself first and prove its identity first. To avoid probing, the initiator of an exchange is required to identify itself first, and usually is required to authenticate itself first. The initiator can, however, learn that the responder supports IKE and what cryptographic protocols it supports. The responder (or someone impersonating the responder) can probe the initiator not only for its identity, but using CERTREQ payloads may be able to determine what certificates the initiator is willing to use.

Use of EAP authentication changes the probing possibilities somewhat. When EAP authentication is used, the responder proves its identity before the initiator does, so an initiator that knew the name of a valid initiator could probe the responder for both its name and certificates.

Repeated rekeying using CREATE_CHILD_SA without additional Diffie-Hellman exchanges leaves all SAs vulnerable to cryptanalysis of a single key or overrun of either endpoint. Implementers should take note of this fact and set a limit on CREATE_CHILD_SA exchanges between exponentiations. This memo does not prescribe such a limit.

The strength of a key derived from a Diffie-Hellman exchange using any of the groups defined here depends on the inherent strength of the group, the size of the exponent used, and the entropy provided by the random number generator used. Due to these inputs, it is difficult to determine the strength of a key for any of the defined groups. Diffie-Hellman group number two, when used with a strong random number generator and an exponent no less than 200 bits, is common for use with 3DES. Group five provides greater security than group two. Group one is for historic purposes only and does not provide sufficient strength except for use with DES, which is also for historic use only. Implementations should make note of these estimates when establishing policy and negotiating security parameters.

Note that these limitations are on the Diffie-Hellman groups themselves. There is nothing in IKE that prohibits using stronger groups nor is there anything that will dilute the strength obtained from stronger groups (limited by the strength of the other algorithms negotiated including the prf function). In fact, the extensible framework of IKE encourages the definition of more groups; use of elliptical curve groups may greatly increase strength using much smaller numbers.

It is assumed that all Diffie-Hellman exponents are erased from memory after use. In particular, these exponents MUST NOT be derived from long-lived secrets like the seed to a pseudo-random generator that is not erased after use.

The strength of all keys is limited by the size of the output of the negotiated prf function. For this reason, a prf function whose output is less than 128 bits (e.g., 3DES-CBC) MUST NOT be used with this protocol.

The security of this protocol is critically dependent on the randomness of the randomly chosen parameters. These should be generated by a strong random or properly seeded pseudo-random source (see [RANDOMNESS]). Implementers should take care to ensure that use of random numbers for both keys and nonces is engineered in a fashion that does not undermine the security of the keys.

For information on the rationale of many of the cryptographic design choices in this protocol, see [SIGMA] and [SKEME]. Though the security of negotiated CHILD_SAs does not depend on the strength of the encryption and integrity protection negotiated in the IKE_SA, implementations MUST NOT negotiate NONE as the IKE integrity protection algorithm or ENCR_NULL as the IKE encryption algorithm.

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.

The NAT_DETECTION_*_IP notifications contain a hash of the addresses and ports in an attempt to hide internal IP addresses behind a NAT. Since the IPv4 address space is only 32 bits, and it is usually very sparse, it would be possible for an attacker to find out the internal address used behind the NAT box by trying all possible IP addresses and trying to find the matching hash. The port numbers are normally fixed to 500, and the SPIs can be extracted from the packet. This reduces the number of hash calculations to 2^32. With an educated guess of the use of private address space, the number of hash calculations is much smaller. Designers should therefore not assume that use of IKE will not leak internal address information.

When using an EAP authentication method that does not generate a shared key for protecting a subsequent AUTH payload, certain man-in-the-middle and server impersonation attacks are possible [EAPMITM]. These vulnerabilities occur when EAP is also used in protocols that are not protected with a secure tunnel. Since EAP is a general-purpose authentication protocol, which is often used to provide single-signon facilities, a deployed IPsec solution that relies on an EAP authentication method that does not generate a shared key (also known as a non-key-generating EAP method) can become compromised due to the deployment of an entirely unrelated application that also happens to use the same non-key-generating EAP method, but in an unprotected fashion. Note that this vulnerability is not limited to just EAP, but can occur in other scenarios where an authentication infrastructure is reused. For example, if the EAP mechanism used by IKEv2 utilizes a token authenticator, a man-in-the-middle attacker could impersonate the web server, intercept the token authentication exchange, and use it to initiate an IKEv2 connection. For this reason, use of non-key-generating EAP methods SHOULD be avoided where possible. Where they are used, it is extremely important that all usages of these EAP methods SHOULD utilize a protected tunnel, where the initiator validates the responder's certificate before initiating the EAP exchange. {{ Demoted the SHOULD }} Implementers should describe the vulnerabilities of using non-key-generating EAP methods in the documentation of their implementations so that the administrators deploying IPsec solutions are aware of these dangers.

An implementation using EAP MUST also use a public-key-based authentication of the server to the client before the EAP exchange begins, even if the EAP method offers mutual authentication. This avoids having additional IKEv2 protocol variations and protects the

EAP data from active attackers.

If the messages of IKEv2 are long enough that IP-level fragmentation is necessary, it is possible that attackers could prevent the exchange from completing by exhausting the reassembly buffers. The chances of this can be minimized by using the Hash and URL encodings instead of sending certificates (see Section 3.6). Additional mitigations are discussed in [DOSUDPPROT].

6. IANA Considerations

{{ This section was changed to not re-define any new IANA registries. }}

[IKEV2] defined many field types and values. IANA has already registered those types and values, so the are not listed here again. No new types or values are registered in IKEv2.1.

7. Acknowledgements

{{ Added new acknowledgements. }}

Charlie Kaufman did a huge amount of work on the original IKEv2 document, on which this document is primarily based. Pasi Eronen worked hard on the clarifications document, which is the basis for the differences between IKEv2 and IKEv2.1. The individuals on the IPsec mailing list was very helpful in both pointing out where clarifications and changes were needed, as well as in reviewing the clarifications suggested by others.

The acknowledgements from the IKEv2 document were:

This document is a collaborative effort of the entire IPsec WG. If there were no limit to the number of authors that could appear on an RFC, the following, in alphabetical order, would have been listed: Bill Aiello, Stephane Beaulieu, Steve Bellovin, Sara Bitan, Matt Blaze, Ran Canetti, Darren Dukes, Dan Harkins, Paul Hoffman, John Ioannidis, Charlie Kaufman, Steve Kent, Angelos Keromytis, Tero Kivinen, Hugo Krawczyk, Andrew Krywaniuk, Radia Perlman, Omer Reingold, and Michael Richardson. Many other people contributed to the design. It is an evolution of IKEv1, ISAKMP, and the IPsec DOI, each of which has its own list of authors. Hugh Daniel suggested the feature of having the initiator, in message 3, specify a name for the responder, and gave the feature the cute name "You Tarzan, Me Jane". David Faucher and Valery Smyzlov helped refine the design of the traffic selector negotiation.

This paragraph lists references that appear only in figures. The section is only here to keep the 'xml2rfc' program happy, and will be removed when the document is published. Feel free to ignore it.
[DES] [IDEA] [MD5] [X.501] [X.509]

8. References

8.1. Normative References

[ADDGROUP]
Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, May 2003.

[ADDRIPV6]
Hinden, R. and S. Deering, "Internet Protocol Version 6 (IPv6) Addressing Architecture", RFC 3513, April 2003.

[Clarif] "IKEv2 Clarifications and Implementation Guidelines", draft-eronen-ipsec-ikev2-clarifications (work in progress).

[EAP] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[ECN] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[ESPCBC] Pereira, R. and R. Adams, "The ESP CBC-Mode Cipher Algorithms", RFC 2451, November 1998.

[IANACONS]
Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434.

[IKEV2] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[IPSECARCH]
Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[MUSTSHOULD]
Bradner, S., "Key Words for use in RFCs to indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[PKIX] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[UDPENCAPS] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", RFC 3948, January 2005.

8.2. Informative References

[AH] Kent, S., "IP Authentication Header", RFC 4302, December 2005.

[ARCHGUIDEPHIL] Bush, R. and D. Meyer, "Some Internet Architectural Guidelines and Philosophy", RFC 3439, December 2002.

[ARCHPRINC] Carpenter, B., "Architectural Principles of the Internet", RFC 1958, June 1996.

[DES] American National Standards Institute, "American National Standard for Information Systems-Data Link Encryption", ANSI X3.106, 1983.

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

[DHCP] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, March 1997.

[DIFFSERVARCH] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475.

[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.

[DIFFTUNNEL] Black, D., "Differentiated Services and Tunnels", RFC 2983, October 2000.

[DOI] Piper, D., "The Internet IP Security Domain of Interpretation for ISAKMP", RFC 2407, November 1998.

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

[DSS] National Institute of Standards and Technology, U.S. Department of Commerce, "Digital Signature Standard", FIPS 186, May 1994.

[EAPMITM] N. Asokan, V. Nierni, and K. Nyberg, "Man-in-the-Middle in Tunneled Authentication Protocols", November 2002, <http://eprint.iacr.org/2002/163>.

[ESP] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

[EXCHANGEANALYSIS] R. Perlman and C. Kaufman, "Analysis of the IPsec key exchange Standard", WET-ICE Security Conference, MIT , 2001, <http://sec.femto.org/wetice-2001/papers/radia-paper.pdf>.

[HMAC] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[IDEA] X. Lai, "On the Design and Security of Block Ciphers", ETH Series in Information Processing, v. 1, Konstanz: Hartung-Gorre Verlag, 1992.

[IKEV1] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[IPCOMP] Shacham, A., Monsour, B., Pereira, R., and M. Thomas, "IP Payload Compression Protocol (IPComp)", RFC 3173, September 2001.

[IPSECARCH-OLD] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[IPV6ADDR] Hinden, R. and S. Deering, "Internet Protocol Version 6 (IPv6) Addressing Architecture", RFC 3513, April 2003.

[ISAKMP] Maughan, D., Schneider, M., and M. Schertler, "Internet Security Association and Key Management Protocol (ISAKMP)", RFC 2408, November 1998.

[LDAP] Wahl, M., Howes, T., and S. Kille, "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997.

[MAILFORMAT] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[MD5] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[MIPV6] Johnson, D., Perkins, C., and J. Arkko, "Mobility Support in IPv6", RFC 3775, June 2004.

[MLDV2] Vida, R. and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, June 2004.

[NAI] Aboba, B. and M. Beadles, "The Network Access Identifier", RFC 2486, January 1999.

[NATREQ] Aboba, B. and W. Dixon, "IPsec-Network Address Translation (NAT) Compatibility Requirements", RFC 3715, March 2004.

[OAKLEY] Orman, H., "The OAKLEY Key Determination Protocol", RFC 2412, November 1998.

[PFKEY] McDonald, D., Metz, C., and B. Phan, "PF_KEY Key Management API, Version 2", RFC 2367, July 1998.

[PHOTURIS] Karn, P. and W. Simpson, "Photuris: Session-Key Management Protocol", RFC 2522, March 1999.

[PKCS1] B. Kaliski and J. Staddon, "PKCS #1: RSA Cryptography Specifications Version 2", September 1998.

[PRFAES128CBC] Hoffman, P., "The AES-XCBC-PRF-128 Algorithm for the Internet Key Exchange Protocol (IKE)", RFC 3664, January 2004.

[PRFAES128CBC-bis] Hoffman, P., "The AES-XCBC-PRF-128 Algorithm for the Internet Key Exchange Protocol (IKE)", draft-hoffman-rfc3664bis (work in progress), October 2005.

[RADIUS] Rigney, C., Rubens, A., Simpson, W., and S. Willens, "Remote Authentication Dial In User Service (RADIUS)", RFC 2138, April 1997.

[RANDOMNESS] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[REAUTH] Nir, Y., ""Repeated Authentication in IKEv2", draft-nir-ikev2-auth-lt (work in progress), May 2005.

[RSA] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", February 1978.

[SHA] National Institute of Standards and Technology, U.S. Department of Commerce, "Secure Hash Standard", FIPS 180-1, May 1994.

[SIGMA] H. Krawczyk, "SIGMA: the `SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and its Use in the IKE Protocols", Advances in Cryptography - CRYPTO 2003 Proceedings LNCS 2729, 2003, <http://www.informatik.uni-trier.de/~ley/db/conf/crypto/crypto2003.html>.

[SKEME] H. Krawczyk, "SKEME: A Versatile Secure Key Exchange Mechanism for Internet", IEEE Proceedings of the 1996 Symposium on Network and Distributed Systems Security , 1996.

[TRANSPARENCY] Carpenter, B., "Internet Transparency", RFC 2775, February 2000.

[X.501] ITU-T, "Recommendation X.501: Information Technology - Open Systems Interconnection - The Directory: Models", 1993.

[X.509] ITU-T, "Recommendation X.509 (1997 E): Information Technology - Open Systems Interconnection - The Directory: Authentication Framework", 1997.

Appendix A. Summary of changes from IKEv1

The goals of this revision to IKE are:

1. To define the entire IKE protocol in a single document, replacing RFCs 2407, 2408, and 2409 and incorporating subsequent changes to support NAT Traversal, Extensible Authentication, and Remote Address acquisition;

2. To simplify IKE by replacing the eight different initial exchanges with a single four-message exchange (with changes in authentication mechanisms affecting only a single AUTH payload rather than restructuring the entire exchange) see [EXCHANGEANALYSIS];

3. To remove the Domain of Interpretation (DOI), Situation (SIT), and Labeled Domain Identifier fields, and the Commit and Authentication only bits;

4. To decrease IKE's latency in the common case by making the initial exchange be 2 round trips (4 messages), and allowing the ability to piggyback setup of a CHILD_SA on that exchange;

5. To replace the cryptographic syntax for protecting the IKE messages themselves with one based closely on ESP to simplify implementation and security analysis;

6. To reduce the number of possible error states by making the protocol reliable (all messages are acknowledged) and sequenced. This allows shortening CREATE_CHILD_SA exchanges from 3 messages to 2;

7. To increase robustness by allowing the responder to not do significant processing until it receives a message proving that the initiator can receive messages at its claimed IP address, and not commit any state to an exchange until the initiator can be cryptographically authenticated;

8. To fix cryptographic weaknesses such as the problem with symmetries in hashes used for authentication documented by Tero Kivinen;

9. To specify Traffic Selectors in their own payloads type rather than overloading ID payloads, and making more flexible the Traffic Selectors that may be specified;

10. To specify required behavior under certain error conditions or when data that is not understood is received in order to make it easier to make future revisions in a way that does not break backwards compatibility;

11. To simplify and clarify how shared state is maintained in the presence of network failures and Denial of Service attacks; and

12. To maintain existing syntax and magic numbers to the extent possible to make it likely that implementations of IKEv1 can be enhanced to support IKEv2 with minimum effort.

Appendix B. Diffie-Hellman Groups

There are two Diffie-Hellman groups defined here for use in IKE. These groups were generated by Richard Schroeppel at the University of Arizona. Properties of these primes are described in [OAKLEY].

The strength supplied by group one may not be sufficient for the mandatory-to-implement encryption algorithm and is here for historic reasons.

Additional Diffie-Hellman groups have been defined in [ADDGROUP].

B.1. Group 1 - 768 Bit MODP

This group is assigned id 1 (one).

The prime is: 2^768 - 2 ^704 - 1 + 2^64 * { [2^638 pi] + 149686 }
Its hexadecimal value is:

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A63A3620 FFFFFFFF FFFFFFFF
```

The generator is 2.

B.2. Group 2 - 1024 Bit MODP

This group is assigned id 2 (two).

The prime is 2^1024 - 2^960 - 1 + 2^64 * { [2^894 pi] + 129093 }.
Its hexadecimal value is:

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE65381
FFFFFFFF FFFFFFFF
```

The generator is 2.

Appendix C. Exchanges and Payloads

{{ Clarif-AppA }}

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

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

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

C.2. IKE_AUTH Exchange without EAP

```
request             --> IDi, [CERT+],
                        [N(INITIAL_CONTACT)],
                        [[N(HTTP_CERT_LOOKUP_SUPPORTED)], CERTREQ+],
                        [IDr],
                        AUTH,
                        [CP(CFG_REQUEST)],
                        [N(IPCOMP_SUPPORTED)+],
                        [N(USE_TRANSPORT_MODE)],
                        [N(ESP_TFC_PADDING_NOT_SUPPORTED)],
                        [N(NON_FIRST_FRAGMENTS_ALSO)],
                        SA, TSi, TSr,
                        [V+]

response            <-- IDr, [CERT+],
                        AUTH,
                        [CP(CFG_REPLY)],
                        [N(IPCOMP_SUPPORTED)],
                        [N(USE_TRANSPORT_MODE)],
                        [N(ESP_TFC_PADDING_NOT_SUPPORTED)],
                        [N(NON_FIRST_FRAGMENTS_ALSO)],
                        SA, TSi, TSr,
                        [N(ADDITIONAL_TS_POSSIBLE)],
                        [V+]
```

C.3. IKE_AUTH Exchange with EAP

```
   first request       --> IDi,
                           [N(INITIAL_CONTACT)],
                           [[N(HTTP_CERT_LOOKUP_SUPPORTED)], CERTREQ+],
                           [IDr],
                           [CP(CFG_REQUEST)],
                           [N(IPCOMP_SUPPORTED)+],
                           [N(USE_TRANSPORT_MODE)],
                           [N(ESP_TFC_PADDING_NOT_SUPPORTED)],
                           [N(NON_FIRST_FRAGMENTS_ALSO)],
                           SA, TSi, TSr,
                           [V+]

   first response      <-- IDr, [CERT+], AUTH,
                           EAP,
                           [V+]

                     / --> EAP
   repeat 1..N times |
                     \ <-- EAP

   last request        --> AUTH

   last response       <-- AUTH,
                           [CP(CFG_REPLY)],
                           [N(IPCOMP_SUPPORTED)],
                           [N(USE_TRANSPORT_MODE)],
                           [N(ESP_TFC_PADDING_NOT_SUPPORTED)],
                           [N(NON_FIRST_FRAGMENTS_ALSO)],
                           SA, TSi, TSr,
                           [N(ADDITIONAL_TS_POSSIBLE)],
                           [V+]
```

C.4. CREATE_CHILD_SA Exchange for Creating or Rekeying CHILD_SAs

```
request             --> [N(REKEY_SA)],
                         [N(IPCOMP_SUPPORTED)+],
                         [N(USE_TRANSPORT_MODE)],
                         [N(ESP_TFC_PADDING_NOT_SUPPORTED)],
                         [N(NON_FIRST_FRAGMENTS_ALSO)],
                         SA, Ni, [KEi], TSi, TSr

response            <-- [N(IPCOMP_SUPPORTED)],
                         [N(USE_TRANSPORT_MODE)],
                         [N(ESP_TFC_PADDING_NOT_SUPPORTED)],
                         [N(NON_FIRST_FRAGMENTS_ALSO)],
                         SA, Nr, [KEr], TSi, TSr,
                         [N(ADDITIONAL_TS_POSSIBLE)]
```

C.5. CREATE_CHILD_SA Exchange for Rekeying the IKE_SA

```
request             --> SA, Ni, [KEi]

response            <-- SA, Nr, [KEr]
```

C.6. INFORMATIONAL Exchange

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

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

Appendix D. Changes Between Internet Draft Versions

This section will be removed before publication as an RFC.

D.1. Changes from IKEv2 to draft -00

There were a zillion additions from the Clarifications document. These are noted with "{{ Clarif-nn }}".

Cleaned up many of the figures. Made the table headings consistent. Made some tables easier to read by removing blank spaces. Removed the "reserved to IANA" and "private use" text wording and moved it into the tables.

Changed many SHOULD and MUST requirements to better match RFC 2119.

Author's Address

Paul Hoffman
VPN Consortium
127 Segre Place
Santa Cruz, CA 95060
US

Phone: 1-831-426-9827
Email: paul.hoffman@vpnc.org

Full Copyright Statement

Intellectual Property

copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.