IPSEC Working Group
INTERNET-DRAFT

Dan Harkins
Charlie Kaufman
Radia Perlman
editors

draft-ietf-ipsec-ikev2-00.txt

November 2001

The Internet Key Exchange (IKE) Protocol
<draft-ietf-ipsec-ikev2-00.txt>

Status of this Memo

This document is an Internet Draft and is in full conformance with all provisions of Section 10 of RFC2026 [Bra96]. Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and working groups. Note that other groups may also distribute working documents as Internet Drafts.

Internet Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Abstract

This document describes version 2 of the IKE (Internet Key Exchange) protocol. IKE performs mutual authentication and establishes an IKE security association that can be used to efficiently establish SAs for ESP, AH and/or IPcomp. This version greatly simplifies IKE by replacing the 8 possible phase 1 exchanges with a single exchange based on public signature keys. The single exchange provides identity hiding, yet works in 2 round trips (all the identity hiding exchanges in IKE v1 required 3 round trips). Latency of setup of an IPsec SA is further reduced from IKEv1 by allowing setup of an SA for ESP, AH, and/or IPcomp to be piggybacked on the initial IKE exchange. It also improves security by allowing the Responder to be stateless until it can be assured that the Initiator can receive at the claimed IP source address. This version also presents the entire protocol in a single self-contained document, in contrast to IKEv1, in which the protocol was described in ISAKMP (RFC 2408), IKE (RFC 2409), and the

DOI (RFC 2407) documents.

Table of Contents

1. Introduction....3
1.1 The IKE Protocol....3
1.2 Changes from IKEv1....4
1.3 Requirements Terminology....5
2 Protocol Overview....5
2.1 Use of Retransmission Timers....6
2.2 Use of Sequence Numbers for Message ID....6
2.3 Window Size for overlapping requests....7
2.4 State Synchronization and Connection Timeouts....7
2.5 Version Numbers and Forward Compatibility....8
2.6 Cookies....10
2.7 Cryptographic Algorithm Negotiation....12
2.8 Rekeying....13
2.9 Traffic Selector Negotiation....14
2.10 Nonces....15
3 The Phase 1 Exchange....15
3.1 Generating Keying Material for the IKE-SA....17
3.2 Authentication of the IKE-SA....17
4 The CREATE-CHILD-SA (Phase 2) Exchange....18
4.1 Generating Keying Material for Child-SAs....19
4.2 Generating Keying Material for IKE-SAs during rollover...20
5 Informational (Phase 2) Exchange....20
6 Error Handling....22
7 Header and Payload Formats....23
7.1 The IKE Header....23
7.2 Generic Payload Header....26
7.3 Security Association Payload....27
7.3.1 Proposal Substructure....29
7.3.2 Transform Substructure....31
7.3.3 Mandatory Transform Types....34
7.3.4 Mandatory Transform-IDs....34
7.3.5 Transform Attributes....35
7.3.6 Attribute Negotiation....36
7.4 Key Exchange Payload....36
7.5 Identification Payload....37
7.6 Certificate Payload....39
7.7 Certificate Request Payload....40
7.8 Authentication Payload....41
7.9 Nonce Payload....42
7.10 Notify Payload....43
7.10.1 Notify Message Types....44
7.11 Delete Payload....48

7.12 Vendor ID Payload.....49
7.13 Traffic Selector Payload.....50
7.13.1 Traffic Selector Substructure.....51
7.14 Other Payload types.....53
8 Diffie-Hellman Groups.....53
9 Security Considerations.....55
10 IANA Considerations.....56
10.1 Transform Types and Attribute Values.....56
10.2 Exchange Types.....57
10.3 Payload Types.....57
11 Acknowledgements.....58
12 References.....58
Appendix A: Attribute Assigned Numbers.....61
Appendix B: Cryptographic Protection of IKE Data.....63
Authors' Addresses.....64

1. Introduction

IP Security (IPsec) provides confidentiality, data integrity, 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 of IKE. Version 1 of IKE was defined in RFCs 2407, 2408, and 2409. This single document is intended to replace all three of those RFCs.

1.1 The IKE Protocol

IKE performs mutual authentication between two parties and establishes an IKE security association that includes shared secret information that can be used to efficiently establish SAs for ESP (RFC 2406), AH (RFC 2402) and/or IPcomp (RFC 2393). We call the IKE SA an "IKE-SA", and the SAs for ESP, AH, and/or IPcomp that get set up through that IKE-SA we call "child-SA"s.

We call the setup of the IKE-SA "phase 1" and subsequent IKE exchanges "phase 2" even though setup of a child-SA can be piggybacked on the initial phase 1 exchange. The phase 1 exchange is two request/response pairs. A phase 2 exchange is one request/response pair, and can be used to create or delete a child-SA, rekey or delete the IKE-SA, or give information such as error

conditions.

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 retransmits the request.

The first request/response of a phase 1 exchange, which we'll call IKE_SA_init, negotiates security parameters for the IKE-SA, and sends Diffie-Hellman values. We call the response IKE_SA_init_response.

The second request/response, which we'll call IKE_auth, transmits identities, proves knowledge of the private signature key, and optionally sets up an SA for AH and/or ESP and/or IPcomp. We call the response IKE_auth_response.

If the Responder feels it is under attack, and wishes to use a stateless cookie (see section on cookies). it can respond to an IKE_SA_init with an IKE_SA_init_reject with a cookie value that must be sent with a subsequent IKE_SA_init_request. The Initiator then sends another IKE_SA_init, but this time including the Responder's cookie value.

Phase 2 exchanges each consist of a single request/response pair. The types of exchanges are CREATE_CHILD_SA (creates a child-SA), or an informational exchange which deletes a child-SA or the IKE-SA or informs the other side of some error condition. All these messages require a response, so an informational message with no payloads can serve as a check for aliveness.

1.2 Changes from IKEv1

The goals of this revision to IKE are:

1) To define the entire IKE protocol in a single document, rather than three that cross reference one another;

2) To simplify IKE by eliminating the Aggressive Mode option and all but one of the authentication algorithms making phase 1 a single exchange (based on public signature keys);

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 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 algorithms 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 Phase 2 exchanges from 3 messages to 2;

7) To increase robustness by allowing the Responder, if under attack, to require return of a cookie before the Responder commits any state to the exchange;

8) To fix bugs such as the hash problem documented in [draft-ietf-ipsec-ike-hash-revised-02.txt];

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

10) To avoid unnecessary exponential explosion of space in attribute negotiation, by allowing choices when multiple algorithms of one type (say, encryption) can work with any of a number of acceptable algorithms of another type (say, integrity protection);

11) 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;

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

13) 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.

1.3 Requirements Terminology

Keywords "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT" and "MAY" that appear in this document are to be interpreted as described in [Bra97].

2 Protocol Overview

IKE runs over UDP port 500. 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 at least one of a series of retransmitted packets reaches its destination before timing out and 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).

2.1 Use of Retransmission Timers

All messages in IKE exist in pairs: a request and a response. Either end of a 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 labelled as either a request or a response and for each pair one end of the security association is the Initiator and the other is the Responder.

For every pair of messages, the Initiator is responsible for retransmission in the event of a timeout. The Responder will never retransmit a response unless it receives a retransmission of the request. In that event, the Responder MUST either ignore the retransmitted request except insofar as it triggers a retransmission of the response OR if the request is idempotent, the Responder may choose to process the request again and send a semantically equivalent reply.

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.

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, with is zero for the first IKE message. 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 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 setup, each integer n will 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 each packet contains enough information to determine which of the four messages a particular one is.

In the case where the IKE_SA_init is rejected (e.g. in order to require a cookie), the second IKE_SA_init message will begin the sequence over with Message #0.

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. 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 assure 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. An IKE endpoint SHOULD be prepared to accept and process multiple requests while it has a request outstanding.

An IKE endpoint MUST NOT exceed the peer's stated window size (see section 7.3.2) for transmitted IKE requests. In other words, if Bob stated his window size is N, then when Alice needs to make a request X, she MUST wait until she 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 with semantic equivalence) the number of previous responses equal to its contracted window size in case its response was lost and the Initiator requests its retransmission by retransmitting the request.

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

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 those 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. 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 provides a null query notify message that requires an acknowledgment. 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 generate responses to 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. An exception to this rule is that a Responder who has not received a cryptographically protected message on an IKE-SA MUST eventually time it out and delete it. Note that consuming state on an IKE Responder by setting up large numbers of half-open IKE-SAs is a likely denial of service attack, so the policy for timing these out and limiting the resources they consume should be considered carefully.

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 acknowledgment to an IKE message, then the IKE SA and all child-SAs set up through that IKE-SA are deleted.

An IKE endpoint MAY delete inactive Child-SAs to recover resources used to hold their state. If an IKE endpoint chooses to do so, it MUST send Delete payloads to the other end notifying it of the deletion. It MAY similarly time out the IKE-SA. Closing the IKE-SA implicitly closes all associated Child-SAs. An IKE endpoint SHOULD send a Delete payload indicating that it has closed the IKE-SA.

2.5 Version Numbers and Forward Compatibility

This document describes version 2.0 of IKE, meaning the major version number is 2 and the minor version number is zero. It is likely that

some implementations will want to support both version 1.0 and version 2.0, and in the future, other versions.

The major version number should only be incremented if the packet formats have changed so dramatically that an older version node would not be able to interoperate with messages in the new version format. 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.

If you receive a message with a higher major version number, you MUST drop the message and SHOULD send an unauthenticated notification message containing the highest version number you support. If you support major version n, and major version m, you MUST support all versions between n and m. If you receive a message with a major version that you support, you 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 A is capable of speaking versions n, n+1, and n+2, and B is capable of speaking versions n and n+1, then they will negotiate speaking n+1, where A will set the flag indicating 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 v1 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. Given the design of v1, there is no way of preventing this, but this version number discipline will prevent such problems in future versions.

Also for forward compatibility, all fields marked RESERVED MUST be set to zero by a version 2.0 implementation and their content MUST be ignored by a version 2.0 implementation ("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, field types that are not defined are reserved for future use and implementations of version 2.0 MUST skip over those fields 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 unsupported, the message MUST be rejected and the response to the IKE request containing that payload MUST include a notify payload INVALID-PAYLOAD-TYPE, indicating an unsupported critical payload was included. If the critical flag is not set and the payload type is unsupported, that payload is simply skipped.

2.6 Cookies

The term "cookies" originates with Karn and Simpson [RFC 2522] in Photurus, an early proposal for key managment with IPsec. It has persisted because the IETF has never rejected an offer involving cookies. In IKEv2, the cookies serve two purposes. First, they are used as IKE-SA identifiers in the headers of IKE messages. As with ESP and AH, in IKEv2 the recipient of a message chooses an IKE-SA identifier that uniquely defines that SA to that recipient. For this purpose (IKE-SA identifiers), it might be convenient for the cookie value to be chosen so as to be a table index for fast lookups of SAs. But this conflicts with the second purpose of the cookies (to be explained shortly).

Unlike ESP and AH where only the recipient's SA identifier appears in the message, in IKE, the sender's IKE SA identifier is also sent in every message. In IKEv1 the IKE-SA identifier consisted of the pair (Initiator cookie, Responder cookie), whereas in IKEv2, the SA is uniquely defined by the recipient's SA identifier even though both are included in the IKEv2 header.

The second use of cookies in IKEv2 is for a limited protection from denial of service attacks. Receipt of a request to start an SA can consume substantial resources. A likely denial of service attack against IKE is to overwhelm a system with large numbers of SA requests from forged IP addresses. This can consume CPU resources doing the crypto, and memory resources remembering the state of the "half open" connections until they time out. A robust design would limit the resources it is willing to devote to new connection establishment, but even so the denial of service attack could effectively prevent any new connections.

This attack can be rendered more difficult by requiring that the Responder to an SA request do minimal computation and allocate no memory until the Initiator has proven that it can receive messages at

the address it claims to be sending from. This is done in a stateless way by computing the cookie in a way that the Responder can recompute the same value, but the Initiator can't guess it. A recommended strategy is to compute the cookie as a cryptographic hash of the Initiator's IP address, the Initiator's cookie value (its chosen IKE security identifier), and a secret known only to the Responder. That secret should be changed periodically to prevent the "cookie jar" attack where an attacker accumulates lots of cookies from lots of IP addresses over time and then replays them all at once to overwhelm the Responder.

In ISAKMP and IKEv1, the term cookie was used for the connection identifier, but the protocol did not permit their use against this particular denial of service attack. To avoid the cookie exchange adding extra messages to the protocol in the common case where the Responder is not under attack, IKEv2 goes back to the approach in Oakley (RFC 2412) where the cookie challenge is optional. Upon receipt of an IKE_SA_init, a Responder may either proceed with setting up the SA or may tell the Initiator to send another IKE_SA_init, this time providing a supplied cookie.

It may be convenient for the IKE-SA identifier to be an index into a table. It is not difficult for the Initiator to choose an IKE-SA identifier that is convenient as a table identifier, since the Initiator does not need to use it as an anti-clogging token, and is keeping state. IKEv2 allows the Responder to initially choose a stateless anti-clogging type cookie by responding to an IKE_SA_init with a cookie request, and then upon receipt of an IKE_SA_init with a valid cookie, change his cookie value from the computed anti-clogging token to a more convenient value, by sending a different value for his cookie in the IKE_SA_init_response. This will not confuse the Initiator (Alice), because she will have chosen a unique cookie value A, so if her SA state for the partially set up IKE-SA says that Bob's cookie for the SA that Alice knows as "A" is B, and she receives a response from Bob with cookies (A,C), that means that Bob wants to change his value from B to C for the SA that Alice knows uniquely as "A".

Another reason why Bob might want to change his cookie value is that it is possible (though unlikely) that Bob will choose the same cookie for multiple SAs if the hash of the Initiator cookie, Initiator IP address, and whatever other information might be included happens to hash to the same value.

In IKEv2, like IKEv1, both 8-byte cookies appear in the message, but in IKEv2 (unlike v1), the value chosen by the message recipient always appears first in the message. This change eliminates a flaw in IKEv1, as well as having other advantages (allowing the recipient to

look up the SA based on a small, conveniently chosen value rather than a 16-byte pseudorandom value.)

The flaw in IKEv1 is that it was possible (though unlikely) for two connections to have the same set of cookies. For instance, if Alice chose A as the Initiator cookie when initiating a connection to Bob, she might subsequently receive a connection request from Carol, and Carol might also have chosen A as the Initiator cookie. Whatever value Alice responds to Carol, say B, might be selected as the Responder cookie by Bob for the Alice-Bob SA. Then Alice would be involved in two IKE sessions, both of which had Initiator cookie=A and Responder cookie=B. To minimize, but not eliminate, the probability of this happening, version 1 IKE recommended that cookies be chosen at random.

One additional rule in IKEv2 is that the two cookie values have to be different. The Responder is responsible for choosing a value different from the one chosen by the Initiator. If the Responder's stateless cookie happens to be equal to the Initiator's cookie, that is legal provided that the Responder change his cookie value to something different from the Initiator's in his IKE_SA_init_response. The reason the cookies must be different in the two directions is to prevent reflection attacks. Another way reflection attacks could have been avoided was to compute different integrity and encryption keys in the two directions, but that would be another change from IKEv1.

The cookies are one of the inputs into the function that computes the keying material. If the Responder initially sends a stateless cookie value in its IKE_SA_init_reject, and changes to a different value when it sends its IKE_SA_init_response, it is the cookie value in the IKE_SA_init_response that is the input for generating the keying material.

2.7 Cryptographic Algorithm Negotiation

The payload type known as "SA" indicates a proposal for a set of choices of protocols (e.g., IKE, ESP, AH, and/or IPcomp) for the SA as well as cryptographic algorithms associated with each protocol. In IKEv1 it was extremely complex, and required a separate proposal for each possible combination. If there were n algorithms of one type (say encryption) that were acceptable and worked with any one of m algorithms of another type (say integrity protection), then it would take space proportional to n*m to express all of the possibilities.

IKEv2 has simplified the format of the SA payload somewhat, but in addition to simplifying the format, solves the exponential explosion by allowing, within a proposal, multiple algorithms of the same type. If more than one algorithm of the same type (say encryption) appears

in a proposal, that means that the sender of that SA proposal is willing to accept the proposal with any of those choices, and the recipient when it accepts the proposal selects exactly one of each of the types of algorithms from the choices offered within that proposal.

An SA consists of one or more proposals. Each proposal has a number (so that the recipient can specify which proposal has been accepted), and contains a protocol (IKE, ESP, AH, or IPcomp), a SPI to identify the SA for ESP or AH or IPcomp, and set of transforms. Each transform consists of a type (e.g., encryption, integrity protection, authentication, Diffie-Hellman group, compression) and a transform ID (e.g., DES, IDEA, HMAC-MD5). To negotiate an SA that does ESP, IPcomp, and AH, the SA will contain three proposals with the same proposal number, one proposing ESP, a 4 byte SPI to be used with ESP, and a set of transforms; one proposing AH, a 4-byte SPI to be used with AH, and a set of transforms; and one proposing IPcomp, a 2-byte SPI to be used with IPcomp, and a set of transforms. If the recipient selects that proposal number, it means that SAs will be created for all of ESP, AH, and IPcomp.

In IKEv2, since the Initiator sends her Diffie-Hellman value in the IKE_SA_init, she must guess at the Diffie-Hellman group that Bob will select from her list of supported groups. Her guess MUST be the first in the list to allow Bob to unambiguously identify which group the accompanying KE payload is from. If her guess is incorrect then Bob's response informs her of the group he would choose, and notifies her that her offer is invalid because the KE payload is not from the desired group. In this case Alice will send a new IKE_SA_init, with the same original choices in the list (this is important to prevent an active attacker from tricking them into using a weaker group than they would have agreed upon) but with Bob's preferred group first, and a KE payload containing an exponential from that group.

If none of Alice's options are acceptable, then Bob notifies her accordingly.

2.8 Rekeying

Security associations negotiated in both phase 1 and phase 2 contain secret keys which may only be used for a limited amount of time. This determines 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 can be established. Reestablishment of security associations to take the place of ones which expire is referred to as "rekeying".

To rekey a child-SA, create a new, equivalent SA (see section 4 and

4.1 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 4 and 4.2 below) with the peer to whom the old IKE-SA is shared using a Phase 2 negotiation 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.

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 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 dithered (delayed by a random amount of time).

This form of rekeying will 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. The node that initiated the rekeying SHOULD delete the older SA after the new one is established.

2.9 Traffic Selector Negotiation

When an IP packet is received by an RFC2401 compliant IPsec subsystem and matches a "protect" selector in its SPD, the subsystem MUST protect that packet with IPsec. When no SA exists yet it is the task of IKE to create it. Information about the traffic that needs protection is transmitted to the IKE subsystem in a manner outside the scope of this document (see [PFKEY] for an example). This information is negotiated between the two IKE endpoints using TS (Traffic Selector) payloads.

The TS payload consists of a set of individual traffic selectors. The selector from the SPD has "source" and "destination" components and these are represented in IKE as a pair of TS payloads, TSi (traffic selector-initiator) and TSr (traffic selector-responder).

TSi describes the addresses and ports that the Initiator will send from over the SA and which it will accept packets for. TSr describes the addresses and ports that the Initiator will sent to over the SA and which it will accept packets from.

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

Note that the traffic selectors apply to both child-SAs (from the Initiator to the Responder and from the Responder to the Initiator), but the Responder does not change the order of the TS payloads. An address within the selector of TSi would appear as a source address in the child-SA from the Initiator, and would appear as a destination address in traffic on the child-SA to the Initiator (from the Responder).

IKEv2 is more flexible than IKEv1. IKEv2 allows sets of ranges of both addresses and ports, and allows the Responder to choose a subset of the requested traffic rather than simply responding "not acceptable".

2.10 Nonces

The IKE_SA_init_request and the IKE_SA_init_response each contain a nonce. These nonces are used as inputs to cryptographic functions. The child-create-request and the child-create-response also contain a nonce. These nonces are used to add freshness to the key derivation technique used to obtain keys for child SAs. Nonces used in IKEv2 MUST therefore have strong pseudo-random properties (see RFC1715).

3 The Phase 1 Exchange

The base Phase 1 exchange is a four message exchange (two request/response pairs). The first pair of messages, the IKE_SA_init exchange, negotiate cryptographic algorithms, indicate trusted CA names, exchange nonces, and do a Diffie-Hellman exchange. This pair might be repeated if the response indicates that none of the cryptographic proposals are acceptable, or the Diffie-Hellman group chosen by the Initiator for sending her Diffie-Hellman value is not the group that the Responder would have chosen, of if the Responder is under attack and will only answer IKE_SA_init requests containing a valid returned cookie value.

The second pair of messages, the IKE_auth and the IKE_auth_response, authenticate the previous messages, exchange identities and certificates, and optionally also establish a child_SA. This pair of

messages is encrypted with a key established through the IKE_SA_init exchange, so the identities are hidden from eavesdroppers.

In the following description, the payloads contained in the message are indicated by names such as SA. The details of the contents of each payload are described later. Payloads which may optionally appear will be shown in brackets, such as [CERTREQ], would indicate that optionally a certificate request payload can be included. The certificate request payload indicates a CA name trusted by the sender. If the sender trusts multiple CAs, it includes multiple CERTREQ payloads, one for each trusted CA.

The Phase 1 exchange is as follows:

```
    Initiator                          Responder
   -----------                        -----------
    HDR, SA, KE, Ni [,CERTREQ]  -->
```

The SA payload states the cryptographic algorithms the Initiator supports. The KE payload sends the Initiator's Diffie-Hellman value. Ni is the Initiator's nonce, sent in an N payload.

```
                                <--    HDR, SA, KE, Nr [,CERTREQ]
```

The Responder chooses among the Initiator's cryptographic algorithms and expresses that choice in the SA payload, completes the Diffie-Hellman exchange with the KE payload, and sends its nonce in the N payload (with an "r" to signify the Responder's nonce).

At this point in time each party generates SKEYSEED and its derivatives. The following two messages, the SA_auth and SA_auth_response, are encrypted (as indicated by the '*' following the IKE header) and the encryption bit in the IKE header is set.

```
    HDR*, ID, AUTH, [CERT,]
          SA, TSi, TSr         -->
```

The Initiator identifies herself with the ID payload, authenticates herself to the Responder with the AUTH payload, optionally sends one or more certificates, and begins negotiation of a child-SA using the SA payload and the Traffic Selector payloads: TSi (which describes sources of packets to be sent over the child-SA), and TSr (which describes destinations of packets to be sent over the child-SA). The protocol (ESP, AH, and/or IPcomp) and the SPI she wants to use to identify her inbound child-SA are placed in the "protocol" and "SPI" fields, respectively, in the SA payload.

```
                                <--    HDR*, ID, AUTH, [CERT,]
                                             SA, TSi, TSr
```

The Responder identifies himself with an ID payload authenticates himself with the AUTH payload, optionally sends one or more certificates, and completes negotiation of a child-SA using the SA payload. The Responder places the SPI he wants to use to identify his inbound child-SA in the SA payload. The TSi and TSr, respectively, describe the sources and destinations of packets to be sent over the child-SA. These MUST be equal to, or a subset of, the ones suggested by the Initiator.

3.1 Generating Keying Material for the IKE-SA

The shared secret information is 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 three other secrets: SKEYSEED_d used for deriving new keys for the child-SAs established with this IKE-SA; SKEYSEED_a used for authenticating the component messages of subsequent exchanges; and SKEYSEED_e used for encrypting (and of course decrypting) all subsequent exchanges. SKEYSEED and its derivatives are computed as follows:

```
SKEYSEED = prf(Ni | Nr, g^ir)
SKEYSEED_d = prf(SKEYSEED, g^ir | CKY-I | CKY-R | 0)
SKEYSEED_a = prf(SKEYSEED, SKEYSEED_d | g^ir | CKY-I | CKY-R | 1)
SKEYSEED_e = prf(SKEYSEED, SKEYSEED_a | g^ir | CKY-I | CKY-R | 2)
```

CKY-I and CKY-R are the Initiator's and Responder's cookie, respectively, from the IKE header. g^ir is the shared secret from the ephemeral Diffie-Hellman exchange. Ni and Nr are the nonces, stripped of any headers. 0, 1, and 2 are represented by a single byte containing the value 0, 1, or 2 (the values, not the ASCII representation of the digits). prf is the "pseudo-random" cryptographic function negotiated in the IKE-SA-init exchange. The pseudo-random functions defined for IKE are HMAC_MD5 and HMAC_SHA, defined in RFC 2104.

3.2 Authentication of the IKE-SA

The peers are authenticated by having each sign the concatenation of the first two messages of the exchange. Optionally, they MAY include a certificate or certificate chain providing evidence that the public key they are using belongs to the name in the ID payload. The public key signature will be computed using algorithms chosen by the signer, most commonly an RSA-signed PKCS1-padded-SHA1-hash of the

concatenated messages or a DSS-signed SHA1-hash of the concatenated messages. 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 public key each has. This type is either indicated in the certificate supplied or, if the public keys were exchanged out of band, the key types must have been similarly learned.

4 The CREATE-CHILD-SA (Phase 2) Exchange

A phase 2 exchange is one request/response pair, and can be used to create or delete a child-SA, delete the IKE-SA, or deliver information such as error conditions. It is encrypted and integrity protected using the keys negotiated during the creation of the IKE-SA. The two directions of flow use the same keys.

Messages are cryptographically protected using the cryptographic algorithms and keys negotiated in the first two messages of the IKE exchange using a syntax based on the encoding in ESP (see Appendix B). Encryption uses a key derived from SKEYSEED_e; Integrity uses a key derived from SKEYSEED_a.

Either side may initiate a phase 2 exchange. A child-SA is created by sending a CREATE_CHILD_SA request. If PFS for the child-SA is desired, the CREAT_CHILD_SA request contains KE payloads for an additional Diffie-Hellman exchange. The keying material for the child_SA is a function of SKEYSEED_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 the Diffie-Hellman group for the child-SA is desired to be different from the group for the IKE-SA, then a Diffie-Hellman group transform MUST be included in the SA payload. If it is absent, the Diffie-Hellman group is assumed to be the same as the one in the IKE-SA.

The CREATE_CHILD_SA request contains:

```
    Initiator                                  Responder
   -----------                                -----------
    HDR*, SA, Ni, [KEi,]
          TSi, TSr              -->
```

The Initiator sends SA offer(s) in the SA payload(s), a nonce in the Ni payload, optionally a Diffie-Hellman value in the KE payload, and the proposed traffic selectors in the TSi and TSr payloads. The message past the header is encrypted and the message including the header is integrity protected using the cryptographic algorithms negotiated in Phase 1.

The CREATE_CHILD_SA response contains:

```
                              <--    HDR*, SA, Nr, [KEr,]
                                     TSi, TSr
```

The Responder replies (using the same Message ID to respond) with the accepted offer in an SA payload, optionally a Diffie-Hellman value in the KE payload, and the traffic selectors for traffic to be sent on that SA in the TS payloads, which may be a subset of what the Initiator of the child-SA proposed.

4.1 Generating Keying Material for IPsec SAs

Child-SAs are created either by being piggybacked on the phase 1 exchange, or in a phase 2 CREATE_CHILD_SA exchange. Keying material for them is generated as follows:

```
KEYMAT = prf(SKEYSEED_d, protocol | SPId | Ns | Nd )
```

For phase 2 exchanges with PFS the keying material is defined as:

```
KEYMAT = prf(SKEYSEED_d, g(p2)^ir | protocol | SPId | Ns | Nd )
```

where g(p2)^ir is the shared secret from the ephemeral Diffie-Hellman exchange of this phase 2 exchange.

In either case, "protocol", and "SPI", are from the SA payload that contained the negotiated (and accepted) proposal, Ns is the body of the Source's nonce payload (minus the generic header), and Nr is the body of the Destination's nonce payload (minus the generic header).

A single child-SA negotiation results in two security associations--one inbound and one outbound. Different Nonces and SPIs for each SA (one chosen by the Initiator, the other by the Responder) guarantee a different key for each direction. The SPI chosen by the destination of the SA and the Nonces (ordered source followed by destination) are used to derive KEYMAT for that SA.

For situations where the amount of keying material desired is greater than that supplied by the prf, KEYMAT is expanded by feeding the results of the prf back into itself and concatenating results until the required keying material has been reached. In other words,

```
KEYMAT = K1 | K2 | K3 | ...
where:
 K1 = prf(SKEYSEED_d, [ g(p2)^ir | ] protocol | SPId | Ns | Nd)
 K2 = prf(SKEYSEED_d, K1 | [ g(p2)^ir | ] protocol | SPId | Ns | Nd)
 K3 = prf(SKEYSEED_d, K2 | [ g(p2)^ir | ] protocol | SPId | Ns | Nd)
 etc.
```

This keying material (whether with PFS or without) MUST be used with the negotiated SA. In the case of an ESP SA needing two keys for encryption and authentication, the encryption key is taken from the first bytes of KEYMAT and the authentication key is taken from the next bytes.

4.2 Generating Keying Material for IKE-SAs from a create-child exchange

The create-child exchange can be used to re-key an existing IKE-SA (see section 2.8). When used for this purpose the create-child exchange MUST be done with the PFS option. New Initiator and Responder cookies are supplied in the SPI fields. The TS payloads are omitted when rekeying an IKE-SA. SKEYSEED for the new IKE-SA is computed using SKEYSEED_d from the existing IKE-SA as follows:

```
    SKEYSEED = prf(SKEYSEED_d (old), g(p2)^ir | 0 | CKY-I | CKY-R
                                              | Ni | Nr)
```

where g(p2)^ir is the shared secret from the ephemeral Diffie-Hellman exchange of this phase 2 exchange, CKY-I is the 8-byte "SPI" from the SA payload in the CREATE_CHILD_SA request, CKY-R is the 8-byte "SPI" from the SA payload in the CREATE_CHILD_SA response, and Ni and Nr are the two nonces stripped of any headers. "0" is a single byte containing the value zero (the protocol ID of IKE).

SKEYSEED_d, SKEYSEED_a, and SKEYSEED_e are computed from SKEYSEED as specified in section 3.1.

5 Informational (Phase 2) Exchange

At various points during 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 a (reliable) Informational exchange. Usually Informational exchanges happen during phase 2 and are cryptographically protected with the IKE exchange.

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.

There are two cases in which there is no IKE-SA to protect the information. One is in the response to an IKE_SA_init_request to request a cookie or to refuse the SA proposal. This would be conveyed in a Notify payload of the IKE_SA_init_response.

The other case in which there is no IKE-SA to protect the information is when a packet is received with an unknown SPI. In that case the notification of this condition will be sent in an informational exchange that is cryptographically unprotected.

Messages in an Informational Exchange contain zero or more Notification or Delete 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 can be a message with no payloads. Actually, the request message in an Informational Exchange can also contain no payloads. This is the expected way an endpoint can ask the other endpoint to verify that it is alive.

ESP, AH, and IPcomp SAs always exist in pairs, with one SA in each direction. When an SA is closed, both members of the pair MUST be closed. When SAs are nested, as when data is encapsulated first with IPcomp, then with ESP, and finally with AH between the same pair of endpoints, all of the SAs (up to six) must be deleted together. To delete an SA, an Informational Exchange with one or more delete payloads is sent listing the SPIs (as known to the recipient) of the SAs to be deleted. The recipient MUST close the designated SAs. 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 that it has already issued a delete request for, it MUST delete the incoming SAs while processing the request and the outgoing 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.

A node SHOULD regard half open connections as anomalous and audit their existence should 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 open 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 which will implicitly close all SAs negotiated under it. It can then rebuild the SA's it needs on a clean base under a new IKE-SA.

The Informational Exchange is defined as:

```
    Initiator                          Responder
   -----------                        -----------
    HDR*, N, ..., D, ...    -->
                            <--        HDR*, N, ..., D, ...
```

The processing of an Informational Exchange is determined by its component payloads.

6 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 non-existent 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 outside the context of an IKE-SA (and not a request to start one), it may be the result of a recent crash. 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 from whence it came with the IKE cookies reversed in the header and the Message ID copied. The response MUST NOT be cryptographically protected and MUST contain a notify payload indicating the nature of the problem.

A node receiving such a message 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. 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. The recipient MUST NOT change the state of any SA's as a result but SHOULD 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.

7 Header and Payload Formats

7.1 The IKE Header

IKE messages use UDP port 500, with one IKE message per UDP datagram. Information from the UDP header is largely ignored except that the IP addresses from the headers are reversed and used for return packets. 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.

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.

The format of the IKE header is shown in Figure 1.

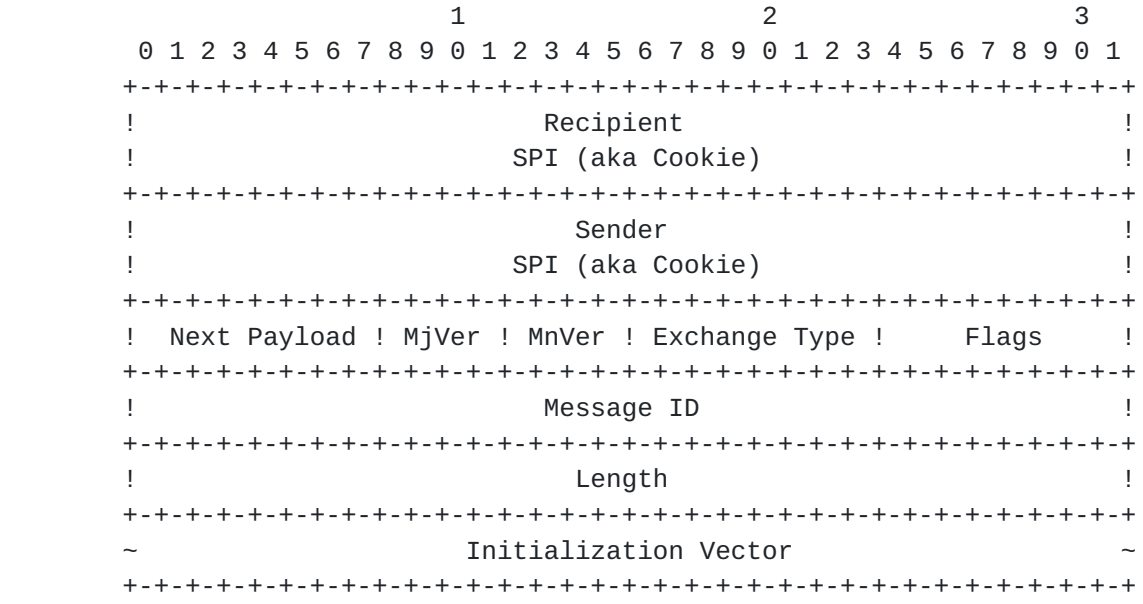

Figure 1: IKE Header Format

- Recipient SPI (aka Cookie) (8 bytes) - A value chosen by the recipient to identify a unique IKE security association. [NOTE: this is a deviation from ISAKMP and IKEv1, where the cookies were always sent with the Initiator of the IKE-SA's cookie first and the Responder's second. See section 2.6.]

- Sender SPI (aka Cookie) (8 bytes) - A value chosen by the sender to identify a unique IKE security association.

- Next Payload (1 byte) - Indicates the type of payload that immediately follows the header. The format and value of each payload is 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 byte) - indicates the type of exchange being

used. This dictates the payloads sent in each message and message orderings in the exchanges.

Exchange Type	Value
RESERVED	0
Reserved for ISAKMP	1 - 31
Reserved for IKEv1	32 - 33
Phase One	34
CREATE-CHILD-SA	35
Informational	36
Reserved for IKEv2+	37-239
Reserved for private use	240-255

o Flags (1 byte) - 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 byte.

-- E(ncryption) (bit 0 of Flags) - If set, all payloads following the header are encrypted and integrity protected using the algorithms negotiated during session establishment and a key derived during the key exchange portion of IKE. If not set, the payloads are not protected. All payloads MUST be protected if a key has been negotiated and any unprotected payload may only be used to establish a new session or indicate a problem.

-- C(ommit) (bit 1 of Flags) - This bit is defined by ISAKMP but not used by IKEv2. Implementations of IKEv2 MUST clear this bit when sending and SHOULD ignore it in incoming messages.

-- A(uthentication Only) (bit 2 of Flags) - This bit is defined by ISAKMP but not used by IKEv2. Implementations of IKEv2 MUST clear this bit when sending and SHOULD ignore it in incoming messages.

-- I(nitiator) (bit 3 of Flags) - This bit MUST be set in messages sent by the Initiator of an exchange and MUST be cleared in messages sent by the Responder. It is used by the recipient to determine whether the message number should be interpreted in the context of its initiating state or its responding state.

-- 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.

-- R(eserved) (bits 5-7 of Flags) - These bit MUST be cleared in messages sent and received messages with these bits set MUST be rejected.

o Message ID (4 bytes) - Message identifier used to control retransmission of lost packets and matching of requests and responses. See section 2.2. In the first message of a Phase 1 negotiation, the value MUST be set to 0. The response to that message MUST also have a Message ID of 0.

o Length (4 bytes) - Length of total message (header + payloads) in bytes. Session encryption can expand the size of an IKE message and that is reflected in the total length of the message.

o Initialization Vector (variable) - random bytes used to provide initialization to an encryption mode-- e.g. cipher block chaining (CBC) mode. This field MUST be present when the encryption bit is set in the flags field (see below) and MUST NOT be present otherwise. The length of the Initialization Vector is cipher and mode dependent.

7.2 Generic Payload Header

Each IKE payload defined in sections 7.3 through 7.13 begins with a generic header, shown in Figure 2. Figures for each payload below will include the generic payload header but for brevity a repeat of the description of each field will be omitted. The construction and processing of the generic payload header is identical for each payload and will similarly be omitted.

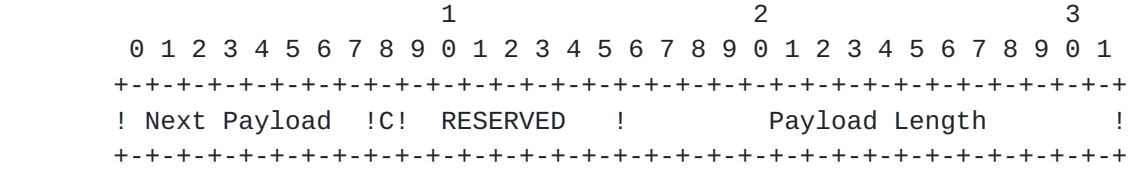

Figure 2: Generic Payload Header

The Generic Payload Header fields are defined as follows:

o Next Payload (1 byte) - 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.

- Critical (1 bit) - MUST be set to zero if the sender wants the recipient to skip this payload if he does not understand the payload type code. MUST be set to one if the sender wants the recipient to reject this entire message if he does not understand this payload type. MUST be ignored by 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 byte.

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

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

7.3 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 may contain multiple proposals. Each proposal may contain multiple protocols (where a protocol is IKE, ESP, AH, or IPCOMP), 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, while 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 a Protocol-id. Each structure MUST have the same Proposal # as the previous one or one greater. The first Proposal MUST have a Proposal # of one. 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 five transforms: a Diffie-Hellman group, an authentication algorithm, an integrity check algorithm, a PRF algorithm, and an encryption algorithm. For each Protocol, the set of permissible transforms are 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 - say (3DES and HMAC-MD5) or (IDEA and HMAC-SHA), there is no way to encode that as multiple transforms within a single Proposal/Protocol. 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.

Note that the semantics of Transforms and Attributes are quite different than 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.

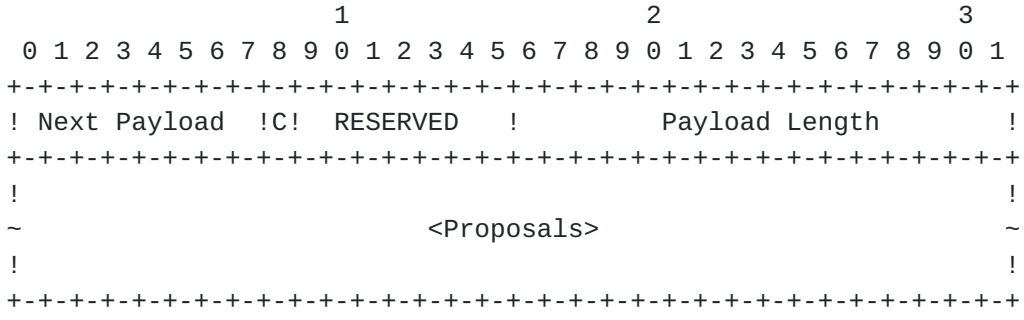

Figure 3: Security Association Payload

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

The payload type for the Security Association Payload is one (1).

7.3.1 Proposal Substructure

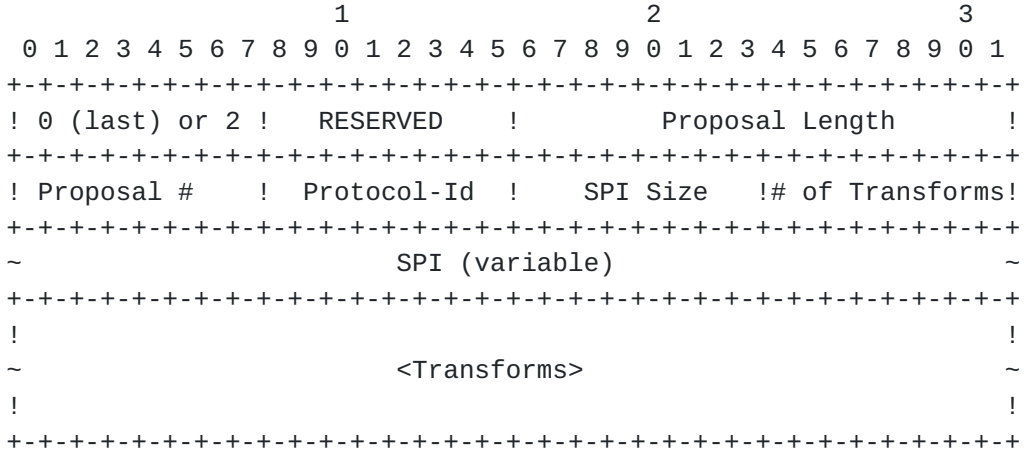

Figure 4: Proposal Substructure

o 0 (last) or 2 (more) (1 byte) - 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, and the first four bytes of the Proposal structure are designed to look somewhat like the header of a Payload.

o RESERVED (1 byte) - MUST be sent as zero; MUST be ignored.

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

- Proposal # (1 byte) - When a proposal is made, the first proposal in an SA 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 must be the same and must match the number on the proposal sent that was accepted.

- Protocol-Id (1 byte) - Specifies the protocol identifier for the current negotiation. During phase 1 negotiation this field MUST be zero (0). During phase 2 it will be the protocol of the SA being established as assigned by IANA, for example, 50 for ESP, 51 for AH, and 108 for IPComp.

- SPI Size (1 byte) - During phase 1 negotiation this field MUST be zero. During phase 2 negotiation it is equal to the size, in bytes, of the SPI of the corresponding protocol (4 for ESP and AH, 2 for IPcomp).

- # of Transforms (1 byte) - 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 bytes, 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. This case occurs when negotiating the IKE-SA.

- Proposal # (1 byte) - Identifies the immediate proposal. The first proposal has the number of one (1) and each subsequent proposal has a number which is one greater than the last.

- Proposal Length (2 bytes) - Length in bytes of the proposal including all SA Attributes.

- SA Attributes (variable length) - This field contains SA attributes for the immediate transform. The SA Attributes MUST be represented using the Transform Attributes format described below.

- RESERVED (1 byte) - MUST be sent as zero; MUST be ignored.

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

7.3.2 Transform Substructure

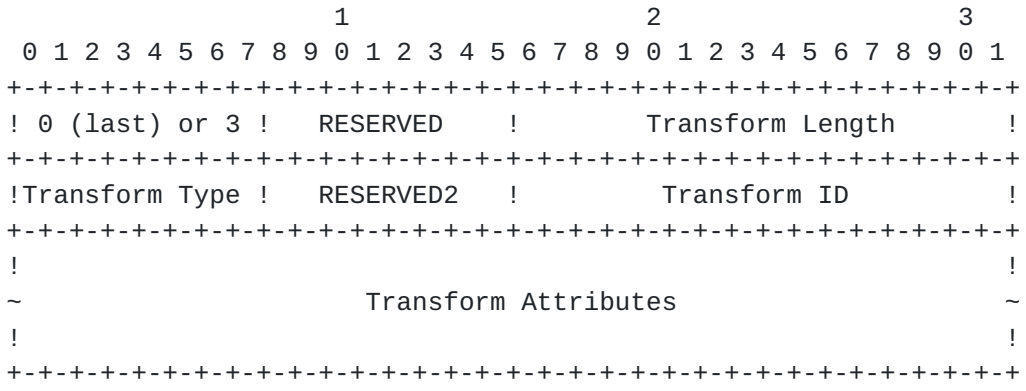

Figure 5: Transform Substructure

- 0 (last) or 3 (more) (1 byte) - 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, and the first four bytes of the Transform structure are designed to look somewhat like the header of a Payload.

- RESERVED (1 byte) - MUST be sent as zero; MUST be ignored.

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

- Transform Type (1 byte) - 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.

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

Transform Type Values

	Transform Type	Used In
Encryption Algorithm	1	(IKE and ESP)
Pseudo-random Function	2	(IKE)
Authentication Method	3	(IKE)
Integrity Algorithm	4	(IKE, AH, and optional in ESP)
Diffie-Hellman Group	5	(IKE and optional in AH and ESP)
Compression	6	(IPcomp)
Window Size	7	(IKE)

values 8-240 are reserved to IANA. Values 241-255 are for private use among mutually consenting parties.

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)
ENCR_3DES	3	(RFC2451)
ENCR_RC5	4	(RFC2451)
ENCR_IDEA	5	(RFC2451)
ENCR_CAST	6	(RFC2451)
ENCR_BLOWFISH	7	(RFC2451)
ENCR_3IDEA	8	(RFC2451)
ENCR_DES_IV32	9	
ENCR_RC4	10	
ENCR_NULL	11	(RFC2410)
ENCR_AES	12	

values 12-240 are reserved to IANA. Values 241-255 are for private use among mutually consenting parties.

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

Name	Number	Defined In
RESERVED	0	
PRF_HMAC_MD5	1	(RFC2104)
PRF_HMAC_SHA	2	(RFC2104)
PRF_HMAC_TIGER	3	(RFC2104)

values 3-240 are reserved to IANA. Values 241-255 are for private use among mutually consenting parties.

For Transform Type 3 (Authentication Method), defined Transform-IDs are:

Name	Number	Defined In
RESERVED	0	
Methods in IKEv1	1 - 5	(RFC2409)
Authenticated Diffie-Hellman	6	(this memo)

values 7-240 are reserved to IANA. Values 241-255 are for private use among mutually consenting parties.

For Transform Type 4 (Integrity Algorithm), defined Transform-IDs are:

Name	Number	Defined In
RESERVED	0	
AUTH_HMAC_MD5	1	(RFC2403)
AUTH_HMAC_SHA	2	(RFC2404)
AUTH_DES_MAC	3	
AUTH_KPDK_MD5	4	(RFC1826)

For Transform Type 5 (Diffie-Hellman Group), defined Transform-IDs are:

Name	Number	
RESERVED	0	
Pre-defined (see section 8)	1 - 5	
RESERVED	6 - 200	
MODP (exponentiation)	201	(w/attributes)
ECP (elliptic curve over GF[P]	202	(w/attributes)
EC2N (elliptic curve over GF[2^N])	203	(w/attributes)

values 6-200 are reserved to IANA for new MODP, ECP or EC2N groups. Values 204-255 are for private use among mutually consenting parties. Specification of values 201, 202 or 203 allow peers to define a new Diffie-Hellman group in-line as part of the exchange. Private use of values 204-255 may entail complete definition of a group or may require attributes to accompany them. Attributes MUST NOT accompany groups using values between 6 and 200.

For Transform Type 6 (Compression), defined Transform-IDs are:

Name	Number	Defined In
RESERVED	0	
IPCOMP_OUI	1 (w/attributes)	
IPCOMP_DEFLATE (RFC2394)	2	
IPCOMP_LZS (RFC2395)	3	

values 4-240 are reserved to IANA. Values 241-255 are for private use among mutually consenting parties.

For Transform Type 7 (Window Size), the Transform-ID specifies the window size a peer is contracting to support to handle overlapping requests (see section 2.3).

7.3.3 Mandatory Transform Types

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 support all mandatory and optional types for each protocol it supports. Whether the optional types are present in a particular proposal depends solely on the discretion of the sender.

Protocol	Mandatory Types	Optional Types
IKE	1, 2, 3, 5, 7	
ESP	1	4, 5
AH	4	5
IPCOMP	6	

7.3.4 Mandatory Transform-IDs

Each transform type has corresponding transform IDs to specify the specific transform. Some transforms are mandatory to support and others are optional to support. The mandatory transform IDs for AH, ESP, and IPCOMP are left to their respective RFCs, RFC2402, RFC2406, and RFC2393. The transform IDs that are mandatory to support for IKEv2 are:

Name	TransType	Mandatory Transform-ID
Encryption Algorithm	1	12 (ENCR_AES)
Pseudo-Random Function	2	2 (PRF_HMAC_SHA)
Authentication Method	3	6 (signed D-H)
Diffie-Hellman Group	5	5 (1536 bit MODP)
Window Size	7	1

All other transform-IDs for a given transform type are optional to support. While implementations MUST have a window size of at least 1 they SHOULD support a window size of at least 10 and MAY support larger window sizes.

7.3.4 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 in Appendix A. 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 byte length or a variable length value. For the latter the attribute is the form of type/length/value.

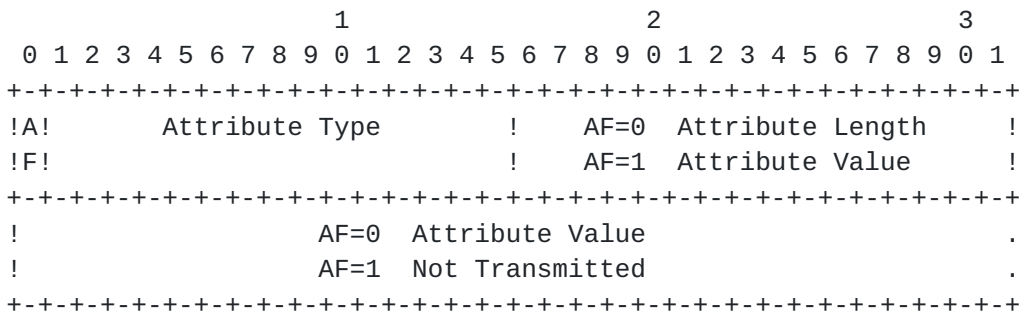

Figure 6: Data Attributes

- Attribute Type (2 bytes) - Unique identifier for each type of attribute. The identifiers for IKE are defined in Appendix A.

  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 bytes) - Length in bytes of the Attribute Value. When the AF bit is a one (1), the Attribute Value is only 2 bytes 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 bytes.

7.3.5 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. Diffie-Hellman groups are specified either using a defined group description (section 5) or by defining all attributes of a group (see Appendix A) in an IKE policy offer. Group attributes, such as group type or prime number MUST NOT be offered in conjunction with a previously defined group. SA offers include proposed attributes and a Diffie-Hellman public number (KE) in the same message. If 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 start over this time using a different group (see section 2.7).

Implementation Note:

> Certain negotiable attributes can have ranges or could have multiple acceptable values. These are the Diffie-Hellman group and 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 which they deem to supply greater security. For instance if a peer is configured to accept a variable lengthed cipher with a key length of X bits and is offered that cipher with a larger key length an implementation SHOULD accept the offer.
>
> 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 foo".

7.4 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 header followed by the Diffie-Hellman public value itself.

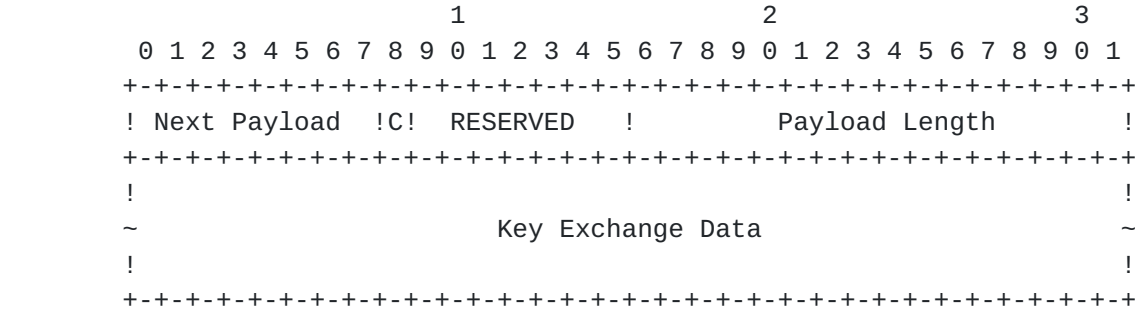

Figure 7: Key Exchange Payload Format

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.

A key exchange payload is processed by first checking whether the length of the key exchange data (the "Payload Length" from the generic header minus the size of the generic header) is equal to the length of the prime modulus over which the exponentiation was performed.

The payload type for the Key Exchange payload is four (4).

7.5 Identification Payload

The Identification Payload, denoted ID in this memo, allows peers to identify themselves to each other. In Phase 1, the ID Payload names the identity to be authenticated with the signature. In Phase 2, the ID Payload is optional and if present names an identity asserted to be responsible for this SA. An example use would be a shared computer opening an IKE-SA to a server and asserting the name of its logged in user for the Phase 2 SA. If missing, this defaults to the Phase 1 identity.

NOTE: In IKEv1, two ID payloads were used in each direction in Phase 2 to hold Traffic Selector information for data passing over the SA. In IKEv2, this information is carried in Traffic Selector (TS) payloads (see section 7.13).

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

Figure 8: Identification Payload Format

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

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

- 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 type for the Identification Payload is five (5).

The following table lists the assigned values for the Identification Type field, followed by a description of the Identification Data which follows:

ID Type	Value
RESERVED	0
ID_IPV4_ADDR	1

A single four (4) byte IPv4 address.

ID_FQDN	2

A fully-qualified domain name string. An example of a ID_FQDN is, "lounge.org". The string MUST not contain any terminators (e.g. NULL, CR, etc.).

ID_USER_FQDN	3

A fully-qualified username string, An example of a ID_USER_FQDN is, "lizard@lounge.org". The string MUST not contain any terminators.

ID_IPV6_ADDR 5

A single sixteen (16) byte IPv6 address.

ID_DER_ASN1_DN 9

The binary 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 byte stream which may be used to pass vendor-specific information necessary to do certain proprietary forms of identification.

7.6 Certificate Payload

The Certificate Payload, denoted CERT in this memo, provides a means to transport certificates or other certificate-related information via IKE. Certificate payloads SHOULD be included in an exchange if certificates are available to the sender.

The Certificate Payload is defined as follows:

Figure 9: Certificate Payload Format

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

Certificate Encoding	Value
NONE	0
PKCS #7 wrapped X.509 certificate	1
PGP Certificate	2
DNS Signed Key	3
X.509 Certificate - Signature	4
X.509 Certificate - Key Exchange	5
Kerberos Tokens	6
Certificate Revocation List (CRL)	7
Authority Revocation List (ARL)	8
SPKI Certificate	9
X.509 Certificate - Attribute	10
RESERVED	11 - 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 six (6).

7.7 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 first, second, or third message of Phase 1. Certificate Request payloads SHOULD be included in an exchange whenever the peer may have multiple certificates, some of which might be trusted while others are not. If multiple root CA's are trusted, then multiple Certificate Request payloads SHOULD be transmitted.

The Certificate Payload is defined as follows:

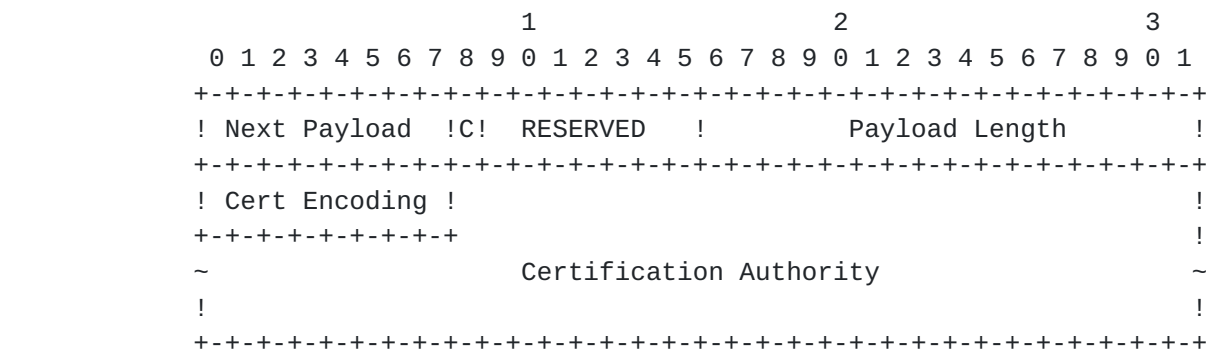

Figure 10: Certificate Request Payload Format

- Certificate Encoding (1 byte) - Contains an encoding of the type of certificate requested. Acceptable values are listed in

section 7.6.

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

The payload type for the Certificate Request Payload is seven (7).

The Certificate Request Payload is constructed by setting the "Cert Encoding" field to be the type of certificate being desired and the "Certification Authority" field to a proper encoding of a certification authority for the specified certificate. For example, for an X.509 certificate this field would contain the Distinguished Name encoding of the Issuer Name of an X.509 certification authority acceptable to the sender of this payload.

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 which can be validated up to the specified certification authority. This can be a chain of certificates. If a certificate exists which satisfies the criteria specified in the Certificate Request Payload it MUST be sent back to the certificate requestor; if a certificate chain exists which goes back to the certification authority specified in the request the entire chain MUST be sent back to the certificate requestor. If no certificates exist then no further processing is performed-- this is not an error condition of the protocol.

7.8 Authentication Payload

The Authentication Payload, denoted AUTH in this memo, contains data used for authentication purposes. The only authentication method defined in this memo is digital signatures and therefore the contents of this payload when used with this memo will be the output generated by a digital signature function.

The Authentication Payload is defined as follows:

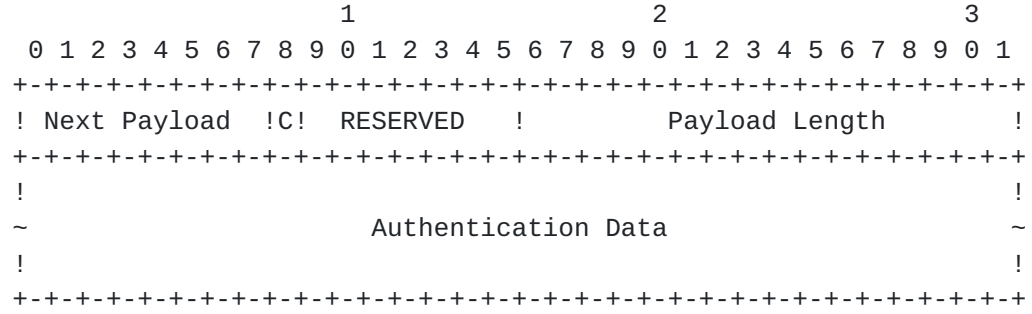

Figure 11: Authentication Payload Format

o Authentication Data (variable length) - Data that results from applying the digital signature function to the IKE state (see section 3).

The payload type for the Authentication Payload is nine (9).

The Authentication Payload is constructed by computing a digital signature over the concatentation of the two IKE messages in the initial unprotected IKE-SA-INIT exchange and placing the result in the "Authentication Data" portion of the payload. The signature MUST be a PKCS#1 encoded signature using the cryptographic hash and signature algorithms chosen by the signer. The algorithms used by the two ends MAY be different. The payload length is the size of the generic header plus the size of the "Authentication Data" portion of the payload which depends on the specific digital signature algorithm being used.

The Authentication Payload is processed by extracting the "Authentication Data" from the payload and verifying it according to the specific digital signature being used. If authentication fails a NOTIFY Error message of AUTHENTICATION-FAILED MUST be sent back to the peer and the connection closed.

7.9 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:

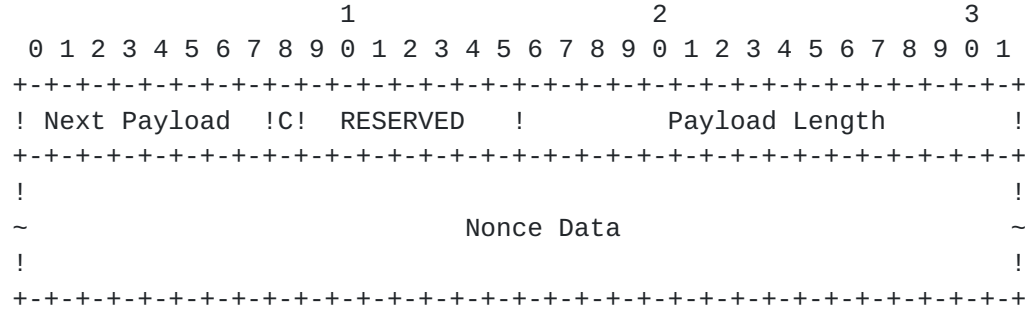

Figure 12: Nonce Payload Format

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

The payload type for the Nonce Payload is ten (10).

The Nonce Payload is constructed by computing a pseudo-random value and copying it into the "Nonce Data" field. The size of a Nonce in this memo must be between eight (8) and two-hundred fifty-six (256) bytes inclusive.

7.10 Notify Payload

The Notify Payload, denoted NOTIFY in this memo, 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), or in an Informational Exchange (to report an error not in an IKE request).

The Notify Payload is defined as follows:

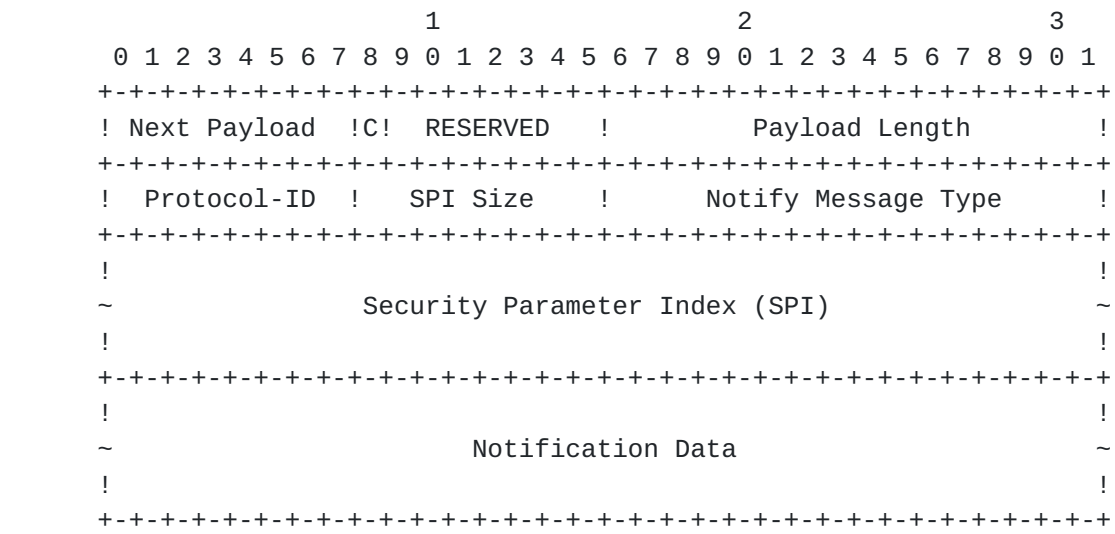

Figure 13: Notification Payload Format

- Protocol-Id (1 byte) - Specifies the protocol about which this notification is being sent. For phase 1 notifications, this field MUST be zero (0). For phase 2 notifications concerning IPsec SAs this field will contain an IPsec protocol (either ESP, AH, or IPcomp). For notifications for which no protocol ID is relevant, this field MUST be sent as zero and MUST be ignored.

- SPI Size (1 byte) - Length in bytes of the SPI as defined by the Protocol-Id or zero if no SPI is applicable. For phase 1 notification concerning the IKE-SA, the SPI Size MUST be zero.

- Notify Message Type (2 bytes) - 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 message specific, see below.

The payload type for the Notification Payload is eleven (11).

7.10.1 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. For example, a secure front end or security gateway may use the Notify message to synchronize SA communication. The table below lists the Notification messages and their corresponding values.

NOTIFY MESSAGES - ERROR TYPES	Value
INVALID-PAYLOAD-TYPE	1
Only sent if the payload has the "critical" bit set. Notification Data contains the one byte payload type.	
INVALID-COOKIE	4
Indicates an IKE message was received with an unrecognized destination cookie. This usually indicates that the recipient has rebooted and forgotten the existence of an IKE-SA.	
INVALID-MAJOR-VERSION	5
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.	
INVALID-EXCHANGE-TYPE	7
Notification Data contains the one byte Exchange Type.	
INVALID-FLAGS	8
Notification Data contains one byte with the unacceptable flag bits set.	

INVALID-MESSAGE-ID 9

Sent when either an IKE MESSAGE-ID more that ten greater than the highest acknowledged MESSAGE-ID. 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 byte invalid MESSAGE-ID.

INVALID-PROTOCOL-ID 10

Notification Data contains the one byte invalid protocol ID.

INVALID-SPI 11

MAY be sent in an IKE Informational Exchange when a node receives an ESP or AH packet with an invalid SPI. address as the source address in the invalid packet. This usually indicates a node has rebooted and forgotten an SA. This Informational Message is sent outside the context of an IKE-SA, and therefore should only be used by the recipient as a "hint" that something might be wrong (because it could easily be forged).

INVALID-TRANSFORM-ID 12

Notification Data contains the one byte invalid transform ID.

ATTRIBUTES-NOT-SUPPORTED 13

The "Notification Data" for this type are the attribute or attributes that are not supported.

NO-PROPOSAL-CHOSEN 14

BAD-PROPOSAL-SYNTAX 15

PAYLOAD-MALFORMED 16

INVALID-KEY-INFORMATION 17

The KE field is the wrong length. This can occur where there is no error if the Initiator guesses incorrectly which Diffie-Hellman group the Responder will accept. Notification data contains the Transform Substructure describing the chosen Diffie-Hellman group.

INVALID-ID-INFORMATION 18

INVALID-CERT-ENCODING 19

The "Notification Data" for this type are the "Cert Encoding" field from a Certificate Payload or Certificate Request Payload.

INVALID-CERTIFICATE 20

The "Notification Data" for this type are the "Certificate Data" field from a Certificate Payload.

CERT-TYPE-UNSUPPORTED 21

This is identical to the INVALID-CERT-ENCODING error.

INVALID-CERT-AUTHORITY 22

The "Notification Data" for this type are the "Cert Encoding" field from a Certificate Payload or Certificate Request Payload.

AUTHENTICATION-FAILED 24

INVALID-SIGNATURE 25

ADDRESS-NOTIFICATION 26

Don't understand.

UNSUPPORTED-EXCHANGE-TYPE 29

The "Notification Data" for this type are the Exchange Type field from the IKE header.

UNEQUAL-PAYLOAD-LENGTHS 30

The "Notification Data" for this type are the entire message in which the unequal lengths were observed. Receipt of this notify MAY be logged for debugging purposes.

UNSUPPORTED-NOTIFY-TYPE 31

The "Notification Data" for this type is the two byte Notify Type that was not supported.

IKE-SA-INIT-REJECT 32

This notification is sent in an IKE-SA-RESPONSE to request that the Initiator retry the request with the supplied cookie (and optionally the supplied Diffie-Hellman group). This is not really an error, but is processed like one in that it indicates that the connection request was rejected. The Notification Data, if present, contains the Transform Substructure describing the preferred Diffie-Hellman group.

INVALID-KE-PAYLOAD 33

This error indicates that the KE payload does not match the chosen Diffie-Hellman group. It can occur legitimately in either Phase 1 or Phase 2 if the Initiator supports multiple Diffie-Hellman groups and incorrectly anticipates which one the Responder will choose.

SINGLE-PAIR-REQUIRED 34

This error indicates that a Phase 2 SA request is unacceptable because the Responder requires a separate SA for each source / destination address pair. The Initiator is expected to respond by requesting an SA for only the specific traffic he is trying to forward.

RESERVED - Errors 34 - 8191

Private Use - Errors 8192 - 16383

NOTIFY MESSAGES - STATUS TYPES	Value
RESERVED	16384 - 24577

INITIAL-CONTACT 24578

This notification indicates 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-SA's 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).

RESERVED	24578 - 40959
Private Use - STATUS	40960 - 65535

7.11 Delete Payload

The Delete Payload, denoted DEL 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 14 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 with 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 0 (IKE) but no SPIs. Deletion which is concerned with a Child-SA, such as ESP or AH, will contain the Protocol-Id of that protocol (e.g. ESP, AH) and the SPI is the receiving entity's SPI(s).

NOTE: What's the deal with IPcomp SAs. This mechanism is probably not appropriate for deleting them!!

The Delete Payload is defined as follows:

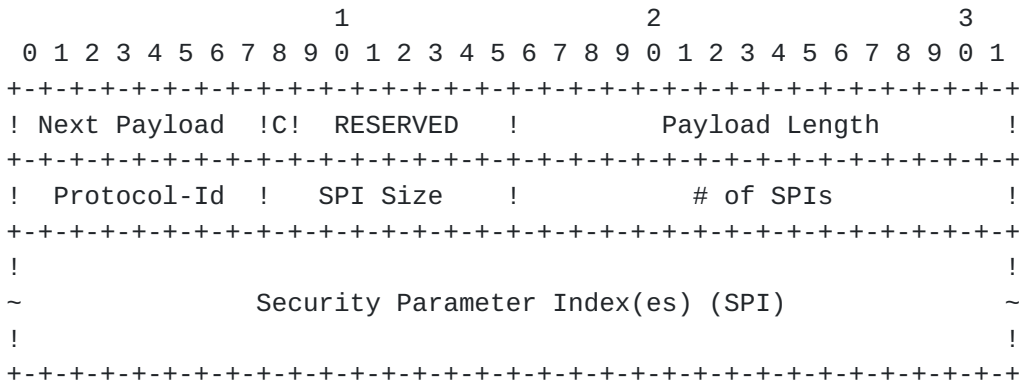

Figure 14: Delete Payload Format

- Protocol-Id (1 byte) - Must be zero for an IKE-SA, [] for ESP, [] for AH, and [] for IPcomp.

- SPI Size (1 byte) - Length in bytes of the SPI as defined by the Protocol-Id. Zero for IKE (SPI is in message header), four for AH and ESP, two for IPcomp.

- # of SPIs (2 bytes) - 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 twelve (12).

7.12 Vendor ID Payload

The Vendor ID Payload 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 backwards compatibility.

The Vendor ID payload is not an announcement from the sender that it will send private payload types but rather an announcement of the sort of private payloads it is willing to accept. The implementation sending the Vendor ID MUST not make any assumptions about private payloads that it may send unless a Vendor ID of like stature is received as well. 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 ID's 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 which 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:

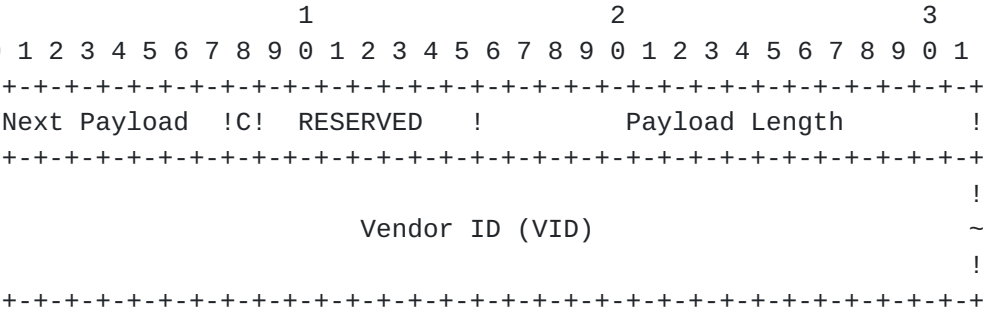

Figure 15: Vendor ID Payload Format

o Vendor ID (variable length) - It is the responsibility of the person choosing the Vendor ID to assure its uniqueness in spite of the absence of any central registry for IDs. 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 thirteen (13).

7.13 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 header followed by selector information fields as follows:

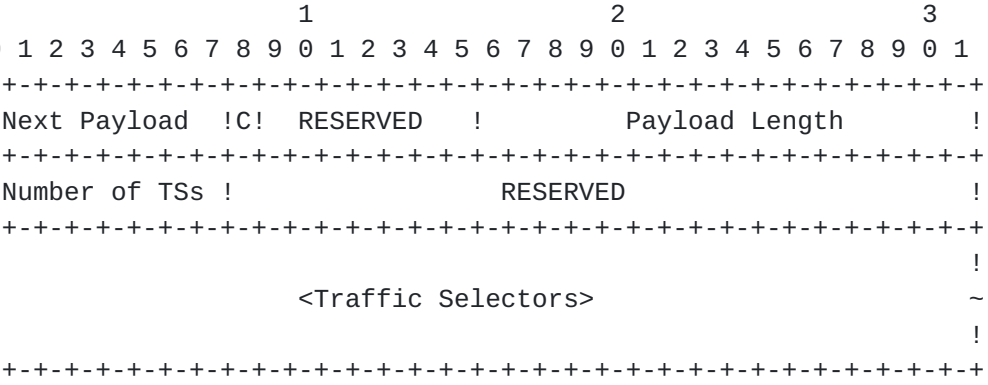

Figure 16: Traffic Selectors Payload Format

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

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

- Traffic Selectors (variable length) - one or more traffic selector substructures.

The length of the Traffic Selector payload includes the TS header and all the traffic selector substructures.
The payload type for the Traffic Selector payload is fourteen (14).

7.13.1 Traffic Selector Substructure

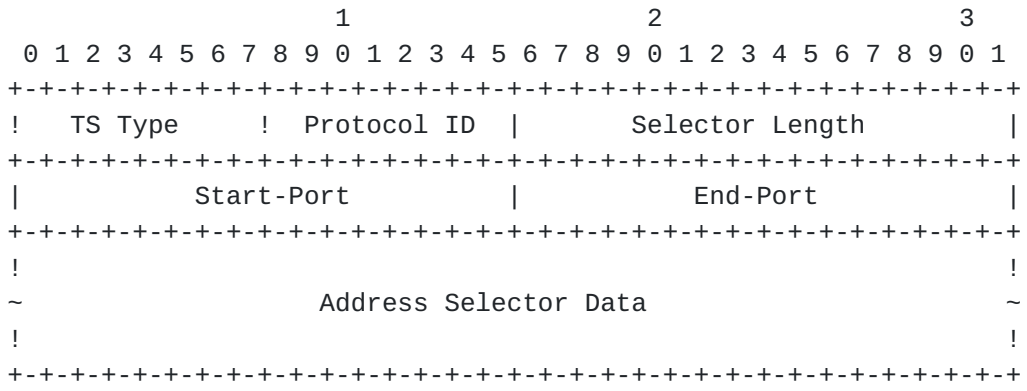

Figure 17: Traffic Selector Substructure

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

- Protocol ID (1 byte) - Value specifying an associated IP protocol ID (e.g. UDP/TCP). 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 bytes) - Value specifying the smallest port number allowed by this Traffic Selector. For protocols for which port is undefined, or if all ports are allowed by this Traffic Selector, this field MUST be zero.

- End-Port (2 bytes) - Value specifying the largest port number allowed by this Traffic Selector. For protocols for which port is undefined, or it all ports are allowed by this Traffic Selector, this field MUST be 65535.

- Address Selector Data - a specification of one or more addresses included in this Traffic Selector with format determined by TS type.

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

TS Type	Value
RESERVED	0
TS_IPV4_ADDR	1

A four (4) byte IPv4 address

TS_IPV4_ADDR_SUBNET	4

An IPv4 subnet represented by a pair of four (4) byte values. The first value is an IPv4 address. The second is an IPv4 network mask. Note that ones (1s) in the network mask indicate that the corresponding bit in the address is fixed, while zeros (0s) indicate a "wildcard" bit.

TS_IPV6_ADDR	5

A sixteen (16) byte IPv6 address

TS_IPV6_ADDR_SUBNET	6

An IPv6 subnet represented by a pair sixteen (16) byte values. The first value is an IPv6 address. The second is an IPv6 network mask. Note that ones (1s) in the network mask indicate that the corresponding bit in the address is fixed, while zeros (0s) indicate a "wildcard" bit.

TS_IPV4_ADDR_RANGE	7

A range of IPv4 addresses, represented by two four (4) byte 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 (16) byte 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.

7.14 Other Payload Types

Payload type values 15-127 are reserved to IANA for future assignment in IKEv2 (see section 10). Payload type values 128-255 are for private use among mutually consenting parties.

8 Diffie-Hellman Groups

There are 5 groups different Diffie-Hellman groups defined for use in IKE. These groups were generated by Richard Schroeppel at the University of Arizona. Properties of these primes are described in [Orm96].

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

8.1 First Group

IKE implementations MAY support a MODP group with the following prime and generator. 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.

8.2 Second Group

IKE implementations SHOULD support a MODP group with the following prime and generator. 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 (decimal)

8.3 Third Group

IKE implementations SHOULD support a EC2N group with the following characteristics. This group is assigned id 3 (three). The curve is based on the Galois Field GF[2^155]. The field size is 155. The irreducible polynomial for the field is:

u^155 + u^62 + 1.

The equation for the elliptic curve is:

y^2 + xy = x^3 + ax^2 + b.

```
Field Size:                          155
Group Prime/Irreducible Polynomial:
             0x080000000000000000000004000000000000001
Group Generator One:                 0x7b
Group Curve A:                       0x0
Group Curve B:                       0x07338f
Group Order: 0x080000000000000000057db5698537193aef944
```

The data in the KE payload when using this group is the value x from the solution (x,y), the point on the curve chosen by taking the randomly chosen secret Ka and computing Ka*P, where * is the repetition of the group addition and double operations, P is the curve point with x coordinate equal to generator 1 and the y coordinate determined from the defining equation. The equation of curve is implicitly known by the Group Type and the A and B coefficients. There are two possible values for the y coordinate; either one can be used successfully (the two parties need not agree on the selection).

8.4 Fourth Group

IKE implementations SHOULD support a EC2N group with the following characteristics. This group is assigned id 4 (four). The curve is based on the Galois Field GF[2^185]. The field size is 185. The irreducible polynomial for the field is:

u^185 + u^69 + 1.

The equation for the elliptic curve is:
y^2 + xy = x^3 + ax^2 + b.

```
Field Size:                          185
Group Prime/Irreducible Polynomial:
            0x020000000000000000000000000000200000000000000001
Group Generator One:                 0x18
Group Curve A:                       0x0
Group Curve B:                       0x1ee9
Group Order: 0x01ffffffffffffffffffffdbf2f889b73e484175f94ebc
```

The data in the KE payload when using this group will be identical to that as when using Oakley Group 3 (three).

8.5 Fifth Group

IKE implementations MUST support a MODP group with the following prime and generator. This group is assigned id 5 (five).

The prime is 2^1536 - 2^1472 - 1 + 2^64 * {[2^1406 pi] + 741804}. 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 ECE45B3D C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8
FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA237327 FFFFFFFF FFFFFFFF
```

The generator is 2.

9 Security Considerations

Repeated re-keying using Phase 2 without PFS can consume the entropy of the Diffie-Hellman shared secret. Implementers should take note of this fact and set a limit on Phase 2 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. The default Diffie-Hellman group (number two) when used with a strong random number generator and an exponent no less than 160 bits is sufficient to use for 3DES. Groups three through five provide

greater security. Group one is for historic purposes only and does not provide sufficient strength to the required cipher (although it is sufficient for use with DES, which is also for historic use only). Implementations should make note of these conservative estimates when establishing policy and negotiating security parameters.

Note that these limitations are on the Diffie-Hellman groups themselves. There is nothing in IKE which prohibits using stronger groups nor is there anything which will dilute the strength obtained from stronger groups. In fact, the extensible framework of IKE encourages the definition of more groups; use of elliptical curve groups will greatly increase strength using much smaller numbers.

It is assumed that the Diffie-Hellman exponents in this exchange 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 security of this protocol is critically dependent on the randomness of the Diffie-Hellman exponents, which should be generated by a strong random or properly seeded pseudo-random source (see RFC1715). While the protocol was designed to be secure even if the Nonces and other values specified as random are not strongly random, they should similarly be generated from a strong random source as part of a conservative design.

10 IANA Considerations

This document contains many "magic numbers" to be maintained by the IANA. This section explains the criteria to be used by the IANA to assign additional numbers in each of these lists.

10.1 Transform Types and Attribute Values

10.1.1 Attributes

Transform attributes are uses to modify or complete the specification of a particular transform. Requests for new transform attributes MUST be accompanied by a standards-track or Informational RFC which defines the transform which it modifies or completes and the method in which it does so.

10.1.2 Encryption Algorithm Transform Type

Values of the Encryption Algorithm define an encryption algorithm to use when called for in this document. Requests for assignment of new encryption algorithm values must be accompanied by a reference to a standards-track or informational RFC that describes how to use this

algorithm with ESP.

10.1.3 Pseudo-random function Transform Type

Values for the pseudo-random function define which pseudo-random function is used in IKE for key generation and expansion. Requests for assignment of a new pseudo-random function MUST be accompanied by a reference to a standards-track or informational RFC describing this function.

10.1.4 Authentication Method Transform Type

The only Authentication method defined in the memo is for digital signatures. Other methods of authentication are possible and MUST be accompanied by a standards-track or informational RFC which defines the following:

- the cryptographic method of authentication.
- content of the Authentication Data in the Authentication Payload.
- new payloads, their construction and processing, if needed.
- additions of payloads to any messages, if needed.

10.1.5 Diffie-Hellman Groups

Values of the Diffie-Hellman Group Transform types define a group in which a Diffie-Hellman key exchange can be completed. Requests for assignment of a new Diffie-Hellman group type MUST be accompanied by a reference to a standards-track or informational RFC which fully defines the group.

10.2 Exchange Types

This memo defines three exchange types for use with IKEv2. Requests for assignment of new exchange types MUST be accompanied by a standards-track or informational RFC which defines the following:

- the purpose of and need for the new exchange.
- the payloads (mandatory and optional) that accompany messages in the exchange.
- the phase of the exchange.
- requirements the new exchange has on existing exchanges which have assigned numbers.

10.3 Payload Types

Payloads are defined in this memo to convey information between peers. New payloads may be required when defining a new

authentication method or exchange. Requests for new payload types MUST be accompanied by a standards-track or informational RFC which defines the physical layout of the payload and the fields it contains. All payloads MUST use the same generic header defined in Figure 2.

11 Acknowledgements

We would like to thank the many members of the IPsec working group that provided helpful and constructive suggestions on improving IKE. Special thanks go to those of you who've implemented it!

This protocol is built on the shoulders of many designers who came before. While they have not necessarily reviewed or endorsed this version and should not be blamed for any defects, they deserve much of the credit for its design. We would like to acknowledge Oakley, SKEME and their authors, Hilarie Orman (Oakley), Hugo Krawczyk (SKEME). Without the hard work of Doug Maughan, Mark Schertler, Mark Schneider, Jeff Turner, Dave Carrel, and Derrell Piper, this memo would not exist. Their contributions to the IPsec WG have been considerable and critical.

12 References

[CAST] Adams, C., "The CAST-128 Encryption Algorithm", RFC 2144, May 1997.

[BLOW] Schneier, B., "The Blowfish Encryption Algorithm", Dr. Dobb's Journal, v. 19, n. 4, April 1994.

[Bra96] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[Bra97] Bradner, S., "Key Words for use in RFCs to indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[Ble98] Bleichenbacher, D., "Chosen Ciphertext Attacks against Protocols Based on RSA Encryption Standard PKCS#1", Advances in Cryptology Eurocrypt '98, Springer-Verlag, 1998.

[BR94] Bellare, M., and Rogaway P., "Optimal Asymmetric Encryption", Advances in Cryptology Eurocrypt '94, Springer-Verlag, 1994.

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

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

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

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

[KBC96] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

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

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

[MSST98] Maughhan, D., Schertler, M., Schneider, M., and J. Turner, "Internet Security Association and Key Management Protocol (ISAKMP)", RFC 2408, November 1998.

[Orm96] Orman, H., "The Oakley Key Determination Protocol", RFC 2412, November 1998.

[PFKEY] McDonald, D., Metz, C., and Phan, B., "PFKEY Key Management API, Version 2", RFC2367, July 1998.

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

[PK01] Perlman, R., and Kaufman, C., "Analysis of the IPsec key exchange Standard", WET-ICE Security Conference, MIT, 2001, http://sec.femto.org/wetice-2001/papers/radia-paper.pdf.

[Pip98] Piper, D., "The Internet IP Security Domain Of Interpretation for ISAKMP", RFC 2407, November 1998.

[RC5] Rivest, R., "The RC5 Encryption Algorithm", Dr. Dobb's Journal, v. 20, n. 1, January 1995.

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

Communications of the ACM, v. 21, n. 2, February 1978.

[Sch96] Schneier, B., "Applied Cryptography, Protocols, Algorithms, and Source Code in C", 2nd edition.

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

[TIGER] Anderson, R., and Biham, E., "Fast Software Encryption", Springer LNCS v. 1039, 1996.

Appendix A

Attribute Assigned Numbers

Certain transforms negotiated in an SA payload can have associated attributes. Attribute types can be either Basic (B) or Variable-length (V). Encoding of these attributes is defined as Type/Value (Basic) and Type/Length/Value (Variable). See section 7.3.3.

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

Attribute Classes

class	value	type
RESERVED	0-5	
Group Prime/Irreducible Polynomial	6	V
Group Generator One	7	V
Group Generator Two	8	V
Group Curve A	9	V
Group Curve B	10	V
RESERVED	11-13	
Key Length	14	B
Field Size	15	B
Group Order	16	V
Block Size	17	B

values 0-5, 11-13, and 18-16383 are reserved to IANA. Values 16384-32767 are for private use among mutually consenting parties.

- Group Prime/Irreducible Polynomial

  The prime number of a MODP Diffie-Hellman group or the irreducible polynomial of an elliptic curve when specifying a private Diffie-Hellman group.

- Generator One, Generator Two

  The X- and Y-coordinate of a point on an elliptic curve. When the Y-coordinate (generator two) is not given it can be computed with the X-coordinate and the definition of the curve.

- Curve A, Curve B

Coefficients from the definition of an elliptic curve:

$$y^2 + xy = x^3 + (\text{curve A})x^2 + (\text{curve B})$$

- 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.

- Field Size

  The field size, in bits, of a Diffie-Hellman group.

- Group Order

  The group order of an elliptical curve group. Note the length of this attribute depends on the field size.

- Block Size

  The number of bits per block of a cipher with a variable block length.

Appendix B: Cryptographic Protection of IKE Data

With the exception of the IKE-SA-INIT-REQUEST, IKE-SA-INIT-RESPONSE, and Informational Exchange error notifications when no IKE-SA exists, all IKE messages are encrypted and integrity protected. The algorithms for encryption and integrity protection are negotiated during IKE-SA setup, and the keys are computed as specified in sections 3 and 4.2.

The encryption and integrity protection algorithms are the same as those available to the ESP protocol, through their application is slightly different. Whereas in ESP the header that is integrity protected but not encrypted is a total of 8 bytes (SPI+Sequence #) plus the IV, in IKE it is the IKE Header which is 28 bytes plus the IV (see section 7.1).

All other aspects of cryptographic processing (including IV insertion, padding, key derivation, trailer insertion) are as specified in [ESP] and its supporting algorithm documents. The Next Header byte in the encrypted ESP payload MUST be set to zero.

NOTE: This is a change from IKEv1, which along with its companion specifications defined its own algorithms for padding, encryption, and integrity protection and its own codes for cryptographic algorithms. Since most IKE implementations will also include ESP implementations, this alternative was thought to simplify both the specification and the implementation, as well as limit the number of techniques in need of analysis for soundness.

Expansion of SKEYSEED

In some circumstances SKEYSEED_e may not be long enough to supply all the necessary keying material an algorithm requires. In this case the key is derived from feeding the results of the prf into itself, concatenating the results and taking the highest necessary bits.

Consider a fictitious cipher AKULA which requires 320 bits of key and the prf used to generate SKEYSEED_e only generates 120 bits of material. The key for AKULA would be the first 320 bits of Ka where:

```
Ka = K1 | K2 | K3
```

and

```
K1 = prf(SKEYSEED_e, 0)
K2 = prf(SKEYSEED_e, K1)
K3 = prf(SKEYSEED_e, K2)
```

where 0 is represented by a single byte. Each result of the prf provides 120 bits of material for a total of 360 bits. AKULA would use the first 320 bits of that 360 bit string.

Authors' Addresses

Dan Harkins
dharkins@tibernian.com
Tibernian Systems

Charlie Kaufman
ckaufman@iris.com
IBM

Radia Perlman
radia.perlman@sun.com
Sun Microsystems