Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: October 14, 2013

D. Harkins, Ed.
Aruba Networks
April 12, 2013

The (Real) Internet Key Exchange
draft-harkins-ikev3-01

Abstract

The current version (v2) of the Internet Key Exchange failed to address many of the shortcomings of the original version (v1). This memo defines a new version (v3) of the Internet Key Exchange that attempts to do so.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on October 14, 2013.

Table of Contents

1. Introduction

The Internet Key Exchange was first defined in [RFC2409] to generate security associations for the IPsec protocols. That specification was poorly written and suffered from too many options, many of which were unneeeded and went unused. In short, it was confusing and complicated. An effort was made to come up with a simpler and less confusing version, and that resulted in [RFC4306], so-called IKEv2 ([RFC2409] was then dubbed IKEv1). While it was arguably simpler and less confusing, IKEv2 failed to achieve its goal. It went through an extensive clarification process that produced [RFC4718] and development of a replacement specification for IKEv2, in [RFC5996]. While [RFC5996] is definitely a cleaner protocol than [RFC2409] it still has too many options and is too complicated, and confusing.

This memo defines an IKEv3 in an effort to have a simpler, more easy to implement protocol, that that has a high probability of achieving interoperability while retaining security and utility.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Characteristics of Version 3 of IKE

Version 3 of the Internet Key Exchange is a simple peer-to-peer protocol in which each side sends and receives two messages. It performs mutual authentication, derives a shared, secret and authenticated key, and negotiates parameters for IPsec security associations (SAs) to protect transient data between the peers.

Either side can initiate the exchange to the other and both sides can initiate simultaneously (hence the claim of a true "peer-to-peer" exchange). This is advantageous for certain smart devices-- aka "the Internet of things"-- or sensor network deployments where there is no strict roles of client or server, initiator or responder.

In an effort to keep the definition of the Internet Key Exchange as simple as possible, negotiation of the terms of its operation-- e.g. encryption algorithm, hash algorithm-- is kept to a minimum.

3. Differences from Previous Versions of IKE

3.1. Identity Confidentiality

IKEv3 does not provide identity confidentiality. This is a tradeoff made to increase simplicity in specification and, more importantly, implementation at the cost of a feature whose benefit is somewhat dubious.

While security on the Internet is a large issue (one that IKEv3 addresses) the problems associated with exposure of the identities of two peers that are engaging in secure communication is not.

If identity hiding critical for a particular deployment, IKEv3 supports obfuscation of identity using an ID blob which has meaning to the two peers of the exchange but has no meaning to any third party that may observe it.

3.2. Single IKE SA, No Lifetime

IKEv3 can handle the situation where both sides initiate to each other without resorting to carrying on two conversations and ending up with two IKE Security Associations.

The IKEv3 SA is also short-lived. Its purpose is to create SAs for IPsec and once it has done that the state that governed a particular protocol run can go away. There is no notion of a long-lived IKE SA.

There is no SA lifetime necessary for IKEv3 to negotiate. This also has the benefit of doing away with the Delete payloads and their corresponding complexity as well as the complexity associated with rekeying of SAs.

There is no need for "initial contact" notification or the need to negotiate, or rekey, multiple IKE SAs.

3.3. Not A Request/Response Exchange

[RFC2409] and [RFC5996] are both Request/Response protocols. There are defined roles-- one side is an "Initiator" and the other is a "Responder"-- and one side makes Requests and the other Responds to those requests.

In IKEv3 there are no roles involved-- no clients and servers, no Initiators and Responders-- just two peers who perform identical behavior. Since either side can initiate and both sides can initiate simultaneously, there is no need to deal with "Exchange Collisions". All the protocol specification complexity to address the problems that occur due to role-based protocol definition goes away.

Many of the IKEv1 and IKEv2 use cases involved strict roles and IKEv3 can support them because the state machine (see Section 6.3) can handle the case where one side initiates and the other responds just as easily as it can handle the case where both sides initiate simulteneously.

3.4. State Machine Definition

IKEv3 defines a very simple state machine that each side runs through to implement the protocol. Accurate compliance with the state machine ensures interoperability.

The state machine allows the protocol definition to be entirely from the point of view of the implementation, making protocol implementation much easier. The protocol is defined in terms of actions causing events which result in a deterministic advancement of state until the protocol is finished. The state machine definition assures that each side either completes the protocol or neither side completes the protocol.

Previous versions of IKE lacked a state machine definition and it showed. IKEv1 achieved interoperability through implementor "bake-offs" and not through a coherent specification. IKEv2 has gone through forty (40) revisions, in total, in an effort to clarify and straighten out ambiguous and confusing text and remains to this day a complicated, ambiguous and confusing specification.

4. Cryptographic Tools

IKEv3 makes use of certain cryptographic primitives to achieve its goals of key generation, mutual authentication, and security. Each of the following subsections indicate negotiable components of the IKE Security Association that are used during the protocol run. Different protocol runs can negotiate different components and the components to use in a particular run of the protocol are established by exchanging payloads in the Init message that describe the attributes of the IKE Security Association (see Section 6.4).

4.1. Authenticated Encryption

Authenticated encryption is employed by IKEv3 to protect the payloads that set up IPsec Security Associations as well as any vendor-specific payloads that are added to the final two messages of the protocol. It is therefore a negotiable component. The privacy algorithm negotiated MUST be a cipher mode, or construction of cipher mode plus integrity check, that provides authenticated encrytion.

AES in SIV mode as defined in [RFC5297] is used in IKEv3 to accomplish this goal. SIV supports authenticated encryption with associated data (which is authenticated but not encrypted) and does not require complex managment of a unique counter space to ensure security. It is simple, secure and robust. A perfect fit for IKEv3.

4.2. Hash Function

A hash function takes a arbitrary-sized input and deterministically produces a fixed sized output, called a digest. It is also a one-way function: it is very easy to produce a digest but computationally infeasible to reconstruct the arbitrary-sized input given a particular digest.

IKEv3 uses a hash function, in [RFC2104] mode for key derivation and key confirmation. IKEv3 also uses a hash function to construct a random function, H():

```
H(x) = HMAC-Hash(0^n, x)
```

where Hash is the agreed-upon hash function and "0^n" signifies a key of all zeros whose length equals the digest size of the hash function.

IKEv3 defines SHA-256 and SHA-512 (as defined in [RFC4634]) for use as hash functions.

4.3. Discrete Logarithm Cryptography

The Internet Key Exchange uses discrete logarithm cryptography. Each party to the protocol derives ephemeral public and private key pairs with respect to a particular domain parameter set, called a "group". The group can be based on either finite field cryptography (modular exponentiation, or MODP, groups) or elliptic curve cryptography (ECP groups).

In this memo, elements in a group are denoted in upper case and scalar values are in lower case-- element X and scalar x.

Groups are identified in messages using a convenient registry maintained by IANA (see Section 8). Each group's domain parmameter set contains the following:

- p - a prime number defining a finite field
- G - a generator, a base element forming a group

- q - a prime number indicating the order of the group defined by G

ECP groups additionally define "a" and "b" which are components of the equation of the elliptic curve-- y^2 = x^3 + ax + b. Some MODP groups are based on safe primes and do not have a specific order defined. For these groups only, the order, q, SHALL be (p-1)/2.

For each group, the following operations are defined:

- "scalar operation" -- takes a scalar and an element in the group to produce another element in the group-- Z = scalar-op(x, Y). For ECP groups this is multiplication of the element by the scalar; for MODP groups this is exponentiation of the element to the scalar.

- "element operation" -- takes two elements in the group to produce a third element in the group-- Z = element-op(X, Y). For ECP groups this is point addition; for MODP groups this is modular multiplication.

- "inverse operation" -- takes an element in the group and returns another element in the group such that the element operation on the two produces the identity element of the group-- Y = inverse(X).

    ECP: element-op(Y, inverse(Y)) = "point at infinity"

    MODP: element-op(Y, inverse(Y)) = 1

In addition, ECP groups require a mapping function, r = F(R), to convert a group element to an integer. The mapping function used in this memo returns the x-coordinate of the point it is passed. MODP groups do not need a mapping function as group elements in MODP groups can be directly represented as integers. For the purpose of protocol definition, the function F() when used with MODP groups will be the identity function-- i.e. i = F(i).

4.4. Key Deriviation Function

IKEv3 uses a key derivation function, KDF(), to stretch a random key to an indeterminate length and bind some arbitrary data to the stretched key.

For ease of programming, IKEv3 uses the prf+() function from [RFC5996] which, in turn, was derived from the prf() function in [RFC2409], as its KDF().

THe pseudo-random function at the core of the prf+() construct is the

agreed-upon hash function in HMAC ([RFC2104]) mode. When KDF() is called for in this memo, it is prf+() from [RFC5996] using HMAC-Hash where hash is the agreed-upon hash algorithm.

5. Authentication and Key Establishment

The goal of any pairwise authenticated key exchange is key establishment and mutual authentication. The IKEv3 protocol achieves these goals. The particular method of key establishment is tied to the authentication method which is tied to the type of credential used for authentication-- a (certified) public key or a PSK.

5.1. Public Key Authentication

Public key authentication uses a Diffie-Hellman key exchange for key establishment and digital signatures by a private key whose public analog is trusted by the peer.

5.1.1. KE Payload with Public Key Authentication

The KE payload is used to present a Diffie-Hellman public value to the peer. Each peer generates a random number between one (1) and the order of the group, q, exclusive. This represents the peer's private value, priv. The peer then performs the group's scalar operation (see Section 4.3) with the group's generator to produce the public value, Pub:

```
Pub = scalar-op(priv, G)
```

The public value is encoded in to the body of the KE Payload (see Section 6.4) according to the integer to octet string conversion technique from [RFC6090].

The Diffie-Hellman key exchange is completed when both sides have finished sending and receiving an Init message. Each side generates the same shared secret, secret, by applying the mapping function, F() (see Section 4.3), to the result of the group's scalar operation with the entities private value, priv, and the peer's public key, PubPeer:

```
secret = F(scalar-op(priv, PubPeer))
```

The secret is then used to generate three additional keys, the authenticated encryption key, the confirmation key, and a key-derivation key. (see Section 5.3).

5.1.2. AU Payload with Public Key Authentication

The AU payload contains a digital signature of the confirmation key and both peers' Init messages concatenated together, transmitter's Init message first. For example, assuming Alice sent the message InitA and Bob send the message InitB, Alice's digital signature would be "sig" where:

```
sig = Sign-Alice(cKEY | InitA | InitB)
```

where "|" signifies concatenation, and Sign-Alice() indicates a digital signature of that data passed to it using the public key of Alice. The portions of the Init messages that are covered by the digital signature consist of the IKE header (inclusive) to the end of the payload.

Bob would similarly send:

```
sig = Sign-Bob(cKEY | InitB | InitA)
```

since Bob was the transmitter of InitB.

To maintain a consistent level of security for IKEv3, the hash algorithm used to generate the digital signature SHALL be the one negotiated in the IKE Security Association that is used for other hashing purposes in IKEv3. The body of the AU payload (see Section 6.4) SHALL consist of the digital signature as a bitstring.

5.2. PSK Authentication

PSK authentication uses the "dragonfly" key exchange to both generate a shared, and secret, key and to mutually authenticate the peers to each other. Each side proves knowledge of the PSK in a manner that is resistant to dictionary attack.

Each side proves possession of a single PSK (or password), there is no notion of a "client's password" and a "server's password"; there is just the one. This single PSK MUST be provisioned on the two peers prior to beginning the IKEv3 exchange. Since there is only one PSK it SHOULD have only one name, which is provisioned along with the PSK. It is this name that is used in the ID payload when initiating the dragonfly exchange to the peer. Note: it may make sense in certain client/server deployments to have a proper client username assigned to the password, in which case the server proving possession of the client's password-- identified by username-- authenticates it to the client.

When PSK authentication is chosen for a particular run of the

protocol, the KE payload contains each peer's "commit" contribution to the dragonfly exchange and the AU payload contains a keyed message authentication code binding the secret key to both peers' Init messages concatenated together.

Prior to beginning the "dragonfly" exchange, both peers MUST agree upon a secret element in the chosen group. A secret seed is generated and that see is used in a group-specific hunting-and-pecking process-- one process for MODP groups and another for ECP groups. First, an 8-bit counter is set to one (1) and a secret base is computed using the negotiated one-way function with the secret PSK, and the counter:

```
base = H(PSK | counter)
```

The base is then stretched using the key derivation function from Section 4.4 to the length of the prime from the group's domain parameter set:

```
seed = KDF(base, "IKE PSK Hunting and Pecking")
```

The seed is then passed to the group-specific hunting and pecking technique.

5.2.1. Hunting and Pecking with ECP Groups

The ECP specific hunting and pecking technique entails looping until a valid point on the elliptic curve has been found. The seed is used as the x-coordinate with the equation of the curve to solve for a y-coordinate. If there is no solution, the counter is incremented, a new base and new seed are generated and the hunting and pecking continues. If there is a solution an ambiguity exists because two values for the y-coordinate would be valid. The low-order bit of the base is used to unambiguously determine the y-coordinate and the resulting (x,y) pair becomes the secret generator for the dragonfly exchange, SKE.

Algorithmically, the process looks like this:

```
found = 0
counter = 1
do {
  base = H(psk | counter)
  seed = KDF(seed, "IKE PSK Hunting And Pecking")
  if (seed < p)
  then
    x = seed
    if ( (x^3 + ax + b) is a quadratic residue mod p)
    then
      y = sqrt(x^3 + ax + b)
      if (LSB(y) == LSB(base))
      then
        SKE = (x,y)
      else
        SKE = (x, p-y)
      fi
      found = 1
    fi
  fi
  counter = counter + 1
} while (found == 0)
```

Figure 1: Fixing SKE for ECP Groups

5.2.2. Hunting and Pecking with MODP Groups

The MODP specific hunting and pecking technique entails finding a random element which, when used as a generator, will create a group with the same order as the group created by the generator from the domain parameter set. The secret generator is found by exponentiating the seed to the value ((p-1)/q), where p is the prime and q is the order from the domain parameter set. If that value is greater than one (1) it becomes SKE, otherwise the counter is incremented, a new base and seed are generated, and the hunting and pecking continues.

Algorithmically, the process looks like this:

```
found = 0
counter = 1
do {
  base = H(psk | counter)
  seed = KDF(base, "IKE SKE Hunting And Pecking")
  if (seed < p)
  then
    SKE = seed ^ ((p-1)/r) mod p
    if (SKE > 1)
    then
      found = 1
    fi
  fi
  counter = counter + 1
} while (found == 0)
```

Figure 2: Fixing SKE for MODP Groups

5.2.3. KE Payload with PSK Authentication

Once SKE has been determined, the peer randomly chooses two numbers between one and the order of the group, q, exclusively. These represent a private value and a mask. The peer then generates a scalar and an element using private, mask, and SKE:

```
scalar = (private + mask) mod q

Element = inverse(scalar-op(mask, SKE))
```

The scalar and element, respectively, are encoded into the KE payload by using the integer to octet string conversion technique from [RFC6090]. Octet strings are pre-pended with zero (0), if necessary, to achieve the required resulting length. Since the length of each component of the KE payload is implicitly known, the scalar and element can be extracted from the KE payload for processing.

The scalar is the same length as the order of the group. It is converted into an octet string and then the octet string is inserted into the body of the KE Payload.

If the selected group is MODP, the element can be treated directly as an integer and converted into an octet string. Its length is the same as the length of the prime of the group. The converted octet string is appended to the octet string representation of the scalar.

If the selected group is ECP, the element is an (x,y) pair and each coordinate is separately converted into an octet string, each of which is the same length as the prime of the group. The octet string

representation of the x-coordinate SHALL be appended to the scalar and the y-coordinate SHALL be appended to the x-coordinate.

The dragonfly key handshake is completed when both sides have finished sending and receiving an Init message. Each side generates the same shared secret, secret, by performing the following computation:

```
secret = F(scalar-op(private,
                     element-op(PeerElement,
                                scalar-op(peerscalar, SKE))))
```

where peerscalar and PeerElement are scalar and element from the peer's KE payload taken out of a received Init message. The secret is then used to generate three additional keys, the authenticated encryption key, the confirmation key, and a key-derivation key. (see Section 5.3).

5.2.4. AU Payload with PSK Authentication

The AU payload contains a keyed message authentication code which proves knowledge of the derived secret, and therefore knowledge of the PSK, and binds the two Init messages to the authenticated state.

Each side produces an authenticating message authentication code, mac, by invoking the HMAC version of the negotiated hash function and passing the confirmation key, cKEY, as the key and the concatentation of both peers' Init messages concatenated together, transmitter's Init message first. For example, assuming Alice sent the message InitA and Bob sent the message InitB, Alice's message authentication code would be "mac" where:

```
mac = HMAC-Hash(cKEY, InitA | InitB)
```

where "|" signifies concatentation and HMAC-Hash is the [RFC2104] instantiation of the negotiated hash algorithm, Hash. The portions of the Init messages passed HMAC-Hash consist of the IKE header (inclusive) to the end of the payload.

Bob would similarly send:

```
mac = HMAC-Hash(cKEY, InitB | InitA)
```

since Bob was the transmitter of InitB.

5.3. Deriving Shared Secrets

Upon successful completion of key establishment, IKEv3 produces three keys, an authenticated encryption key, aeKEY, to protect the Auth Messages, a confirmation key, cKEY, and a derivation key, dKEY, used to derive (a) shared secret(s) when constructing IPsec Security Associations (see Section 6.2).

The length of aeKEY depends on the authenticated encryption mode used and the length of cKEY and dKEY SHALL be the length of the digest of negotiated hash function. The keys are derived by passing the two nonces, appended to each other with the lexicographically larger nonce being first, as the key and secret from the authenticated key exchange concatenated with the label "IKEv3 Key Derivation" as the data:

```
aeKEY | cKEY | dKEY = KDF(max(Na, Nb) | min(Na, Nb),
                          secret | "IKEv3 Key Derivation")
```

where Na and Nb are the two nonces from the exchange (the transmitter is irrelevant in this peer-to-peer protocol), max() returns the lexicographically larger of the two parameters passed, and min() returns the lexicographically smaller of the two parameters passed.

The key aeKEY SHALL be used to protect the exchange of Auth Messages, the same key is used in both directions.

6. The Internet Key Exchange Protocol

The Internet Key Exchange (IKE) authenticates two peers to each other and derives security associations for use by IPsec. The credentials supported by IKE are PSKs and certificates.

IKE supports varying degrees of security by supporting various domain parameter groups, encryption algorithms, and hash algorithms.

IKEv3 supports detection of NATs between two peers through the exchange of source and destination indicators. When (a) NAT(s) is (are) present between the peers the source and/or destination addresses and/or ports will be modified and differ from those in the indicators. When (a) NATS(s) is (are) detected, UDP encapsulation of ESP traffic as defined by [RFC3948] is required. Note that IKEv3 is not required to use port 4500 in the presense of (a) NAT(s).

6.1. Message Flow

In the IKEv3 protocol each peer sends and receives an Init message and an Auth message. The Init message negotiates the type of authentication to be used between the peers, identifies the peers to each other, exchanges random nonces, and exchanges the components of a cryptographic key exchange. The Auth message authenticates the peer to the other peer and establishes IPsec security associations.

Messages are comprised of an IKE header followed by one or more payloads. The on-the-wire format of the IKE header and all payloads defined for use in IKE are in Section 6.4.

As is typical in these sorts of memos, the participants in the protocol are Alice and Bob. The exchange of Init and Auth messages between Alice and Bob look like this:

```
             Alice                                  Bob
            ------                                 ----
Init: hdr, IAa, IDa, NOa,                    hdr, IAb, IDb, NOb,
        KEa  [, CRa]           ---->  <-----     KEb [, CRb ]
Auth: hdr, { [CEa, ] AUa, AIs,               hdr, { [CEb, ] Aub, AIs,
        AId, SAa, TSs, TSd }   ---->  <-----     AId, SAb, TSs, TSd }
```

Where { x } indicates the authenticated encryption of payload x using the mode agreed upon in the exchange of IA payloads.

6.1.1. Init Messages

6.1.1.1. Construction of Init Messages

The IKEv3 header contains two message identifiers called SPIs, one chosen by the transmitter of the message and one chosen by the (intended) recipient of the message. The local SPI from the IKE security association is copied into the transmitter SPI field. If a peer SPI exists in the IKE security association, it is copied into the receipient SPI field. If there is no peer SPI in the IKE security association, the receipient SPI field remains all zero.

The first payload in an Init message MUST be an IA payload which indicates the terms by which the IKE protocol will be run (see Section 4). If this Init message is being constructed in response to receipt of an accepted Init message then the attributes from the received, and accepted, Init message MUST be copied into the Init message being constructed. If the Init message is being constructed due to an indication to the IKEv3 protocol to establish IPsec SAs with a remote peer (see Section 6.3) then the attributes MUST reflect the policy that accompanied that indication.

The next payload after the IA payload MUST be the ID payload which indicates the identity of the peer sending the Init message (see Section 3.1 for a discussion on identity confidentiality). The next payload MUST be a NO payload which contributes additional randomness to the exchange. The next payload MUST be a KE payload. The particular construction of the body of the KE payload depends on the authentication method being used for this run of the protocol (see Section 5). Finally, a CR payload MAY be added if the authentication method is public key authentication and the sender of the Init message believes that it needs the peer's public key.

Vendor specific payloads MAY be appended to an Init message to convey some additional semantics governing the Init message.

6.1.1.2. Processing of Init Messages

The first step of processing an Init Message is to record the peer's SPI by taking it out of the transmitter SPI field of the IKEv3 header and storing it in the IKE security association as the peer's SPI.

Next, the attributes that govern the IKEv3 protocol are checked. If the recipient SPI field is not all zeros (0) then the attributes in the received Init message MUST be identical to the attributes that have already been sent to the peer. If they are not, processing indicates a failure and stops. If the recipient SPI field is all zeros (0) and a message has not been sent to the peer then the attributes are checked for acceptability. If they are not acceptable processing SHALL indicate a failure and stop. If they are acceptable, then processing continues. Otherwise, if the recipient SPI field is all zeros (0) and a message has already been sent to the peer then there are three possible cases:

1. The attributes are identical to the attributes sent to the peer, so processing continues.

2. The attributes are not acceptable in which case the message is discarded and processing stops.

3. The attributes are acceptable but differ from those sent. In this case, a test is made to see which side drops its offer. Each side has sent its SPI to the other as the transmitter SPI in its Init message. If the low-order bit of those SPIs are identical then the transmitter of the larger SPI wins, if the low-order bit of those SPIs differ then the transmitter of the smaller SPI wins. The winner SHALL discard the message and the loser SHALL indicate a failure. In both cases, processing stops. Note: the loser will destroy all state associated with this conversation and the winner will retransmit, allowing the two to

synch up on the new, mutually acceptable attributes.

Finally, the nonce and key exchange data are extracted from the received Init message and processing finishes successfully. If the peer requested a certificate, that fact is noted to ensure that a certificate is included in the subsequent Auth message.

6.1.2. Auth Messages

6.1.2.1. Construction of Auth Messages

The Auth message MAY optionally contain a certificate payload (CE) with the public key of the transmitter and MUST contain, in the following order, an Auth payload (AU), a source Address Indication payload (AIs), a destination Address Indication payload (AId), a Security Association payload (SA), and two Traffic Selector payloads (TS). Optional vendor specific payload(s) MAY be appended to the message but MUST be the last payload(s) in the message.

The contents of AU payload are determined by the authentication method agreed-upon during the exchange of Init messages (see Section 5). The value of the Auth payload, from the point of view of the transmitter, SHALL be determined and copied into the data portion of the payload.

The AIs and AId payloads are constructed from the point of view of the transmitting peer. The source Address Indication payload, AIs, is the address and port being used as the source of the Auth message and the destination Address Indication payload, AId, is the address and port of the destination of the Auth message. The source Address Indication payload MUST precede the destination Address Indication payload.

The contents of the SA payload describe the transforms and options that will be represented in the IPsec SA after successful authentication. If this Auth message is being constructed in response to the receipt of an Auth message from the peer, the transforms in the Auth payload MUST be identical to those accepted when processing the peer's Auth message. If a locally-unique SPI with which to identify the security association for received IPsec packets has not yet been chosen, the transmitter choses a SPI.

The TS payloads contains a description of the flows to protect using IPsec. There are two (2) TS payloads in each Auth message. The first describes the source of the flow and the second describes the sink of the flow, both from the perspective of the transmitter of the Auth message. If local Traffic Selector policy has been narrowed due to the processing of the peer's Auth message, then the narrowed

policy SHALL be reflected in the TS payloads.

Auth messages are sent after each side has both sent and received an Init message and completed the key establishment phase of the IKEv3 protocol. While the peers have not yet authenticated each other, they share a secret which can be used to secure the Auth messages. This is accomplished by using the authenticated encryption mode that was agreed-upon during the exchange of Init messages.

All the payloads of the Auth message are encrypted-- that is, everything after the IKEv3 header to the end of the message. The IKEv3 header, and the encrypted payloads are all authenticated.

When [RFC5297] is used for authenticated encryption the IKEv3 header from the transmitter's SPI (inclusive) to the Length (inclusive) is passed as associated data, and the data immediately following the IKEv3 header, from the generic IKEv3 header of the first payload (inclusive) to the end of the message is the data to encrypt. The ICV/SIV field of the IKEv3 header is not included in the associated data passed to AES-SIV.

The output of the [RFC5297] mode is ciphertext and a Synthetic Initialization Vector (SIV). The SIV SHALL be copied into the IKEv3 header and the ciphertext is appended to the IKEv3 header to form the complete Auth message.

6.1.2.2. Processing of Auth Messages

Auth messages are encrypted and authenticated so the first step in processing is to verify their integrity and to decrypt them. When [RFC5297] is used, the Synthetic Initialization Vector (SIV) is copied from the ICV/SIV field in the IKEv3 header. The IKEv3 header from the transmitter's SPI (inclusive) to the length (inclusive) is passed, along with the SIV to AES-SIV. If AES-SIV outputs FAIL the message is discarded and processing stops. If AES-SIV outputs plaintext, the plaintext will be the sequence of payloads that comprise the Auth message.

The first payload will be the Auth payload (AU). The contents of the AU payload are determined by the authentication method agreed-upon during the exchange of Init messages (see Section 5). The value of the the Auth message from the point of view of the transmitter (i.e. the peer) is calculated and compared to the value in the data portion of the AU payload. If they differ, the peer fails authentication, processing stops and a failure MUST be returned. If they are identical processing continues.

Next the Address Indication payloads are checked. If the source

address or port of the received message differ from the address or port in the source Address Indication payload, or if the destination address or port of the received message differ from the address or port in the destination Address Indication payload then a NAT is detected. Othewise, a NAT is not detected.

The SA payload is next. The transforms in the SA payload are checked to determine whether they are acceptable according to local policy. If they are not the message is discarded and processing stops. If they are, then there are three possibilities:

- An Auth message has not yet been sent to the peer, in which case processing continues; or,
- An Auth message has been sent to the peer and the transforms are identical to those sent, in which case processing continues; or,
- An Auth message has been sent to the peer and the transforms differ from those sent. In this case, a test is made to see which side's offer prevails. Each side has sent its IPsec SPI to the other in the SA payload. If the low-order bit of those SPIs are identical then the transmitter of the larger SPI prevails. If the low-order bit of those SPIs differ then the transmitter of the smaller SPI prevails. The transforms offered by the prevailing party SHALL be adopted by the party which does not prevail. Processing continues.

The Traffic Selectors in the TS payloads are next checked to determine whether they are accptable according to local policy. If the they are not acceptable, and no narrowing of the scope of the traffic flows is possible-- i.e. no intersection between the TS payloads and local policy-- the Auth message SHALL be discarded, processing stops.

If the Traffic Selectors are completely satisfactory and require no narrowing, then the Traffic Selectors are retained for creation of IPsec SAs and construction of an Auth message (if one has not already been sent).

If the Traffic Selectors are partially acceptable, and require narrowing, then the union of the local policy describing the flow and the Traffic Selectors describing the flow SHALL be retained for creation of IPsec SAs and construction of an Auth message (if one has not yet already been sent).

If an Auth message has not yet been sent, a locally-unique SPI SHALL be created to identify the IPsec SA for received IPsec-protected packets. This SPI MUST be retained for use when constructing the

Auth message response.

Upon completion of processing an Auth message, Two IPsec SAs MUST be instantiated (e.g. plumbed into the kernel) with the indicated transforms for the flow described in the (possibly narrowed) Traffic Selectors, one in each direction. The locally-unique SPI becomes the identifier to look up the SA for inbound IPsec packets and the peer's SPI (from its SA payload) becomes the identifier to look up the SA for outbound IPsec packets.

If a NAT was detected, the IPsec SAs MUST use UDP encapsulation for IPsec (see [RFC3948]). Since both sides know the original addresses and ports and the NATted addresses and ports, it is possible to obtain the required information to perform the necessary decapsulation procedures on received UDP-encapsulated IPsec packets.

6.2. IPsec Security Associations

The goal of the Internet Key Exchange is the creation of Security Associations (SAs) for [RFC4301]. SAs are established using Security Association (Section 6.4.12) and Traffic Selector (Section 6.4.11) payloads to negotiate the flow(s) to protect and the means to go about protecting it (them).

IKEv3 derives keys for IPsec SAs using the KDF (Section 4.4). The key derivation key, dKEY, established in Section 5.3 is used as the key and the label "IPsec Key Derivation" is used as the data:

```
key = KDF(dKEY, "IPsec Key Derivation")
```

The length of the key derived by KDF depends on the parameters of the IPsec SA and the key lengths used by the underlying primitives. If multiple, distinct, keying material is used-- for example, an ESP SA that performs encryption and integrity protection separately-- the key used for encryption MUST be taken from first and the key used for integrity protection MUST be taken from the remaining bits.

6.3. State Machine

The IKEv3 protocol is managed by a parent process that receives protocol events and IKEv3 packets and passes them on to instances of the IKEv3 state machine.

The state machine for IKE defines the behavior of a single run of the protocol. Each peer maintains a "protocol instance" for each remote peer that it is actively performing the protocol with that defines the current state of the protocol for that peer. The state machine guarantees that both sides will complete the protocol with each side

installing IPsec Security Associations or each side will fail to complete the protocol.

The state machine addresses the potential of dropped messages with a retransmission timer. This memo does not specify a period that state machines use when setting its retransmission timer.

6.3.1. Parent Process

The parent process of the IKEv3 state machine handles events from the IPsec SADB (e.g. an "acquire" message to create an IPsec security association) as well as receives incoming IKEv3 messages that it dispatches to state machine instances.

The parent process is also responsible for creation of state machine processes. The state of a state machine is stored in an IKEv3 security association so creation of a state machine process entails creation of a nascent IKEv3 security association, generating a unique and unpredictable local SPI, setting the peer address, and putting the state machine in NOTHING state.

When the parent process receives an event from the IPsec SADB to create an IPsec security association it first checks whether there is an existing IKEv3 state machine process with the indicated peer. If so, the parent process drops the event and waits for the process to complete. If there is no existing state machine process, the parent process creates a new state machine (see above) and sends the newly created state machine process a START event.

When the parent process receives an IKEv3 packet from a remote peer it first checks the receipient SPI field in the received packet.

If the recipient SPI field is all zeros, it indicates a peer that is initiating. If the IKEv3 message is an Auth frame, it SHALL be dropped as being meaningless (it is not possible to initiate IKEv3 with an Auth message). If the IKEv3 message is an Init message, the parent process checks whether there is an existing IKEv3 state machine process for the remote peer (the transmitter of the packet) that is in Initiation State. If so, the received message is passed to the state machine process with an INIT event. Otherwise, if there is no existing IKEv3 state machine process in Initiation State, the parent process creates an IKEv3 state machine process (see above) and passes the received message and an INIT event to it.

If the receipient SPI field is not all zeros, the parent process uses the recipient SPI to look up an existing IKEv3 process. If none exists, the packet SHALL be dropped. If the parent process succeeds in looking up an existing IKEv3 state machine process using the

recipient SPI, the message is passed to that state machine process with the appropriate event-- an INIT event for an Init message and an AUTH event for an Auth message.

6.3.2. Components of State Machine

The following states are part of the state machine:

- Nothing: a quiescent state in which nothing has happened
- Initiation: an Initiator has sent an Initiate message to a peer
- Reception: a Responder has sent an Initiate message to a peer
- Done: an Authenticate message has been sent to a peer

The following variables are used in the state machine:

- retrans: the number of retransmissions made (unsigned)
- thresh: the maximum nuber of retransmissions allowed (unsigned)
- committed: a counter on transmitted Auth messages (signed)
- reauth: a counter on received Auth messages (signed)

The following events are delivered to the state machine:

- START: an instruction to initiate IKE to a peer
- INIT: receipt of an Initiate message from a peer
- AUTH: receipt of an Authenticate message from a peer
- TM: expiry of the retransmission timer

The following actions are taken by the state machine:

- init: send an Initiate message to a peer
- auth: send an Authenticate message to a peer

The following timers are used by the state machine:

- tm: the retransmission timer
- fin: a deletion timer

6.3.3. States

The state machine for an IKEv3 process is show in Figure 3.

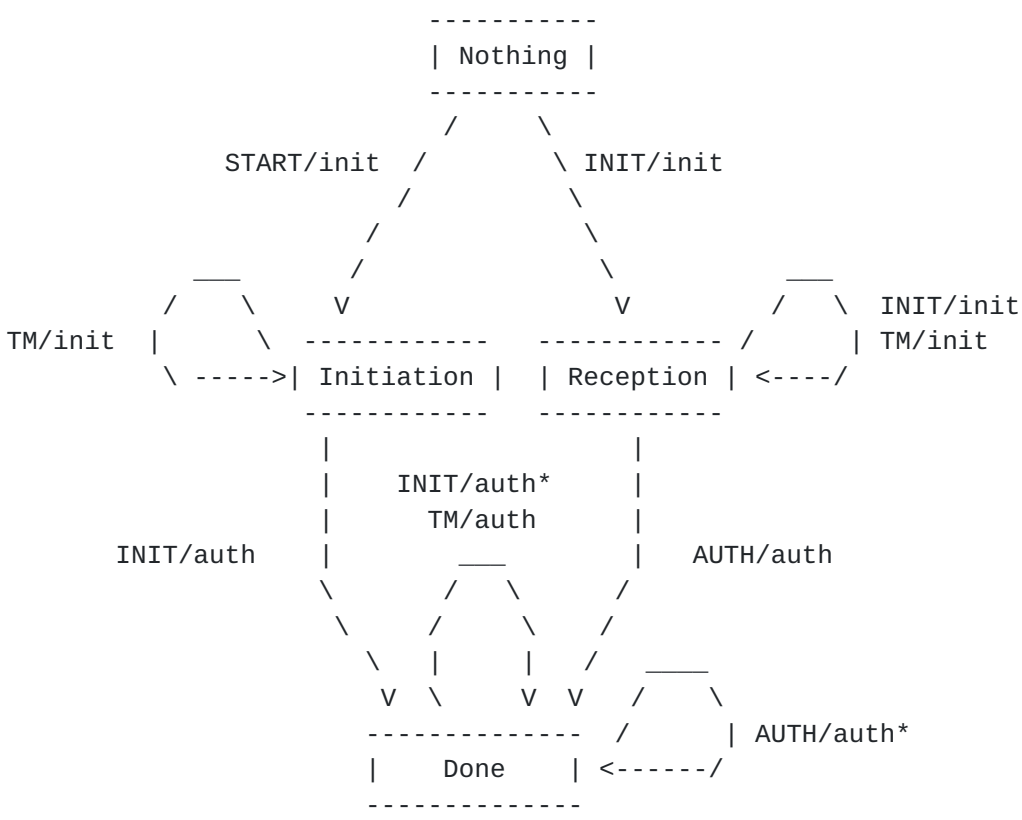

Figure 3: Protocol State Machine

6.3.3.1. Nothing State

Nothing state is the state in which an instance of the IKE state machine has just been created and has not received any events or performed any actions yet. Two events cause the state machine to exit Nothing state: a START event, and an INIT event.

When a state machine instance in Nothing state receives a START event the IKE peer initiates a connection to another peer. The information the IKE peer obtains as part of the START event is implementation specific but MUST idicate at a minimum the following:

- IP address of peer
- SPD information regarding the type of IPsec security association to form.

The method in which policy information regarding the type of authentication to propose, what group to use, etc., is out of scope of this memo. This information MUST be obtained but whether it is part of the START event indication or obtained as part of separate IKE configuration is irrelevant to the protocol.

The peer derives a session identifier, or SPI, to use as its transmitting SPI.

The peer retains the SPD information and SPI and constructs an Init message according Section 6.1.1.1 and transmits the message to the IP address of the peer. The state machine assigns the value zero (0) to the retrans counter, to the committed counter, and to the reauth counter. It sets the restransmission timer, and transitions to state Initiation.

When a state machine instance in Nothing state receives an INIT event, it signifies the reception of an Init message from a remote peer. The instance retains the IP address of the peer, extracts the transmitter's SPI from the message, and assigns the value zero (0) to the retrans counter, the committed counter and the reauth counter. It then processes the Init message according to Section 6.1.1.2. If processing of the Init message is successful, the instance generates an Init message for the peer according to Section 6.1.1.1, transmits the message to the peer, sets the retransmission timer and transitions to state Reception. If processing of the Init message is unsuccessful, the protocol instance remains in Nothing state and all state created as a result of receipt of the Init message MUST be deleted.

Note: a protocol instance that transition from Nothing state to Reception state has both received and sent an Initiate message. It MAY choose to finish the key exchange protocol and generate shared secret state according to the negotiated authentication method, or it may choose to delay such compuation until it receives an AUTH event in Reception state.

6.3.3.2. Initiation State

In Initiation state a protocol instance has initiated the IKE protocol to a peer. An INIT event causes the instance to leave Initiation state, and a TM event causes it to remain in Initiation state.

When a protocol instance in Initiation state receives an INIT event, it signifies receipt of an Initiate message from the peer. The protocol instance first cancels the retransmission timer and then processes the Init message according to Section 6.1.1.2. If

processing indicates that the message was discarded, the protocol instance sets the TM timer and remains in Initiation state. If processing indicates a failure, the protocol instance deletes all state it has created or retained and transitions back to Nothing state. Otherwise, processing is successful and the protocol instance shall finish the key exchange protocol and generate shared secret state according to the negotiated authentication method. It then increments the committed counter and generates an Auth message for the peer according to Section 6.1.2.1, transmits the message to the peer, sets the retransmission timer and transitions to state Done.

When a protocol instance in Initiation state receives a TM event it indicates that the retransmission timer has expired. If the retrans counter is higher than the retransmission threshold it indicates failure of the protocol. In this case the protocol instance deletes all state it has created or retained and transitions back to Nothing state. If the retrans counter is not greater than the retransmission threshhold, the Init message that was transmitted to the peer as part of transitioning into Initiation state is sent again to the peer, the retrans counter is incremented and the protocol instance remains in Initiation state.

6.3.3.3. Reception State

In Reception state a protocol instance is acting as the traditional "responder" in the IKE protocol. It has both sent and received an Init message. An AUTH event causes the instance to leave Reception state, and both an INIT and a TM event cause it to remain in Reception state.

When a protocol instance in Reception state receives an INIT event it signifies that the Init message it sent in order to transition into Reception state was not received by the peer. When it receives a TM event it indicates that its retransmission timer has expired. For an INIT event, the protocol instance first cancels the retransmission timer, after that the behavior a protocol instance takes is the same for an INIT or TM event. The instance retransmits the Init message it sent to the peer in order to transition in to Reception state, it sets its retransmission timer, and it remains in Reception state.

When a protocol instance in Reception state receives an AUTH event it signifies reception of an Auth message from its peer. First the protocol instance cancels its retransmission timer. Next, if the protocol instance has not finished the key exchange protocol (see the note in Section 6.3.3.1) and generated a shared secret it does so here. It then sets the reauth counter to the value of the "committed" field in the processed Auth messages, sets its committed counter to negative one (-1), generates an Auth message for the peer

according to Section 6.1.2.1, transmits the message to the peer, installs the IPsec SAs (per Section 6.2) and transitions to Done state. Note that the retransmission timer is not set.

6.3.3.4. Done State

Done state is the final state of the state machine. An initiator arrives in Done state after it has sent both of its messages to the peer and awaits a final Auth message. A responder arrives in Done state after it has sent and received both messages. To address the possibility of dropped packets and retransmission there are several events that can happen in Done state. Regardless of the event, though, after a state machine enters Done state it never leaves Done state.

When a protocol in Done state receives an INIT event, it signfies the receipt of a retransmitted Init message. Since the protocol has entered Done state it has already received and processed an Init message. If its committed counter is less than zero the protocol instance drops the Init message and remains in Done state. If its committed counter is not less than zero it cancels its retransmission timer, increments the committed counter, generates an Auth message, transmits the Auth message to the peer, sets its retransmission timer, and remains in Done state.

When a protocol in Done state receives a TM event, it signiifies that a previously sent Auth message has not been replied to in a timely manner. In this case, the protocol instance, checks the retransmission counter. If it is greater than the thresh counter the protocol instance destroys all state associated with the current run of the protocol (including any IPsec SAs that it might have installed) and transitions back to Nothing state. If the retransmission counter is not greater than the thresh counter, the protocol instance increments the retransmission counter, increments the committed counter, generates an Auth message, transmits the Auth message to the peer, sets the retransmission counter, and remains in Done state.

When a protocol instance in Done state receives an AUTH event it signifies the receipt of an Auth message from the peer. This could be in response to a message from the protocol instance or it could be a retransmission or the replay of an old Auth message. To prevent a storm of Auth messages going back and forth between protocol instances in Done state, the response to an AUTH message is conditional. If the committed counter is less than zero the protocol instance drops the received Auth message and remains in Done state. If the committed counter is not less than zero, the protocol instance cancels its retransmission timer and processes the Auth message. If

processing of the Auth message fails the protocol instance sets the retransmission timer and remains in Done state. If processing of the Auth message succeeds, the value of the "committed" field in the received Auth message is checked. If it is not numerically greater than the reauth counter the message is dropped, the retransmission timer is set, and the protocol instance remains in Done state. If the "committed" field in the received Auth message is numerically greater than the reauth counter, the reauth counter is set to the value in the "committed" field, the committed counter is set to negative one (-1), and an Auth message is generated. The protocol instance then sends the Auth message to the peer, and installs the IPsec SAs (per Section 6.2) and remains in Done state. Note that the retransmission timer is not set.

6.3.4. Cleaning Up Protocol Instances

The state machine ensures that once each side has sent and received an Auth message it installs IPsec SAs. To handle potential lost messages and retransmissions of its final Auth message a protocol instance remains in for a period of time after it has installed its IPsec SAs. These stale protocol instances can have their state deleted and transitioned back to Nothing state after a sufficient period of time. This memo does not define what "sufficient" means but suggests that after installing IPsec SAs, a protocol instance waits at least the amount of time it would spend before retranmissions would cause it to expire. That is:

```
wait = (thresh - retrans) * retrans-period
```

where "retrans-period" is the amount of time that the retransmission timer is set for. This will ensure that either the protocol instance expires because it retransmitted too many times or it will expire because the protocol has naturally completed.

6.4. IKEv3 Payload Formats

All messages in the IKEv3 protocol consist of an IKE header followed by additional payloads which define the semantics of the message. All payloads contain the same generic header. The IKE header indicates the first payload that follows and the generic header of each payload indicates the payload, if any, that follows. In this fashion payloads are chained together to form messages.

6.4.1. IKE header

The IKE header contains two message identifiers, called Security Parameter Indicies (SPIs), one for the transmitter and one for the receiver, an indicator of the first payload of the message,

versioning information, and the type of message, either an Init or an Auth. The format of the IKE header is shown in Figure 4.

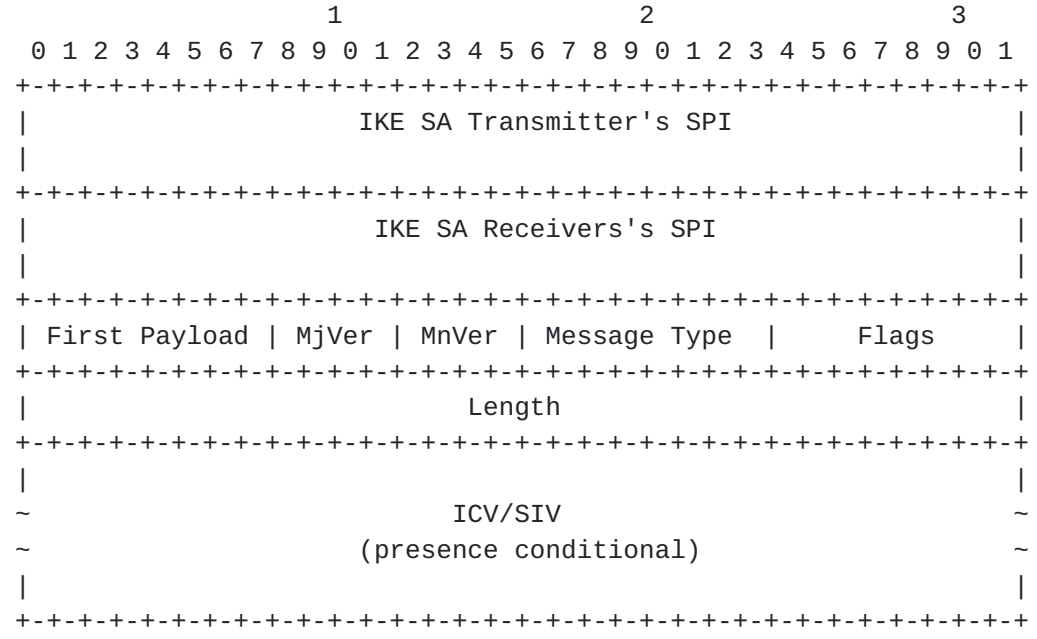

Figure 4: IKE Header Format

- Transmitter's SPI (8 octets) - A session identifier chosen by the transmitter of the message.

- Receiver's SPI (8 octets) - A session identifier chosen by the receiver of the message.

- First Payload (1 octet) - The type of payload that follows the header. See Figure 6.

- Major Version (4 bits) - The major version of this version of IKE. Implementations based on this memo MUST set the major version to three (3). Reciept of an IKE message with a different Major Version is governed by the memo which defines the version. An implementation that is only compliant with this version of IKE MUST drop any message with a Major version other than three (3).

- Minor Version (4 bits) - The minor version of this version of IKE. Implementations based on this memo MUST set the minor version to zero (0) on transmitted messages and ignore the Minor Version on received messages.

- Message Type (1 octet) - The type of message being transmitted: an Init message is type one (1) and an Auth message is type (2).

- Flags (1 octet) - A bitmask that indicates specific options for the message. The bits in this bitmask are as follows:

```
+-+-+-+-+-+-+-+-+
|X|X|X|V|X|X|X|S|
+-+-+-+-+-+-+-+-+
```

  Bits indicated as 'X' MUST be cleared on transmission and ignored on reception. Setting a bit to one (1) indicates that the option applies and clearing the bit to zero (0) indicates the option does not apply.

  * V (Version) - This bit indicates that the transmitter is capable of speaking a higher major version number of the IKE protocol than the one indicated in the major version field of this header.

  * S (Secured) - This bit indicates that the message following this header is authenticated and encrypted. When this bit is set the ICV/SIV field in the header is present.

- Length (4 octets) - an unsigned integer that indicates the length of the total IKE message (IKE header + all payloads) in octets. Note: if the 'E' bit in the Flags is set this length includes the conditional field to hold the Synthetic Initialization Vector/ MAC.

- ICV/SIV (variable) - a conditional field that is present when the message following the header is secured. This field is the byproduct of authenticated encryption and is required for verified decryption. The exact format of the ICV/SIV field depends on the type of authenticated encryption used by the peers.

6.4.2. Generic IKE payload header

The Generic IKE payload header is used to demark and chain all payloads in a message. Each payload used in a message contains this header. The Generic IKE payload header is defined in Figure 5.

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

Figure 5: IKE Header Format

- Next Payload (1 octet) - Indicates the payload, if any, that follows the current payload. See Figure 6.

- RESERVED (1 octet) - Unused by this version of IKE. It MUST be set to zero on all payloads in a transmitted message and ignored on all payloads in a received message.

- Payload Length (2 octets) - An unsigned integer indicating the entire length of the current payload, including this generic header.

Subsequently defined payloads are all shown with the generic header for completeness. Payload types listed here are current as of publication of this memo. Readers are encouraged to see [IKEV3IANA] for the latest values.

Payload Type	Notation	Value
No Next Payload		0
IKE Attributes Payload	IA	1
Identity Payload	ID	2
Nonce Payload	NO	3
Key Exchange Payload	KE	4
Certificate Request Payload	CR	5
Certificate Payload	CE	6
Authentication Payload	AU	7
Address Indication	AI	8
Traffic Selector	TS	9
Security Assocation Payload	SA	10
Vendor Indication	VE	11

Figure 6: IKEv3 Payload Assignment

The value "No Next Payload" SHALL only be used in the last payload of a message.

6.4.3. IKE Attributes payload

The IKE attributes payload lists a number of attributes that define the manner in which a run of the IKEv3 protocol occurs. Attributes consist of type-value tuples to identify the type of attribute and its particular value. These attributes are offered, not negotiated. See Section 6.1.1.2 for a description of the processing and potential rejection of an IA offer. The IA payload is defined in Figure 7.

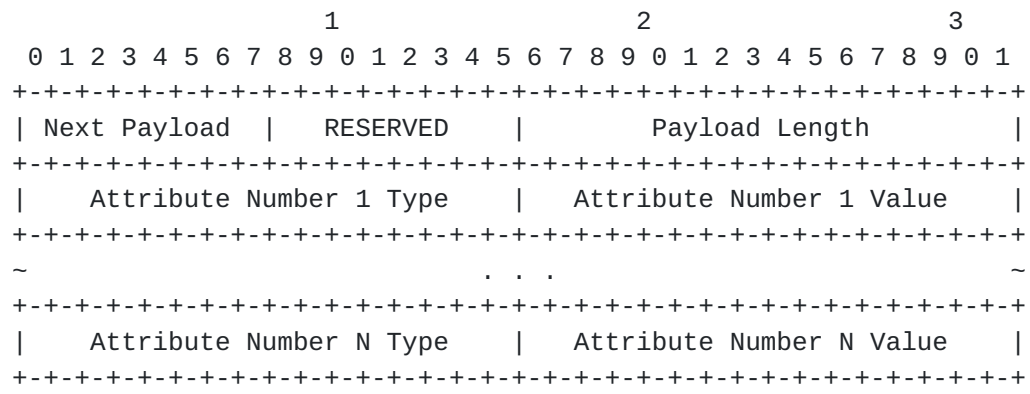

Figure 7: IA Payload

The following attributes types are defined and are indicated by assigning the indicated number to the Attributes Number <X> Type field:

1. Authentication Method

2. Authenticated Encryption Mode

3. Hash Algorithm

4. Diffie-Hellman Group

All other values are reserved to IANA.

When the Attribute Type indicates "Authentication Method", the following values are defined:

1. Digital Signatures

2. Pre-shared Key

All other values are reserved to IANA.

When the Attribute Type indicates "Authentication Encryption Mode", the following values are defined:

1. AES in Synthetic Initialization Mode ([RFC5297]) with a 256-bit key

2. AES in Synthetic Initialization Mode ([RFC5297]) with a 512-bit key

All other values are reserved to IANA.

When the Attribute Type indicates "Hash Algorithm, the following values are defined:

1. SHA-256 ([RFC4634])

2. SHA-512 ([RFC4634])

All other values are reserved to IANA.

When the Attribute Type indicates "Diffie-Hellman Group", the attribute values are taken from the "Diffie-Hellman Group Transform IDs" from [IKEV2IANA].

6.4.4. Identity Payload

The ID payload is used to convey the identity that is to be authenticated by the remote peer.

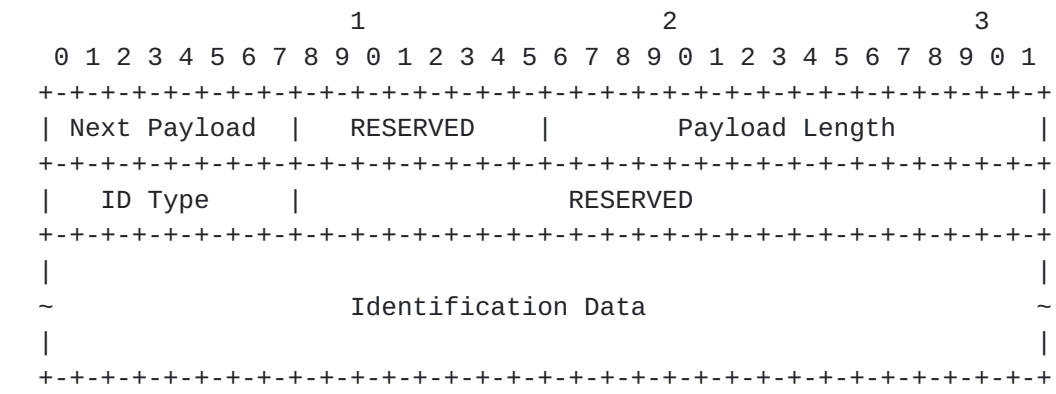

Figure 8: ID Payload

The following ID Types are defined:

ID Type Value		Description
1	ID_IPV4_ADDR	A single four (4) octet IPv4 address
2	ID_FQDN	A fully-qualified domain name string. An ID_FQDN string MUST NOT contain any terminators (e.g. NULL, CR, etc.). All characters in the ID_FQDN are ASCII.
3	ID_RFC822_ADDR	A fully-qualified RFC 822 email address string. An ID_RFC822_ADDR string MUST NOT contain any terminators (e.g. NULL, CR, etc.). This field SHOULD be treated as UTF-8 encoded text.
4	ID_IPV6_ADDR	A single sixteen (16) octet IPv6 address

5	ID_DER_ASN1_DN	The binary Distinguished Encoding Rules (DER) encoding of an ASN.1 X.500 Distinguished Name. See [RFC5280].
6	ID_DER_ASN1_GN	The binary DER encoding of an ASN.1 X.500 GeneralName. See [RFC5280].
7	ID_BLOB_ID	An opaque octet stream used for identity obfuscation. The two parties to the exchange MUST agree in an out-of-band fashion on how to map a Blob ID to an unobfuscated identity.

Table 1

Identification Data is a variable-length field that contains the identity of the specified type.

6.4.5. Nonce Payload

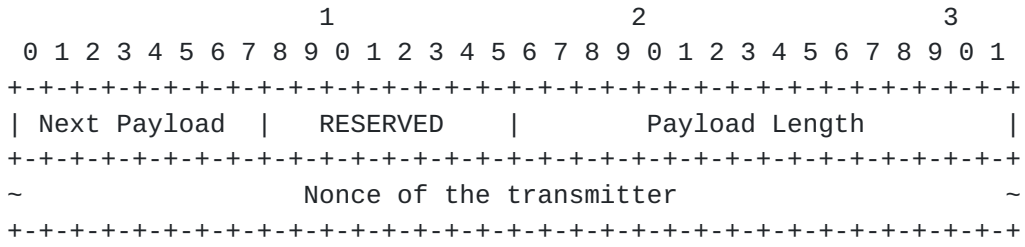

Figure 9: NO Payload

6.4.6. Key Exchange Payload

The KE payload is used to pass data used to perform the key exchange portion of the IKEv3 protocol (see Section 5). The body of the KE payload is authentication method specific. When doing The KE payload is authentication method specific. authentication using digital signatures, the body of the KE payload is a Diffie-Hellman public value. When doing PSK authentication, the body of the KE payload is a concatentation of the Commit and Confirm portions of the Dragonfly key exchange.

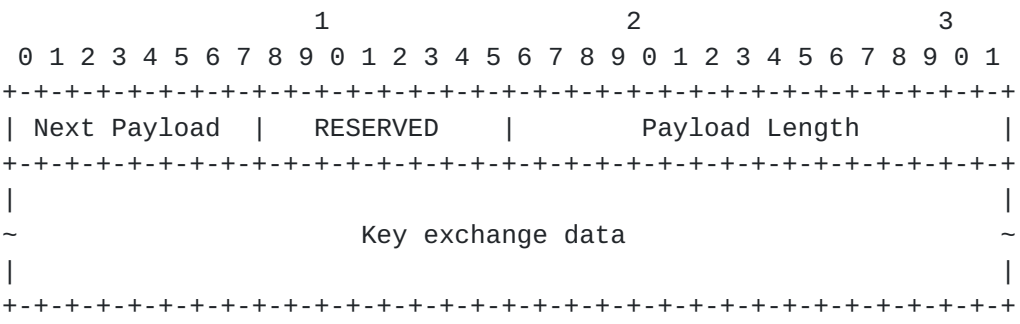

Figure 10: KE Payload

The key exhange data is a variable-length field that contains data to be transmitted to the remote peer to perform a cryptographic key exchange.

6.4.7. Certificate Payload

The CE payload is used to convey a public key which will be used for authentication to a remote peer. The public key can be certified or raw.

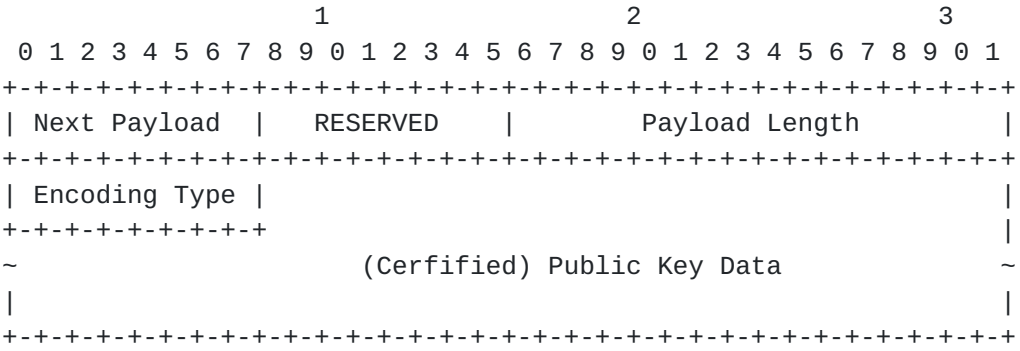

Figure 11: CE Payload

Encoding	Description
1	A DER-enocded X.509 certificate
2	Subject public key info for a raw key encoded according to [RFC5480]
3	Subject public key info for a raw key encoded accoding to [RFC3279]

Table 2

The Public key data is a variable-length field that contains the public key of the specified encoding.

6.4.8. Certificate Request Payload

The CR payload is used to request a certificate from a remote peer. The transmitter indicates a preference for the type of certificate using by setting the Encoding type according to Table 2. The transmitter SHOULD indicate a trusted Certification Authority for certified pubilc keys.

Figure 12: CR Payload

The Certificate Authority field, when present, is a variable-length field that contains the DER encoding of an ASN.1 X.509 IssuerName.

6.4.9. Authentication Payload

The AU payload contains data that authenticates a remote peer. The content of the body of an AU payload is either a digital signature (see Section 5.1.2) when authenticating with digital signatures, or a keyed message authentication function using the negotiated hash algorithm (see Section 5.2.4) when authenticating with a PSK.

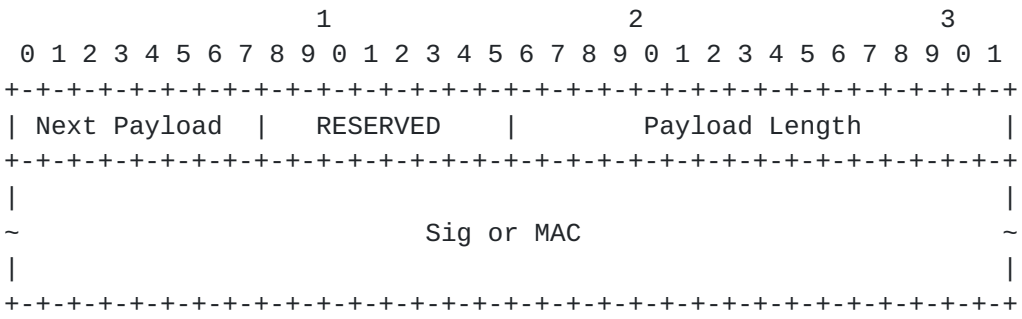

Figure 13: AU Payload

Sig or MAC is a variable-length field that contains either a digital signature of a message authentication code.

6.4.10. Address Indication Payload

The AI payload is used to convey the local view of addressing and port selection to the remote peer for the purposes of NAT detection.

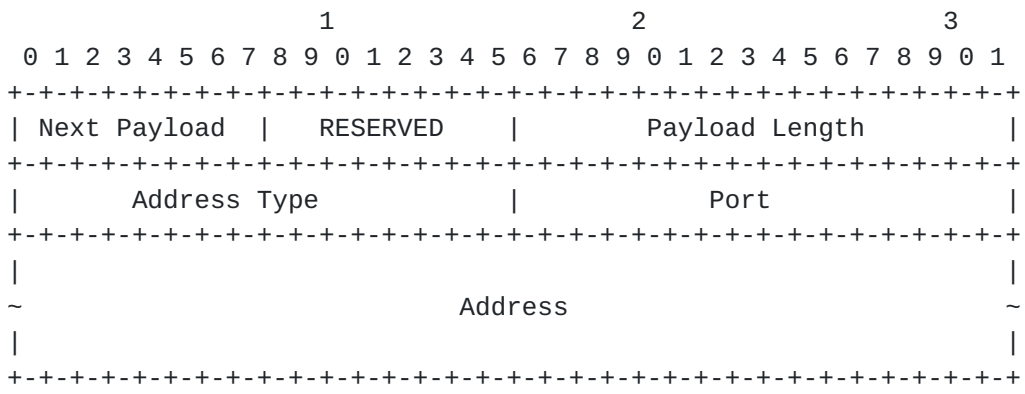

Figure 14: AI Payload

Address Types have the following meaning:

Address Type	Description
1	The Address field is a single four (4) octet IPv4 address
2	The Address field is a single sixteeen (16) octet IPv6 address

Table 3

All other Address Types are reserved and MUST NOT be used.

6.4.11. Traffic Selecor Payload

The TS payload is used to convey a set of traffic selectors used to identify traffic flows for processing by IPsec security services.

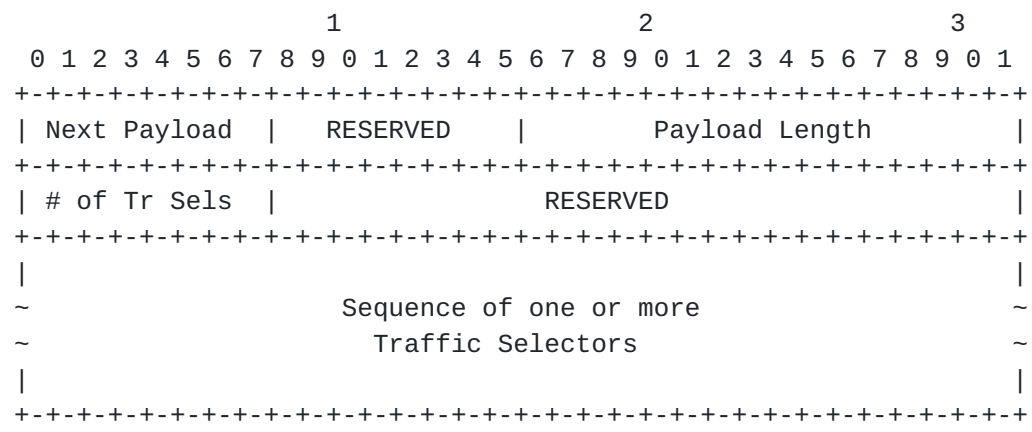

Figure 15: TS Payload

The traffic selectors conveyed to a peer are determined by the local Security Policy Database (see [RFC4301]). They define the data that MUST be protected by IPsec by individual flow. Each Traffic Selector in the set is defined by the following structure:

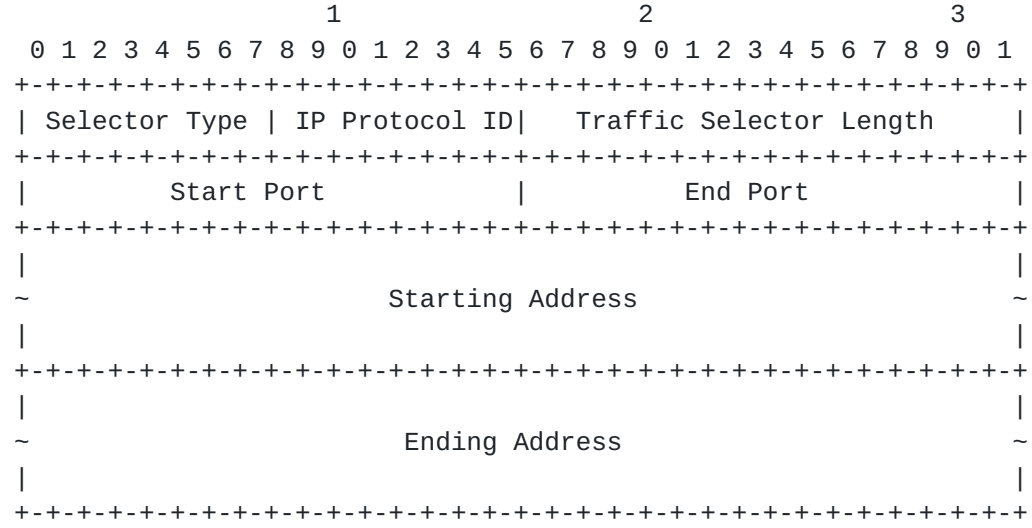

Figure 16: Traffic Selector

- Select Type (one octet) - either a one (1) to indicate an IPV4_ADDR_RANGE or two (2) to indicate an IPV6_ADDR_RANGE. All other types are invalid and MUST be rejected.

- IP Protocol ID (one octet) - indicates the associated IP protocol (such as UDP, TCP, and ICMP). A value of zero (0) means that the traffic selector convers all protocols.

- Traffic Selector Length (two octets) - the total length of the selector, including this sub header.

- Start Port (two octets) - the lowest port in a range of ports that are covered by this traffic selector. A value of zero (0) means that the traffic selector covers all ports. ICMP and ICMPv6 Type and Code values, as well as Mobile IP version 6 (MIPv6) mobility header (MH) Type values, are represented according to Section 4.4.1.1 of [RFC4301]. ICMP Type and Code values are treated as a single 16-bit integer port number, with Type in the most significant 8 bits and Code in the least significant 8 bits. MIPv6 MH Type and Code values are treated as a single 16-bit integer port number with Type in the most significant 8 bits and the least signficant 8 bits set to zero.

- End Port (two octets) - the highest port in a range of ports that are covered by this traffic selector. For protocols for which port is undefined (including protocol 0), or if all ports are allowed, this field MUST be 65535. ICMP and ICMPv6 Type and COde values, as well as MIPv6 MH Type values, are represented in this field as specified in Section 4.4.1.1 of [RFC4301]. ICMP Type and COde values are treated as a single 16-bit port number with Type in the most significant 8 bits and Code in the least significant 8 bits. MIPv6 MH Type values are treated as a single 16-bit integer port number with Type in the most significant 8 bit and the least significant 8 bits set to zero.

- Starting Address (variable) - the lowest address in a range of addresses that are covered by this traffic selector. This field will be either sixteen (16) octets or four (4) octets depending on whether the Selector Type was IPV6_ADDR_RANGE or IPV4_ADDR_RANGE, respectively.

- Ending Address (variable) - the highest address in a range of addresses that are covered by this traffic selector. This field will be either sixteen (16) octets or four (4) octets depending on whether the Selector Type was IPV6_ADDR_RANGE or IPV4_ADDR_RANGE, respectively.

6.4.12. Security Association Payload

The SA payload indicates the type of protection that IPsec will apply to the data that is covered by the Traffic Selector(s) in the TS payload. Protection is described as a number of attributes with each attribute consisting of a type-value tuple to identify the type of attribute and its particular value.

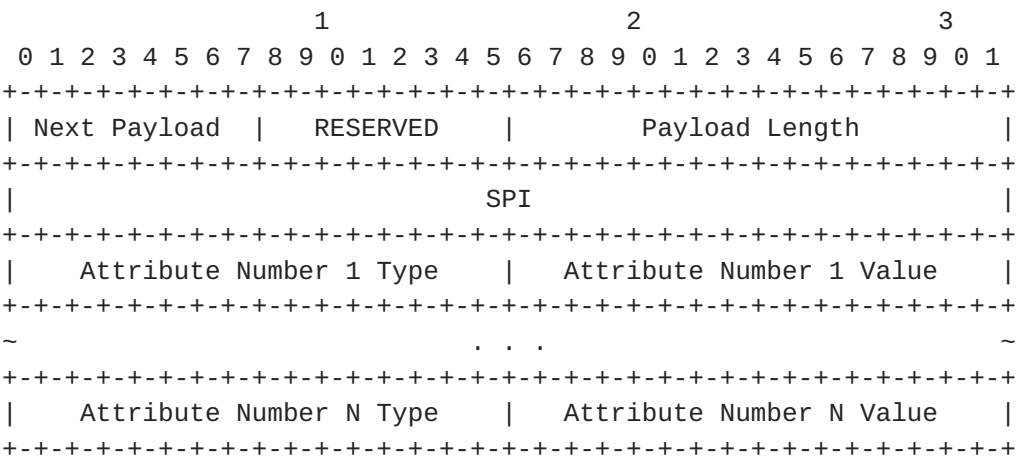

Figure 17: SA Payload

SPI (4 octets) - the Security Parameter Index that the transmitter of the SA payload will use to identify the resulting security association for received IPsec-protected packets.

The following attribute types are defined:

1. Encryption Algorithm

2. Integrity Algorithm

3. Extended Sequence Numbers

If the Encryption Algorithm attribute is present in an SA payload the SA SHALL be for ESP. If it is absent the SA SHALL be for AH. The Integrity Algorithm attribute is OPTIONAL for ESP and MANDATORY for AH. The Extended Sequence Number attribute is OPTIONAL.

The attribute values for the Encryption Algorithm attribute are defined in [IKEV2IANA] in the "Encryption Algorithm Transform IDs" table. The attribute values for the Integrity Algorithm attribute are defined in [IKEV2IANA] in the "Integrity Algorithm Transform IDs" table. The attribute values for the Extended Sequence Numbers attribute are defined in [IKEV2IANA] in the "Extended Sequence Numbers Transform IDs".

6.4.13. Vendor Indication Payload

The VI payload allows vendors of IKEv3 products to identify each other during the protocol.

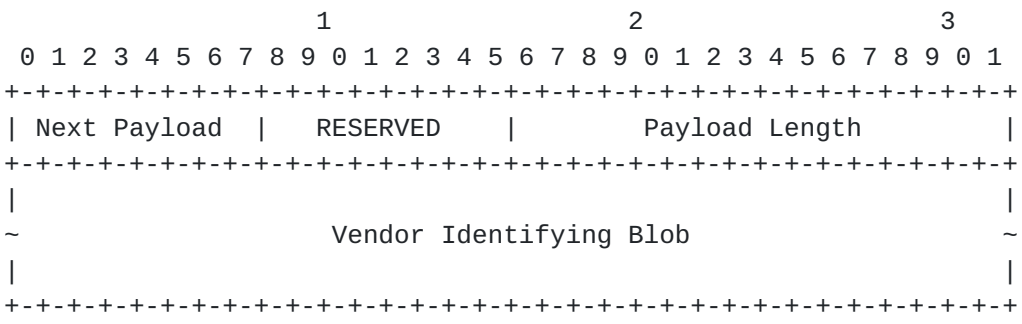

Figure 18: VI Payload

The Vendor Identifying Blob is a variable length field that contains information to identify the vendor of the transmitter of the payload. No registry for vendor identification is used and it is advised that vendors produce an opaque blob that will be different for each run of the protocol to identify themselves. This can be accomplished, for instance, by hashing the transmitter and receiver SPIs and/or the IP addresses of the peers with a vendor-specific constant.

7. Acknowledgements

Portions of the payload descriptions (e.g. Traffic Selector payload) were lifted from [RFC5996]. The author thanks the editors of that document and the IPsecME Working Group that produced that document.

8. IANA Considerations

This section is incomplete.

9. Security Considerations

This section is incomplete.

10. References

10.1. Normative References

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

[IKEV3IANA]
"Internet Key Exchange Version 3 (IKEv3) Parameters",

<http://www.iana.org>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, April 2002.

[RFC3948] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", RFC 3948, January 2005.

[RFC4634] Eastlake, D. and T. Hansen, "US Secure Hash Algorithms (SHA and HMAC-SHA)", RFC 4634, July 2006.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5297] Harkins, D., "Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES)", RFC 5297, October 2008.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, March 2009.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, February 2011.

10.2. Informative References

[RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[RFC4718] Eronen, P. and P. Hoffman, "IKEv2 Clarifications and Implementation Guidelines", RFC 4718, October 2006.

[RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 5996, September 2010.

Author's Address

Dan Harkins (editor)
Aruba Networks
1322 Crossman avenue
Sunnyvale, Californaia 94089
United States of America

Phone: +1 408 227 4500
Email: dharkins@arubanetworks.com