Internet Engineering Task Force (IETF)
Request for Comments: 7815
Category: Informational
ISSN: 2070-1721

T. Kivinen
INSIDE Secure

March 2016

# Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation

## Abstract

This document describes a minimal initiator version of the Internet Key Exchange version 2 (IKEv2) protocol for constrained nodes. IKEv2 is a component of IPsec used for performing mutual authentication and establishing and maintaining Security Associations (SAs). IKEv2 includes several optional features, which are not needed in minimal implementations. This document describes what is required from the minimal implementation and also describes various optimizations that can be done. The protocol described here is interoperable with a full IKEv2 implementation using shared secret authentication (IKEv2 does not require the use of certificate authentication). This minimal initiator implementation can only talk to a full IKEv2 implementation acting as the responder; thus, two minimal initiator implementations cannot talk to each other.

This document does not update or modify RFC 7296 but provides a more compact description of the minimal version of the protocol. If this document and RFC 7296 conflict, then RFC 7296 is the authoritative description.

## Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7815.

Table of Contents

## 1. Introduction

The Internet Protocol Suite is increasingly used on small devices with severe constraints on power, memory, and processing resources. This document describes a minimal IKEv2 implementation designed for use on such constrained nodes that is interoperable with "Internet Key Exchange Protocol Version 2 (IKEv2)" [RFC7296].

A minimal IKEv2 implementation only supports the initiator end of the protocol. It only supports the initial IKE_SA_INIT and IKE_AUTH exchanges and does not initiate any other exchanges. It also replies with an empty (or error) message to all incoming requests.

This means that most of the optional features of IKEv2 are left out: NAT traversal, IKE SA rekey, Child SA rekey, multiple Child SAs, deleting Child / IKE SAs, Configuration payloads, Extensible Authentication Protocol (EAP) authentication, COOKIEs, etc.

Some optimizations can be done because of the limited set of supported features, and this text should not be considered for generic IKEv2 implementations (for example, Message IDs can be done as specified because minimal implementation is only sending out an IKE_SA_INIT and IKE_AUTH request and not any other request).

This document is intended to be standalone, meaning everything needed to implement IKEv2 is copied here except the description of the cryptographic algorithms. The IKEv2 specification has lots of background information and rationale that has been omitted from this document.

Numerous additional numeric values from IANA registries have been omitted from this document; only those which are of interest for a minimal implementation are listed.

The main body of this document describes how to use the shared secret authentication in IKEv2, as it is easiest to implement. In some cases, that is not enough, and Appendix B.2 describes how to use raw public keys instead of shared secret authentication.

For more information, check the full IKEv2 specification in [RFC7296] and [IKEV2IANA].

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 [RFC2119]. The term "Constrained Node" is defined in "Terminology for Constrained-Node Networks" [RFC7228].

### 1.1. Use Cases

One use case for this kind of minimal implementation is in small devices doing machine-to-machine communication. In such environments, the node initiating connections can be very small, and the other end of the communication channel is some kind of larger device.

An example of the small initiating node could be a remote garage door opener device, i.e., a device having buttons that open and close a garage door and that connects to the home area network server over a wireless link.

Another example of such a device is some kind of sensor device, for example, a room temperature sensor, which sends periodic temperature data to some centralized node.

Those devices usually sleep for a long time and only wake up periodically or because of user interaction. The data transfer is always initiated from that sleeping node when they wake up; after they send packets, there might be ACKs or other packets coming back before they go back to sleep. If some data needs to be transferred from a server node to the small device, it can be implemented by polling, i.e., the small node periodically polls for the server to see if it, for example, has some configuration changes or similar. While the device is sleeping, it will not maintain the IKEv2 SA. That is, it will always create the IKEv2 SA again when it wakes up. This means there is no need to do liveness checks for the server, as after the device wakes up again, the minimal implementation will start from the beginning again.

## 2. Exchanges

### 2.1. Initial Exchange

All IKEv2 communications consist of pairs of messages: a request and a response. The pair is called an "exchange" and is sometimes called a "request/response pair". Every request requires a response.

For every pair of IKEv2 messages, the initiator is responsible for retransmission in the event of a timeout. The responder MUST never retransmit a response unless it receives a retransmission of the request.

IKEv2 is a reliable protocol: the initiator MUST retransmit a request until it either receives a corresponding response or deems the IKE SA to have failed. A retransmission from the initiator MUST be bitwise

identical to the original request. Retransmission times MUST increase exponentially.

IKEv2 is run over UDP port 500. All IKEv2 implementations MUST be able to send, receive, and process IKEv2 messages that are up to 1280 octets long. An implementation MUST accept incoming requests even if the source port is not 500 and MUST respond to the address and port from which the request was received.

The minimal implementation of IKEv2 only uses the first two exchanges, called IKE_SA_INIT and IKE_AUTH. These are used to create the IKE SA and the first Child SA. In addition to those messages, a minimal IKEv2 implementation needs to understand the CREATE_CHILD_SA request enough to generate a CREATE_CHILD_SA response containing the NO_ADDITIONAL_SAS error notify. It needs to understand the INFORMATIONAL request enough to generate an empty INFORMATIONAL response to it. There is no requirement to be able to respond to any other requests.

All messages following the IKE_SA_INIT exchange are cryptographically protected using the cryptographic algorithms and keys negotiated in the IKE_SA_INIT exchange.

Every IKEv2 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.

Minimal implementations only need to support the role of initiator, so it typically only sends an IKE_SA_INIT request that, when answered, is followed by an IKE_AUTH. As those messages have fixed Message IDs (0 and 1), it does not need to keep track of its own Message IDs for outgoing requests after that.

Minimal implementations can also optimize Message ID handling of the incoming requests, as they do not need to protect incoming requests against replays. This is possible because minimal implementations will only return error or empty notification replies to incoming requests. This means that any of those incoming requests do not have any effect on the minimal implementation, thus processing them again does not cause any harm. Because of this, a minimal implementation can always answer a request coming in, with the same Message ID than what the request had, and then forget the request/response pair immediately. This means there is no need to keep track of Message IDs of the incoming requests.

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

| Notation | Payload |
|---|---|
| AUTH | Authentication |
| CERTREQ | Certificate Request |
| D | Delete |
| HDR | IKE header (not a payload) |
| IDi | Identification - Initiator |
| IDr | Identification - Responder |
| KE | Key Exchange |
| Ni, Nr | Nonce |
| N | Notify |
| SA | Security Association |
| SK | Encrypted and Authenticated |
| TSi | Traffic Selector - Initiator |
| TSr | Traffic Selector - Responder |

The initial exchanges are as follows:

```
Initiator                           Responder
-----------------------------------------------------------------
HDR(SPIi=xxx, SPIr=0, IKE_SA_INIT,
    Flags: Initiator, Message ID=0),
    SAi1, KEi, Ni  -->

                    <--  HDR(SPIi=xxx, SPIr=yyy, IKE_SA_INIT,
                             Flags: Response, Message ID=0),
                             SAr1, KEr, Nr, [CERTREQ]
```

HDR contains the Security Parameter Indexes (SPIs), version numbers, and flags of various sorts. Each endpoint chooses one of the two SPIs and MUST choose them so as to be unique identifiers of an IKE SA. An SPI value of zero is special: it indicates that the remote SPI value is not yet known by the sender.

Incoming IKEv2 packets are mapped to an IKE SA using only the packet's SPI, not using (for example) the source IP address of the packet.

The SAi1 payload states the cryptographic algorithms the initiator supports for the IKE SA. The KEi and KEr payloads contain Diffie-Hellman values, and Ni and Nr are the nonces. The SAr1 contains the chosen cryptographic suite from the initiator's offered choices. A minimal implementation using shared secrets will ignore the CERTREQ payload.

Minimal implementation will most likely support exactly one set of cryptographic algorithms, meaning the SAi1 payload will be static. It needs to check that the SAr1 received matches the proposal it sent.

At this point in the negotiation, each party can generate SKEYSEED, from which all keys are derived for that IKE SA.

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

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

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

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

(indicating that the quantities SK_d, SK_ai, SK_ar, SK_ei, SK_er, SK_pi, and SK_pr are taken in order from the generated bits of the prf+). g^ir is the shared secret from the ephemeral Diffie-Hellman exchange. g^ir is represented as a string of octets in big endian order padded with zeros if necessary to make it the length of the modulus. Ni and Nr are the nonces, stripped of any headers.

The SK_d is used for deriving new keys for the Child SAs. The SK_ai and SK_ar are used as a key to the integrity protection algorithm for authenticating the component messages of subsequent exchanges. The SK_ei and SK_er are used for encrypting (and of course decrypting) all subsequent exchanges. The SK_pi and SK_pr are used when generating an AUTH payload. The lengths of SK_d, SK_pi, and SK_pr MUST be the preferred key length of the Pseudorandom Function (PRF) agreed upon.

A separate SK_e and SK_a is computed for each direction. The keys used to protect messages from the original initiator are SK_ai and SK_ei. The keys used to protect messages in the other direction are SK_ar and SK_er. The notation SK { ... } indicates that these payloads are encrypted and integrity protected using that direction’s SK_e and SK_a.

```
Initiator                          Responder
-------------------------------------------------------------------
HDR(SPIi=xxx, SPIr=yyy, IKE_AUTH,
    Flags: Initiator, Message ID=1),
    SK {IDi, AUTH, SAi2, TSi, TSr,
        N(INITIAL_CONTACT)}  -->

                  <--  HDR(SPIi=xxx, SPIr=yyy, IKE_AUTH, Flags:
                             Response, Message ID=1),
                             SK {IDr, AUTH, SAr2, TSi, TSr}
```

The initiator asserts its identity with the IDi payload, proves knowledge of the secret corresponding to IDi, and integrity protects the contents of the first message using the AUTH payload. The responder asserts its identity with the IDr payload, authenticates its identity, and protects the integrity of the second message with the AUTH payload.

As minimal implementation usually has only one host where it connects, that means it has only one shared secret. This means it does not need to care about the IDr payload that much. If the other end sends an AUTH payload that the initiator can verify using the shared secret it has, then it knows the other end is the peer it was configured to talk to.

In the IKE_AUTH request, the initiator sends the SA offer(s) in the SAi2 payload and the proposed Traffic Selectors (TSs) for the Child SA in the TSi and TSr payloads. The responder replies with the accepted offer in an SAr2 payload and with the selected Traffic Selectors. The selected Traffic Selectors may be a subset of what the initiator proposed.

In the minimal implementation, both SA payloads and TS payloads are going to be mostly static. The SA payload will have the SPI value used in the Encapsulating Security Payload (ESP), but the algorithms are most likely going to be the one and only supported set. The TS payloads on the initiator end will most likely say from any to any, i.e., full wildcard ranges, or from the local IP to the remote IP. In the wildcard case, the responder quite often narrows the range down to the one IP address pair. Using a single IP address pair as the Traffic Selectors when sending the IKE_AUTH request will simplify processing as the responder will either accept the IP address pair or return an error. If wildcard ranges are used, there is a possibility that the responder will narrow the Traffic Selector range to range that is not acceptable by the initiator.

The IKE_AUTH (and IKE_SA_INIT) response may contain multiple status notification payloads that can be ignored by minimal implementations.

There can also be Vendor ID, Certificate, Certificate Request, or Configuration payloads, but any payload unknown to minimal implementations can simply be skipped over (response messages cannot have critical unsupported payloads).

The exchange above includes N(INITIAL_CONTACT) notification in the request as that is quite commonly sent by a minimal implementation. It indicates to the other end that the initiator does not have any other IKE SAs between it and the responder, and if there is any left from previous runs, those can be deleted by the responder. As minimal implementations delete IKE SAs without sending IKE SA delete requests, this will help the responder to clean up leftover state.

When using shared secret authentication, the peers are authenticated by having each calculating a Message Authentication Code (MAC) over a block of data:

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

The string "Key Pad for IKEv2" is 17 ASCII characters without null termination. The implementation can precalculate the inner prf and only store the output of it. This is possible because a minimal IKEv2 implementation usually only supports one PRF.

In the following calculations, IDi' and IDr' are the entire ID payloads excluding the fixed header, and the Ni and Nr are only the values, not the payloads containing it. Note that neither the nonce Ni/Nr nor the value prf(SK_pr, IDr')/prf(SK_pi, IDi') are transmitted.

The initiator signs the first message (IKE_SA_INIT request), starting with the first octet of the first SPI in the header and ending with the last octet of the last payload in that first message. Appended to this (for purposes of computing the signature) are the responder's nonce Nr and the value prf(SK_pi, IDi').

For the responder, the octets to be signed start with the first octet of the first SPI in the header of the second message (IKE_SA_INIT response) and end with the last octet of the last payload in that second message. Appended to this are the initiator's nonce Ni and the value prf(SK_pr, IDr').

The initiator's signed octets can be described as:

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

The responder's signed octets can be described as:

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

Note that all of the payloads inside the RestOfMessageX are included under the signature, including any payload types not listed in this document.

The initiator might also get an unauthenticated response back that has a notification payload with an error code inside. As that error code will be unauthenticated and may be faked, there is no need to do anything for those. A minimal implementation can simply ignore those errors and retransmit its request until it times out, and if that happens, then the IKE SA (and Child SA) creation failed.

The responder might also reply with an IKE_AUTH response packet that does not contain the payloads needed to set up a Child SA (SAr2, TSi, and TSr) but instead contain AUTH payload and an error. Minimal implementation that does not support the CREATE_CHILD_SA exchange cannot recover from this scenario. It can delete the IKE SA and start over from the beginning (which might fail again if this is a configuration error, or it might succeed if this was temporal failure).

## 2.2. Other Exchanges

Minimal implementations MUST be able to reply to INFORMATIONAL requests by sending back an empty INFORMATIONAL response:

```
Minimal implementation           Other end
-------------------------------------------------------------------
                   <--  HDR(SPIi=xxx, SPIr=yyy, INFORMATIONAL,
                               Flags: none,  Message ID=m),
                               SK {...}

HDR(SPIi=xxx, SPIr=yyy, INFORMATIONAL,
    Flags: Initiator | Response,
    Message ID=m),
    SK {}  -->
```

Minimal implementations MUST be able to reply to incoming CREATE_CHILD_SA requests. A typical implementation will reject the CREATE_CHILD_SA exchanges by sending a NO_ADDITIONAL_SAS error notify back:

```
Minimal implementation           Other end
-------------------------------------------------------------------
                   <--  HDR(SPIi=xxx, SPIy=yyy, CREATE_CHILD_SA,
                               Flags: none, Message ID=m),
                               SK {...}

HDR(SPIi=xxx, SPIr=yyy, CREATE_CHILD_SA,
    Flags: Initiator | Response, Message ID=m),
    SK {N(NO_ADDITIONAL_SAS)}  -->
```

Note that INFORMATIONAL and CREATE_CHILD_SA requests might contain unsupported critical payloads, in which case a compliant implementation MUST ignore the request and send a response message back that has the UNSUPPORTED_CRITICAL_PAYLOAD notification. That notification payload data contains a 1-octet payload type of the unsupported critical payload.

## 2.3. Generating Keying Material

The keying material for the Child SA created by the IKE_AUTH exchange is generated as follows:

```
KEYMAT = prf+(SK_d, Ni | Nr)
```

Where Ni and Nr are the nonces from the IKE_SA_INIT exchange.

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

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

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

Each cryptographic algorithm takes a fixed number of bits of keying material specified as part of the algorithm or negotiated in SA payloads.

## 3. Conformance Requirements

For an implementation to be called conforming to the RFC 7296 specification, it MUST be possible to configure it to accept the following:

- Public Key Infrastructure using X.509 (PKIX) Certificates containing and signed by RSA keys of size 1024 or 2048 bits, where the ID passed is any of ID_KEY_ID, ID_FQDN, ID_RFC822_ADDR, or ID_DER_ASN1_DN.

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

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

This document only supports the second bullet; it does not support PKIX Certificates at all. As full RFC 7296 responders must also support that shared key authentication, this allows a minimal implementation to be able to interoperate with all implementations that are compliant with RFC 7296.

PKIX Certificates are left out from the minimal implementation as those would add quite a lot of complexity to the implementation. The actual code changes needed in the IKEv2 protocol are small, but the certificate validation code would be more complex than the whole

minimal IKEv2 implementation itself. If public-key-based authentication is needed for scalability reasons, then raw public keys would probably be the best compromise (see Appendix B.2).

## 4. Implementation Status

This document describes a minimal implementation written by the author of this document. The minimal implementation supported the base IKE_SA_INIT and IKE_AUTH exchanges and successfully interoperated with a full IKEv2 server. This minimal implementation was presented in the Interconnecting Smart Objects with Internet Workshop in Prague in March 2011 [Kiv11]. This implementation was written as proof of concept in perl.

There was another proof-of-concept implementation written in python, which also interoperated with a full IKEv2 server.

Both implementations were written just for demonstration purposes and included fixed configuration built into the code, and both also implemented ESP, ICMP, and IP layers to the level that was needed to send and receive one ICMP echo packet. Both implementations were about 1000 lines of code excluding cryptographic libraries but including ESP, ICMP, and IP layers.

## 5. Security Considerations

As this implements the same protocol as RFC 7296, this means all security considerations from it also apply to this document.

# 6. References

## 6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <http://www.rfc-editor.org/info/rfc7296>.

## 6.2. Informative References

[EAI] Yang, A., Steele, S., and N. Freed, "Internationalized Email Headers", RFC 6532, DOI 10.17487/RFC6532, February 2012, <http://www.rfc-editor.org/info/rfc6532>.

[IDNA] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <http://www.rfc-editor.org/info/rfc5890>.

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

[IPSEARCH] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <http://www.rfc-editor.org/info/rfc4301>.

[Kiv11] Kivinen, T., "Interconnecting Smart Objects with Internet Workshop 2011-03025; IKEv2 and Smart Objects", March 2011, <https://www.iab.org/wp-content/IAB-uploads/2011/04/Kivinen.pdf>.

[MODES] National Institute of Standards and Technology, U.S. Department of Commerce, "Recommendation for Block Cipher Modes of Operation", SP 800-38A, 2001.

[PKCS1] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, DOI 10.17487/RFC3447, February 2003, <http://www.rfc-editor.org/info/rfc3447>.

[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, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <http://www.rfc-editor.org/info/rfc5322>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <http://www.rfc-editor.org/info/rfc7228>.

[RFC7619] Smyslov, V. and P. Wouters, "The NULL Authentication Method in the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 7619, DOI 10.17487/RFC7619, August 2015, <http://www.rfc-editor.org/info/rfc7619>.

[RFC7670] Kivinen, T., Wouters, P., and H. Tschofenig, "Generic Raw Public-Key Support for IKEv2", RFC 7670, DOI 10.17487/RFC7670, January 2016, <http://www.rfc-editor.org/info/rfc7670>.

## Appendix A. Header and Payload Formats

This appendix describes actual packet payload formats. This is required to make the document self-contained. The descriptions are mostly copied from RFC 7296, and more information can be found from there.

Various payloads contain RESERVED fields, and those MUST be sent as zero and MUST be ignored on receipt.

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

### A.1. The IKE Header

Each IKEv2 message begins with the IKE header, denoted HDR in this document. Following the header are one or more IKE payloads each identified by a Next Payload field in the preceding payload. Payloads are identified in the order in which they appear in an IKE message by looking in the Next Payload field in the IKE header and, subsequently, according to the Next Payload field in the IKE payload itself until a Next Payload field of zero indicates that no payloads follow. If a payload of type "Encrypted" is found, that payload is decrypted and its contents parsed as additional payloads. An Encrypted payload MUST be the last payload in a packet, and an Encrypted payload MUST NOT contain another Encrypted payload.

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

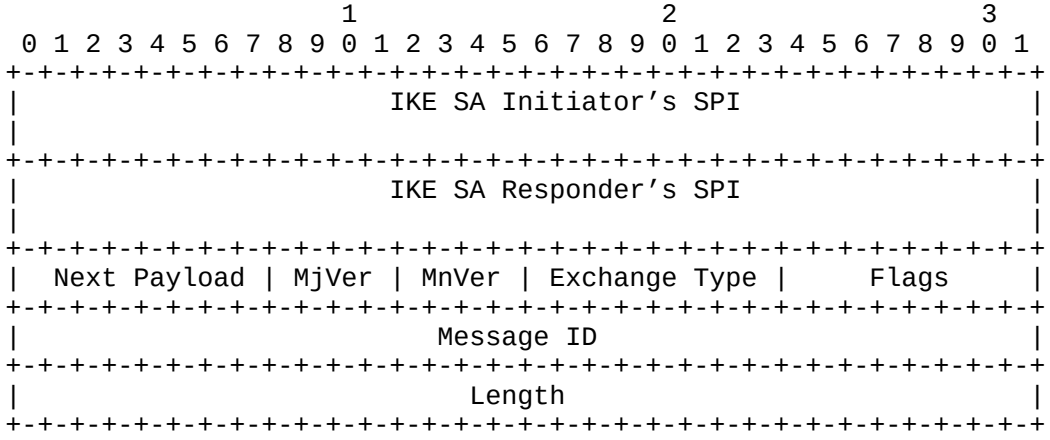


Figure 1: IKE Header Format

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

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

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

- Major Version (4 bits) - Indicates the major version of the IKE protocol in use. Implementations based on this version of IKE MUST set the major version to 2 and MUST drop the messages with a higher major version number.

- 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 zero. They MUST ignore the minor version number of received messages.

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

| Exchange Type | Value |
|---|---|
| IKE_SA_INIT | 34 |
| IKE_AUTH | 35 |
| CREATE_CHILD_SA | 36 |
| INFORMATIONAL | 37 |

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

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

* R (Response) - This bit indicates that this message is a response to a message containing the same Message ID. This bit MUST be cleared in all request messages and MUST be set in all responses. An IKEv2 endpoint MUST NOT generate a response to a message that is marked as being a response.

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

* I (Initiator) - This bit MUST be set in messages sent by the original initiator of the IKE SA and MUST be cleared in messages sent by the original responder. It is used by the recipient to determine which 8 octets of the SPI were generated by the recipient. This bit changes to reflect who initiated the last rekey of the IKE SA.

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

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

## A.2. Generic Payload Header

Each IKE payload begins with a generic payload header, as shown in Figure 2. Figures for each payload below will include the generic payload header, but for brevity, the description of each field will be omitted.

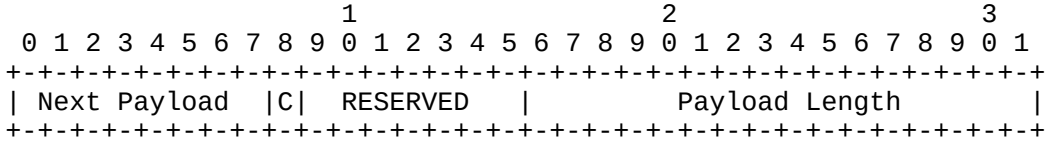


Figure 2: Generic Payload Header

The Generic Payload Header fields are defined as follows:

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

| Next Payload Type | Notation | Value |
|---|---|---|
| No Next Payload | | 0 |
| Security Association | SA | 33 |
| Key Exchange | KE | 34 |
| Identification - Initiator | IDi | 35 |
| Identification - Responder | IDr | 36 |
| Certificate | CERT | 37 |
| Certificate Request | CERTREQ | 38 |
| Authentication | AUTH | 39 |
| Nonce | Ni, Nr | 40 |
| Notify | N | 41 |
| Delete | D | 42 |
| Traffic Selector - Initiator | TSi | 44 |
| Traffic Selector - Responder | TSr | 45 |
| Encrypted and Authenticated | SK | 46 |

- Critical (1 bit) - MUST be set to zero if the sender wants the recipient to skip this payload if it does not understand the payload type code in the Next Payload field of the previous payload. MUST be set to 1 if the sender wants the recipient to reject this entire message if it does not understand the payload type. MUST be ignored by the recipient if the recipient understands the payload type code. MUST be set to zero for payload types defined in this document. Note that the critical bit applies to the current payload rather than the "next" payload whose type code appears in the first octet.

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

### A.3. Security Association Payload

The Security Association payload, denoted SA in this document, is used to negotiate attributes of a Security Association.

An SA payload consists of one or more proposals. Each proposal includes one protocol. Each protocol contains one or more transforms -- each specifying a cryptographic algorithm. Each transform contains zero or more attributes (attributes are needed only if the Transform ID does not completely specify the cryptographic algorithm; currently, the only attribute is the Key Length attribute for variable-length ciphers, meaning there is exactly zero or one attribute).

The responder MUST choose a single suite, which may be any subset of the SA proposal following the rules below.

Each proposal contains one protocol. If a proposal is accepted, the SA response MUST contain the same protocol. Each IPsec protocol proposal contains one or more transforms. Each transform contains a Transform Type. The accepted cryptographic suite MUST contain exactly one transform of each type included in the proposal. For example: if an ESP proposal includes transforms ENCR_3DES, ENCR_AES w/keysize 128, ENCR_AES w/keysize 256, AUTH_HMAC_MD5, and AUTH_HMAC_SHA, the accepted suite MUST contain one of the ENCR_ transforms and one of the AUTH_ transforms. Thus, six combinations are acceptable.

Minimal implementation can create very simple SA proposal, i.e., include one proposal, which contains exactly one transform for each Transform Type. It is important to only include one Diffie-Hellman group in the proposal, so there is no need to do INVALID_KE_PAYLOAD processing in responses.

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.

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 two transforms: Extended Sequence Numbers (ESNs) and an integrity check algorithm.

ESP generally has three: ESN, an encryption algorithm, and an integrity check algorithm. IKEv2 generally has four transforms: a Diffie-Hellman group, an integrity check algorithm, a PRF algorithm, and an encryption algorithm. For each Protocol, the set of permissible transforms is assigned Transform ID numbers, which appear in the header of each transform.

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

A given transform MAY have one or more Attributes. Attributes are necessary when the transform can be used in more than one way, as when an encryption algorithm has a variable key size. The transform would specify the algorithm, and the attribute would specify the key size. To propose alternate values for an attribute (for example, multiple key sizes for the AES encryption algorithm), an implementation MUST include multiple transforms with the same Transform Type each with a single Attribute.

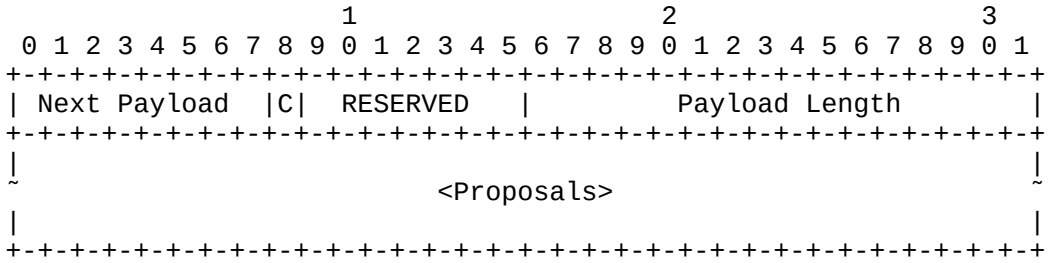


Figure 3: Security Association Payload

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

### A.3.1. Proposal Substructure

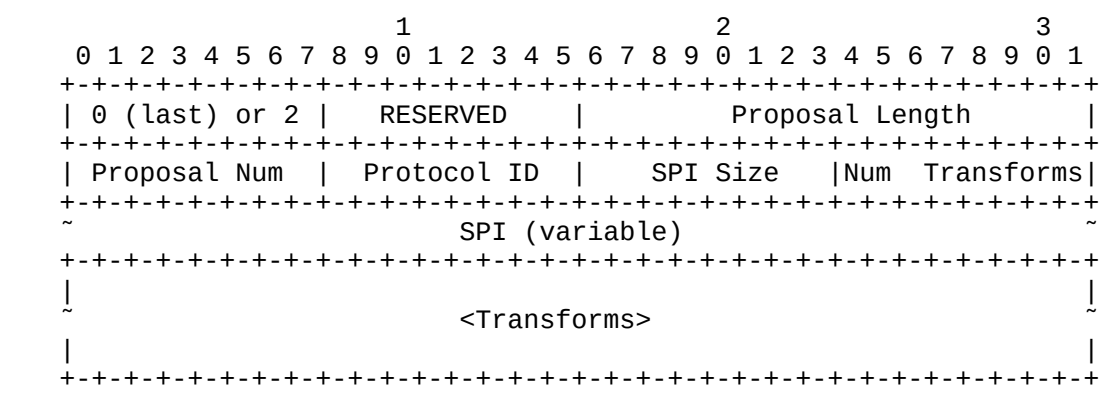


Figure 4: Proposal Substructure

- 0 (last) or 2 (more) (1 octet) - Specifies whether this is the last Proposal Substructure in the SA.

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

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

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

| Protocol | Protocol ID |
|---|---|
| IKE | 1 |
| AH | 2 |
| ESP | 3 |

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

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

- SPI (variable) - The sending entity's SPI. When the SPI Size field is zero, this field is not present in the Security Association payload.

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

### A.3.2. Transform Substructure

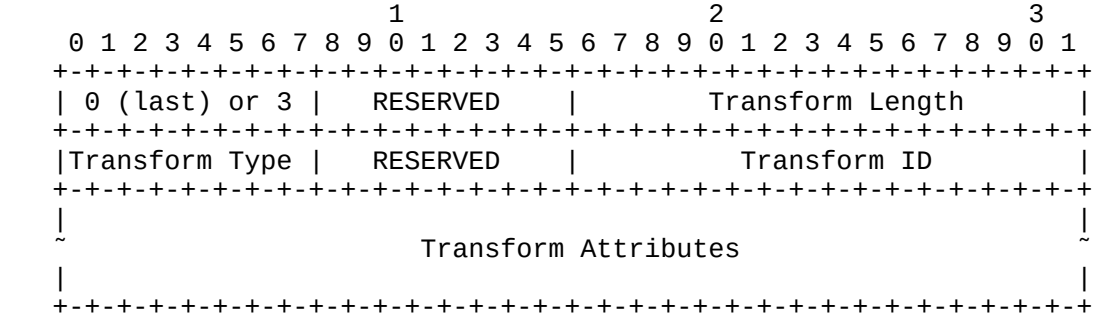


Figure 5: Transform Substructure

- 0 (last) or 3 (more) (1 octet) - Specifies whether this is the last Transform Substructure in the Proposal.

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

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

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

The relevant Transform Type values are listed below. For more information see [RFC7296].

| Description | Trans. Type | Used In |
|---|---|---|
| Encryption Algorithm (ENCR) | 1 | IKE and ESP |
| Pseudorandom Function (PRF) | 2 | IKE |
| Integrity Algorithm (INTEG) | 3 | IKE, AH, optional in ESP |
| Diffie-Hellman group (D-H) | 4 | IKE, optional in AH & ESP |
| Extended Sequence Numbers (ESN) | 5 | AH and ESP |

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

| Name | Number |
|---|---|
| ENCR_AES_CBC | 12 |
| ENCR_AES-CCM_8 | 14 |

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

| Name | Number |
|---|---|
| PRF_HMAC_SHA1 | 2 |

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

| Name | Number |
|---|---|
| AUTH_HMAC_SHA1_96 | 2 |
| AUTH_AES_XCBC_96 | 5 |

For Transform Type 4 (Diffie-Hellman group), the relevant Transform IDs are listed below.

| Name | Number |
|---|---|
| 1536-bit MODP | 5 |
| 2048-bit MODP | 14 |

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

| Name | Number |
|---|---|
| No Extended Sequence Numbers | 0 |
| Extended Sequence Numbers | 1 |

Note that an initiator who supports ESNs will usually include two ESN transforms, with values "0" and "1", in its proposals. A proposal containing a single ESN transform with value "1" means that using normal (non-extended) sequence numbers is not acceptable.

### A.3.3. Valid Transform Types by Protocol

The number and type of transforms that accompany an SA payload are dependent on the protocol in the SA itself. An SA payload proposing the establishment of an SA has the following mandatory and optional Transform Types. A compliant implementation MUST understand all mandatory and optional types for each protocol it supports (though it need not accept proposals with unacceptable suites). A proposal MAY omit the optional types if the only value for them it will accept is NONE.

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

### A.3.4. Transform Attributes

Transform Type 1 (Encryption Algorithm) transforms might include one transform attribute: Key Length.

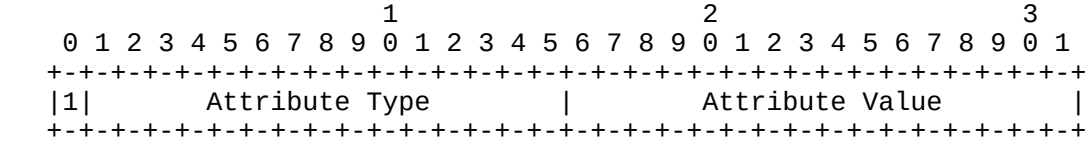


Figure 6: Data Attributes

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

- Attribute Value - Value of the attribute associated with the attribute type.

| Attribute Type | Value |
|---|---|
| Key Length (in bits) | 14 |

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

- The Key Length attribute MUST NOT be used with transforms that use a fixed-length key.
- Some transforms specify that the Key Length attribute MUST be always included. For example, ENCR_AES_CBC.

## A.4. Key Exchange Payload

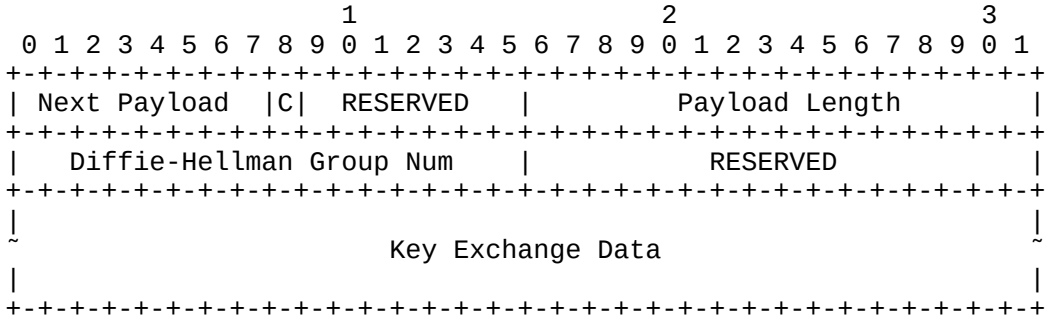


Figure 7: Key Exchange Payload Format

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 for modular exponentiation groups (MODPs) MUST be equal to the length of the prime modulus over which the exponentiation was performed, prepending zero bits to the value if necessary.

The Diffie-Hellman Group Num identifies the Diffie-Hellman group in which the Key Exchange Data was computed. This Diffie-Hellman Group Num MUST match a Diffie-Hellman group specified in a proposal in the SA payload that is sent in the same message.

## A.5. Identification Payloads

The Identification payloads, denoted IDi and IDr in this document, allow peers to assert an identity to one another. When using the ID_IPV4_ADDR/ID_IPV6_ADDR identity types in IDi/IDr payloads, IKEv2 does not require this address to match the address in the IP header of IKEv2 packets or anything in the TSi/TSr payloads. The contents

of IDi/IDr are used purely to fetch the policy and authentication data related to the other party. In minimal implementation, it might be easiest to always use KEY_ID type. This allows the ID payload to be static. Using an IP address has problems in environments where IP addresses are dynamically allocated.

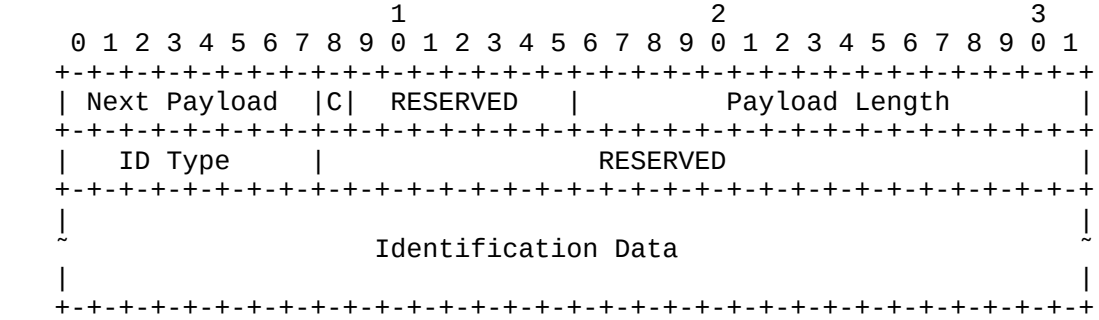


Figure 8: Identification Payload Format

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

- 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 following table lists the assigned semantics for the Identification Type field.

| ID Type | Value |
|---|---|
| ID_IPV4_ADDR | 1 |
| A single four (4) octet IPv4 address. | |
| ID_FQDN | 2 |
| A fully qualified domain name string. An example of an ID_FQDN is "example.com". The string MUST NOT contain any terminators (e.g., NULL, CR, etc.). All characters in the ID_FQDN are ASCII; for an "internationalized domain name", the syntax is as defined in [IDNA], for example, "xn--tmonesimerkki-bfbb.example.net". | |
| ID_RFC822_ADDR | 3 |
| A fully qualified RFC 822 email address string based [RFC5322]. An example of an ID_RFC822_ADDR is "jsmith@example.com". The string MUST NOT contain any terminators. Because of [EAI], implementations would be wise to treat this field as UTF-8-encoded text, not as pure ASCII. | |

ID_IPV6_ADDR 5
A single sixteen (16) octet IPv6 address.

ID_KEY_ID 11
An opaque octet stream that may be used to pass vendor-specific information necessary to do certain proprietary types of identification. Minimal implementation might use this type to send out a serial number or similar device-specific unique static Identification Data for the device.

## A.6. Certificate Payload

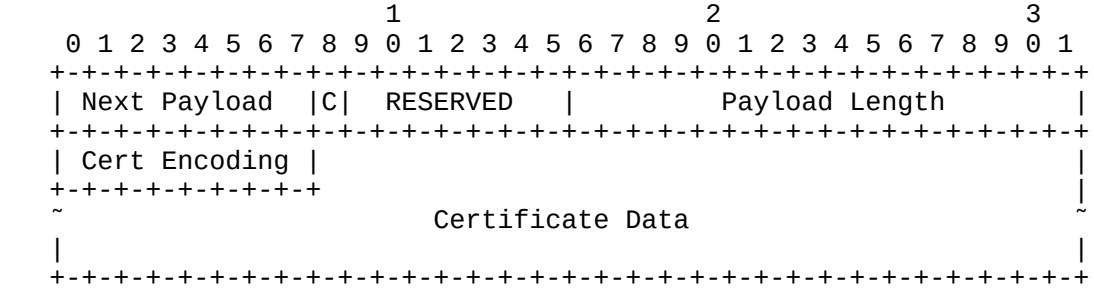


Figure 9: Certificate Payload Format

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

| Certificate Encoding | Value |
|---|---|
| X.509 Certificate - Signature | 4 |
| Raw Public Key | 15 |

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

The syntax of the types above are:

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

- "Raw Public Key" contains a raw public key. In essence, the Certificate Payload contains the SubjectPublicKeyInfo part of the PKIX Certificate (see Section 4.1.2.7 of [RFC5280]). This is a quite simple ASN.1 object that contains mostly static parts before the actual public key values. See [RFC7670] for more information.

## A.7. Certificate Request Payload

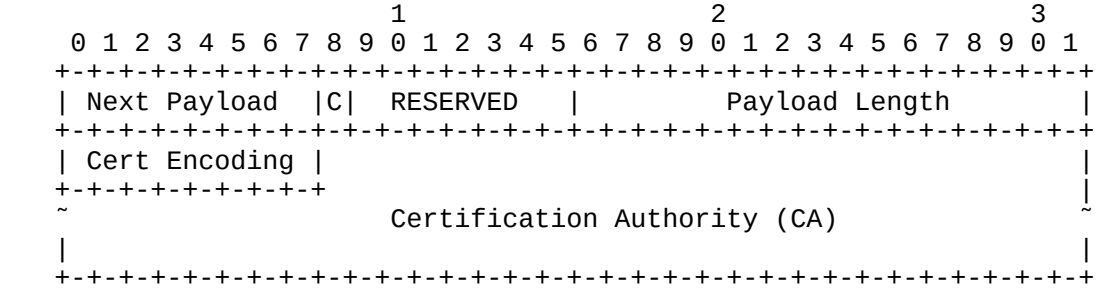


Figure 10: Certificate Request Payload Format

- Certificate Encoding (1 octet) - Contains an encoding of the type or format of certificate requested.

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

The Certificate Encoding field has the same values as those defined by the certificate payload. The Certification Authority field contains an indicator of trusted authorities for this certificate type. The Certification Authority value is a concatenated list of SHA-1 hashes of the public keys of trusted Certification Authorities. Each is encoded as the SHA-1 hash of the Subject Public Key Info element (see Section 4.1.2.7 of [RFC5280]) from each Trust Anchor certificate. The 20-octet hashes are concatenated and included with no other formatting.

## A.8. Authentication Payload

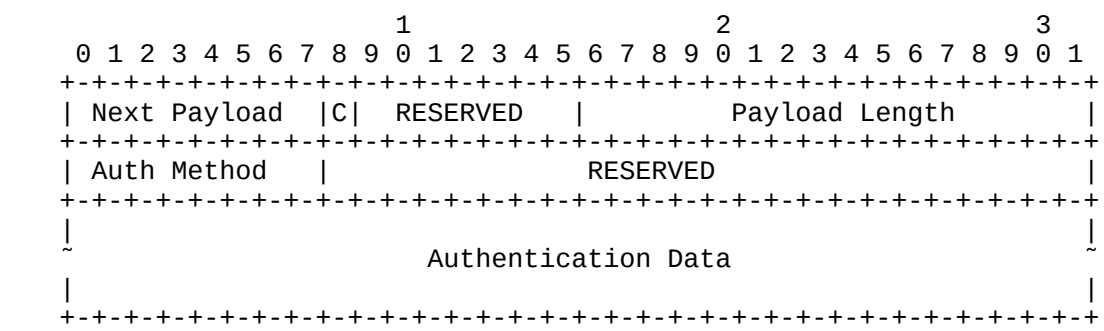


Figure 11: Authentication Payload Format

- Auth Method (1 octet) - Specifies the method of authentication used.

| Mechanism | Value |
|---|---|
| RSA Digital Signature<br>Using an RSA private key with an RSASSA-PKCS1-v1_5 signature scheme specified in [PKCS1]; see Section 2.15 of [RFC7296] for details. | 1 |
| Shared Key Message Integrity Code<br>Computed as specified earlier using the shared key associated with the identity in the ID payload and the negotiated PRF. | 2 |

- Authentication Data (variable length) - see Section 2.1.

## A.9. Nonce Payload

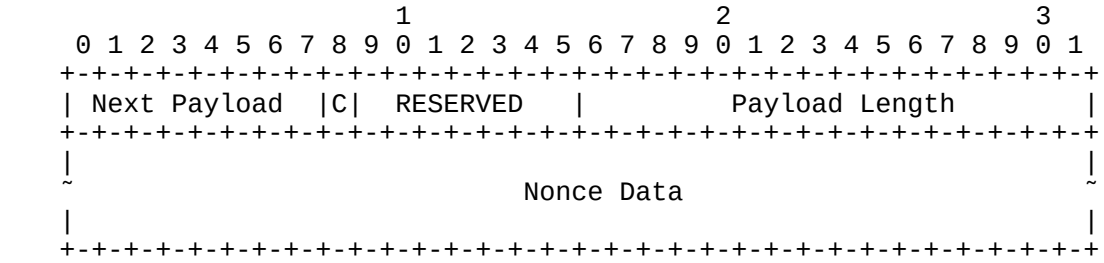


Figure 12: Nonce Payload Format

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

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

## A.10. Notify Payload

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

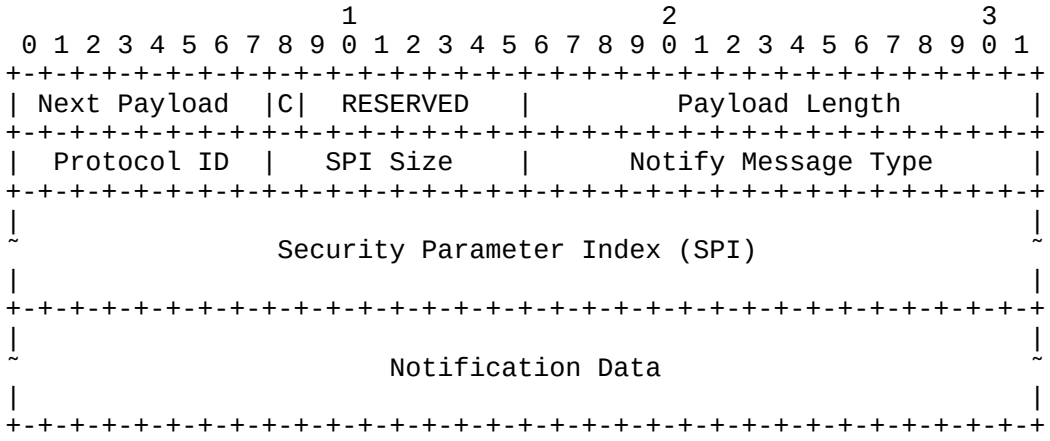


Figure 13: Notify Payload Format

- Protocol ID (1 octet) - If this notification concerns an existing SA whose SPI is given in the SPI field, this field indicates the type of that SA. If the SPI field is empty, this field MUST be sent as zero and MUST be ignored on receipt.

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

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

- SPI (variable length) - Security Parameter Index.

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

### A.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.

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

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

| NOTIFY messages: error types | Value |
|---|---|
| UNSUPPORTED_CRITICAL_PAYLOAD | 1 |
| Indicates that the 1-octet payload type included in the Notification Data field is unknown. | |
| INVALID_SYNTAX | 7 |
| Indicates the IKE message that was received was invalid because some type, length, or value was out of range or because the request was rejected for policy reasons. To avoid a Denial-of-Service (DoS) attack using forged messages, this status may only be returned for and in an encrypted packet if the Message ID and cryptographic checksum were valid. To avoid leaking information to someone probing a node, this status MUST be sent in response to any error not covered by one of the other status types. To aid debugging, more detailed error information should be written to a console or log. | |
| NO_PROPOSAL_CHOSEN | 14 |
| None of the proposed crypto suites was acceptable. This can be sent in any case where the offered proposals are not acceptable for the responder. | |
| NO_ADDITIONAL_SAS | 35 |
| Specifies that the node is unwilling to accept any more Child SAs. | |

| NOTIFY messages: status types | Value |
|---|---|
| INITIAL_CONTACT | 16384 |

Asserts that this IKE SA is the only IKE SA currently active between the authenticated identities.

## A.11. Traffic Selector Payload

Traffic Selector (TS) payloads allow endpoints to communicate some of the information from their Security Policy Database (SPD) to their peers. TS payloads specify the selection criteria for packets that will be forwarded over the newly set up SA.

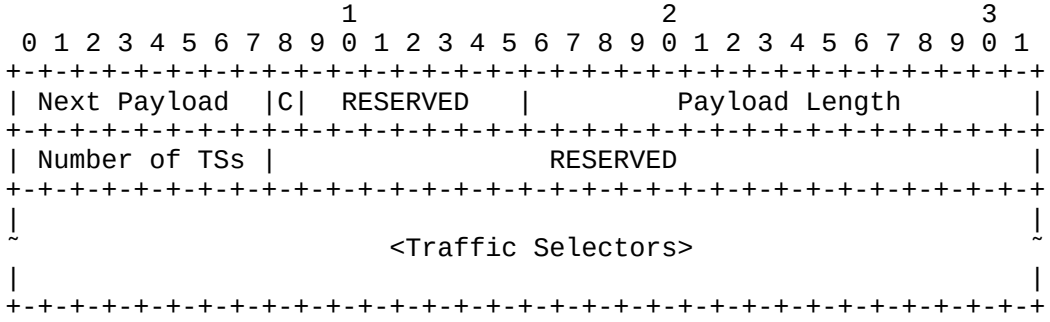


Figure 14: Traffic Selectors Payload Format

- Number of TSs (1 octet) - Number of Traffic Selectors being provided.
- Traffic Selectors (variable length) - One or more individual Traffic Selectors.

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

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

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

The first of the two TS payloads is known as TSi (Traffic Selector - initiator). The second is known as TSr (Traffic Selector - responder). TSi specifies the source address of traffic forwarded from (or the destination address of traffic forwarded to) the initiator of the Child SA pair. TSr specifies the destination address of the traffic forwarded to (or the source address of the traffic forwarded from) the responder of the Child SA pair.

IKEv2 allows the responder to choose a subset of the traffic proposed by the initiator.

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

To enable the responder to choose the appropriate range, if the initiator has requested the SA due to a data packet, the initiator SHOULD include as the first Traffic Selector in each TSi and TSr a very specific Traffic Selector including the addresses in the packet triggering the request. If the initiator creates the Child SA pair not in response to an arriving packet, but rather, say, upon startup, then there may be no specific addresses the initiator prefers for the initial tunnel over any other. In that case, the first values in TSi and TSr can be ranges rather than specific values.

As minimal implementations might only support one SA, the Traffic Selectors will usually be from the initiator's IP address to the responder's IP address (i.e., no port or protocol selectors and only one range).

### A.11.1. Traffic Selector

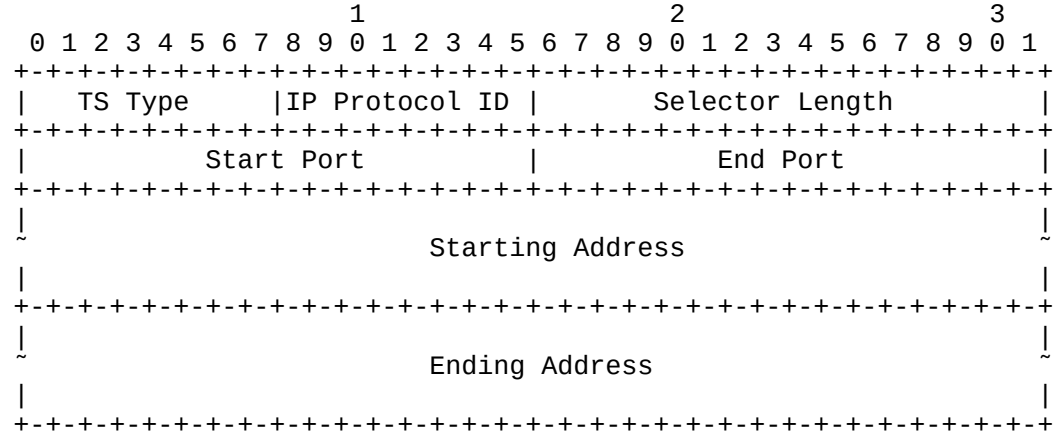


Figure 15: Traffic Selector

- TS Type (1 octet) - Specifies the type of Traffic Selector.
- IP protocol ID (1 octet) - Value specifying an associated IP protocol ID (such as UDP, TCP, and ICMP). A value of zero means that the protocol ID is not relevant to this Traffic Selector -- the SA can carry all protocols.
- Selector Length - Specifies the length of this Traffic Selector substructure including the header.
- Start Port (2 octets, unsigned integer) - Value specifying the smallest port number allowed by this Traffic Selector. For protocols for which port is undefined (including protocol 0), or if all ports are allowed, this field MUST be zero.
- End Port (2 octets, unsigned integer) - Value specifying the largest port number allowed by this Traffic Selector. For protocols for which port is undefined (including protocol 0), or if all ports are allowed, this field MUST be 65535.
- Starting Address - The smallest address included in this Traffic Selector (length determined by TS Type).
- Ending Address - The largest address included in this Traffic Selector (length determined by TS Type).

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

| TS Type | Value |
|---|---|
| TS_IPV4_ADDR_RANGE | 7 |

A range of IPv4 addresses, represented by two 4-octet values. The first value is the beginning IPv4 address (inclusive), and the second value is the ending IPv4 address (inclusive). All addresses falling between the two specified addresses are considered to be within the list.

| | |
|---|---|
| TS_IPV6_ADDR_RANGE | 8 |

A range of IPv6 addresses, represented by two 16-octet values. The first value is the beginning IPv6 address (inclusive), and the second value is the ending IPv6 address (inclusive). All addresses falling between the two specified addresses are considered to be within the list.

## A.12. Encrypted Payload

The Encrypted payload, denoted as SK{...} in this document, contains other payloads in encrypted form.

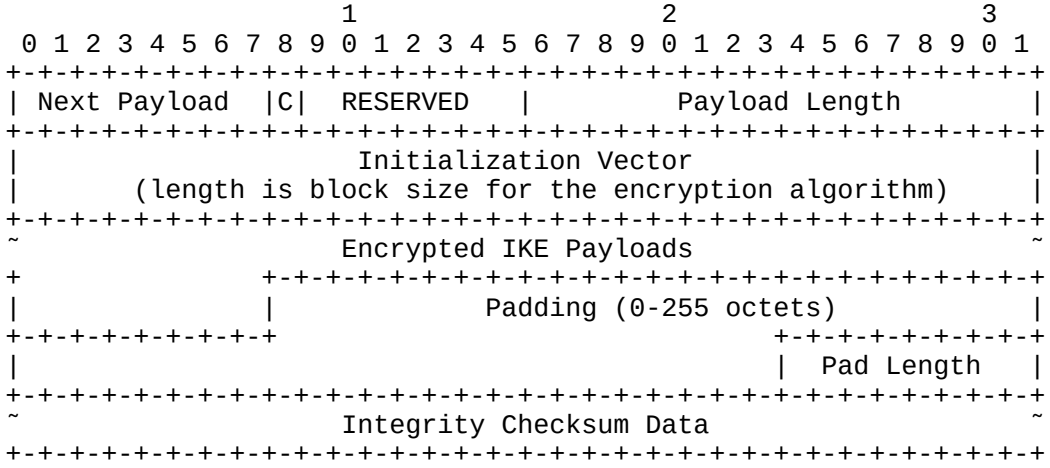


Figure 16: Encrypted Payload Format

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

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

- Initialization Vector - For Cipher Block Chaining (CBC) mode ciphers, the length of the initialization vector (IV) is equal to the block length of the underlying encryption algorithm. Senders MUST select a new unpredictable IV for every message; recipients MUST accept any value. The reader is encouraged to consult [MODES] for advice on IV generation. In particular, using the final ciphertext block of the previous message is not considered unpredictable. For modes other than CBC, the IV format and processing is specified in the document specifying the encryption algorithm and mode.

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

- Padding MAY contain any value chosen by the sender and MUST have a length that makes the combination of the payloads, the Padding, and the Pad Length to be a multiple of the encryption block size. This field is encrypted with the negotiated cipher.

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

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

## Appendix B. Useful Optional Features

There are some optional features of IKEv2, which might be useful for minimal implementations in some scenarios. Such features include raw public keys authentication and sending an IKE SA delete notification.

### B.1. IKE SA Delete Notification

In some scenarios, a minimal implementation device creates an IKE SA, sends one or few packets, perhaps gets some packets back, and then the device goes back to sleep, forgetting the IKE SA. In such scenarios, it would be nice for the minimal implementation to send the IKE SA delete notification to tell the other end that the IKE SA is going away, so it can free the resources.

Deleting the IKE SA can be done by sending one packet with a fixed Message ID and with only one payload inside the Encrypted payload. The other end will send back an empty response:

```
Initiator                          Responder
-------------------------------------------------------------------
HDR(SPIi=xxx, SPIr=yyy, INFORMATIONAL,
    Flags: Initiator, Message ID=2),
    SK {D}  -->

                   <--  HDR(SPIi=xxx, SPIr=yyy, INFORMATIONAL,
                            Flags: Response, Message ID=2),
                            SK {}
```

The Delete payload format is:

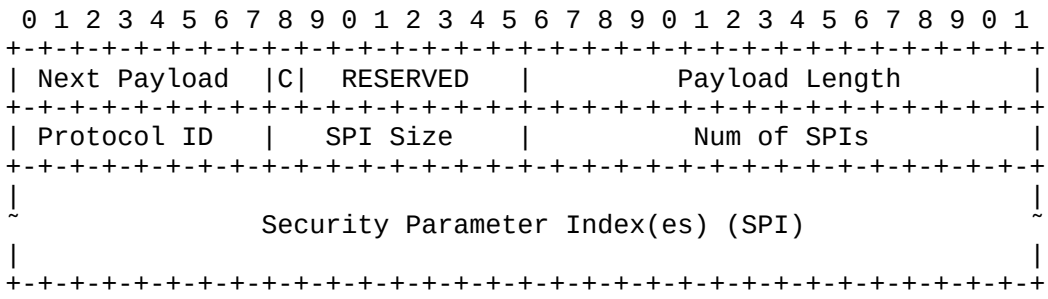


Figure 17: Delete Payload Format

- Protocol ID (1 octet) - Must be 1 for an IKE SA.

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

- Num of SPIs (2 octets, unsigned integer) - The number of SPIs contained in the Delete payload. This MUST be zero for IKE.

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

## B.2. Raw Public Keys

In some scenarios, the shared secret authentication is not safe enough, as anybody who knows the secret can impersonate the server. If the shared secret is printed on the side of the device, then anybody who gets physical access to the device can read it. In such environments, public key authentication allows stronger authentication with minimal operational overhead. Certificate support is quite complex, and minimal implementations do not usually have need for them. Using Raw Public Keys is much simpler, and it scales similar to certificates. The fingerprint of the raw public key can still be distributed by, for example, printing it on the side of the device allowing setup similar to using a shared secret.

Raw public keys can also be used in a "leap of faith" or baby duck style initial setup, where the device imprints itself to the first device it sees when it boots up the first time. After that initial connection, it stores the fingerprint of the Raw Public Key of the server in its own configuration and verifies that it never changes (unless a "reset to factory settings" or similar command is issued).

This changes the initial IKE_AUTH payloads as follows:

```
Initiator                          Responder
-------------------------------------------------------------------
HDR(SPIi=xxx, SPIr=yyy, IKE_AUTH,
    Flags: Initiator, Message ID=1),
    SK {IDi, CERT, AUTH, SAi2, TSi, TSr,
        N(INITIAL_CONTACT)}  -->

                  <--  HDR(SPIi=xxx, SPIr=yyy, IKE_AUTH, Flags:
                             Response, Message ID=1),
                             SK {IDr, CERT, AUTH, SAr2, TSi, TSr}
```

The CERT payloads contain the raw public keys used to sign the hash of the InitiatorSignedOctects/ResponderSignedOctects when generating an AUTH payload. Minimal implementations should use SHA-1 as the hash function as that is the "SHOULD" support algorithm specified in RFC 7296, so it is the most likely one that is supported by all devices.

Note that RFC 7296 already obsoleted the old Raw RSA Key method, and "Generic Raw Public-Key Support for IKEv2" [RFC7670] adds a new format to allow using any types of raw public keys with IKEv2. This document only specifies how to use the new format.

In these setups, it might be possible that authenticating the server is not needed at all. If a minimal device is sending, for example, sensor information to the server, the server wants to verify that the sensor is who it claims to be using raw public keys, but the sensor does not really care who the server is. In such cases, the NULL authentication method [RFC7619] would be useful, as it allows devices to do one-way authentication.

## Acknowledgements

Most of the content of this document is copied from RFC 7296.

## Author’s Address

Tero Kivinen
INSIDE Secure
Eerikinkatu 28
HELSINKI FI-00180
FINLAND

Email: kivinen@iki.fi