Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 5, 2019

V. Smyslov
ELVIS-PLUS

April 3, 2019

Compact Format of IKEv2 Payloads
draft-smyslov-ipsecme-ikev2-compact-05

Abstract

This document describes a method for reducing the size of the Internet Key Exchange version 2 (IKEv2) messages by using an alternative format for IKE payloads. Standard format of many IKE payloads contains a lot of redundancy. This document takes advantage of this fact and specifies a way to eliminate some redundancy by using denser encoding. Reducing size of IKEv2 messages is desirable for low power consumption battery powered devices. It also helps to avoid IP fragmentation of IKEv2 messages.

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 5, 2019.

Table of Contents

1. Introduction

The Internet Key Exchange protocol version 2 (IKEv2) specified in [RFC7296] is used in the IP Security (IPsec) architecture for the purposes of Security Association (SA) parameters negotiation and authenticated key exchange. The protocol uses UDP as the transport for its messages, which size varies from less than one hundred bytes to several kBytes.

Decreasing the size of IKEv2 messages is highly desirable for the Internet of Things (IoT) devices utilizing lower power consumption technology. For some of such devices the power consumption for transmitting extra bits over network is prohibitively high (see appendix A of [IPSEC-IOT-REQS]). Many such devices are battery powered without an ability to recharge or to replace the battery which serves for the life cycle of the device (often several years). For this reason the task of reducing the power consumption for such devices is very important and decreasing messages size is one of the ways to accomplish it.

Large UDP messages may also cause fragmentation at the IP level, which may interact badly with Network Address Translators (NAT). In particular, some NATs drop IP fragments that don't contain TCP/UDP port numbers (non-initial IP fragments), so that the IP datagram

cannot be reassembled on the receiving end and IKE SA cannot be established. One of the possible solutions to the problem is IKEv2 fragmentation [RFC7383]. However, the IKEv2 fragmentation can only be applied to encrypted messages and thus cannot be used in the initial IKEv2 exchange, IKE_SA_INIT. Usually the IKE_SA_INIT messages are relatively small and don't cause IP fragmentation. However, while more and more new algorithms and protocol extensions are included in IKEv2, these messages become larger and larger thus increasing the risk of IP fragmentation. It is desirable to make IKEv2 messages more compact to help avoid this risk.

IKEv2 messages are comprised of data structures called payloads. Each payload consists of a common part (Generic Payload Header) followed by a payload-specific data which is formatted differently depending on the payload's purpose. Section 3 of [RFC7296] lists formats for all standard IKEv2 payloads. As one can see some IKEv2 payloads are formatted in such a way, that there are substantial redundancy in their encoding. This document defines an alternative format for the IKEv2 payloads, which provides denser encoding, that allows making IKEv2 messages more compact.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Overview

The idea behind the protocol is to develop an alternative compact encoding of IKEv2 payloads that meets the following requirements:

1. The compact encoding must be applicable to all already defined IKEv2 payloads as well as to future payloads, so it must not depend on payload format. There may be few special cases if specific encoding is justified by much higher efficiency.

2. The compact encoding must be easily converted to standard encoding and vice versa. This allows implementations to re-use existing composing/parsing code and to apply compact encoding/ decoding as pre/post process steps in message processing.

3. The compact encoding/decoding algorithm must consume low resources in terms of code size, CPU consumption and memory footprint.

4. The compact encoding must never increase the payload size and must be effective enough to noticeably decrease the size of most payloads.

To meet these requirements the encoding algorithm must be independent from the particular payload format. In other words, it must take any given payload and perform some kind of compression. General purpose lossless compression algorithms are usually not very effective when they are applied to small amount of data. Fortunately format of many IKEv2 payloads has a peculiarity that allows using simple and relatively efficient compression algorithm.

Many IKEv2 payloads contain a lot of zero octets. These octets come from the following sources:

- Many Payload Headers contain RESERVED fields that must be zeroed according to IKEv2 specification.

- Payload length is encoded in two octets, while most IKEv2 payloads are less than 256 octets in size.

- Substantial number of IKEv2 parameters is encoded in two octets, while the number of currently defined values for these parameters is less than one hundred (see [IKEV2-IANA]).

The idea is to omit these zero octets from the compact payload and supply a bitmap that will indicate which octets were omitted.

IKEv2 header also contains some redundancy - in particular the Length field occupies four octets, while IKEv2 messages are extremely unlikely to exceed few Kb in size. However, some middleboxes perform an inspection of IKEv2 header and changing its format will likely interact badly with these middleboxes. Thus, the possible saving of few octets doesn't justify modification of IKEv2 header.

4. Compact Representation of IKEv2 Payloads

This document defines one generic compact format suitable for compacting any IKEv2 payload and two specific compact formats - one for SA payload and the other for Notify payload containing status notification with no data. The rationale for such a design is the following.

One common generic compact format simplifies implementations and allows using it with any currently defined and not yet defined payloads. However, it doesn't provide high level of efficiency.

SA payload contains a lot of redundancy and can be encoded in highly efficient way. Moreover, this payload grows up quickly once more new transforms are defined and implementations start using them. In some cases the SA payload can be the largest payload in the IKE_SA_INIT exchange. So, there is a natural desire to encode it differently to gain better result. On the other hand, Notify payload with status notification containing no data is often used in IKEv2 initial exchange to negotiate support for various protocol extensions. So, there are usually several such payloads in the IKE_SA_INIT request and response messages, ant it is anticipated that their number will increase as long as more and more IKEv2 extensions are defined. That's why this payload is also encoded differently to make initial messages more compact.

4.1. Compact Generic Payload

Generic IKEv2 payload is depicted below (Figure 1) for convenience. It consists of generic payload header followed by payload data. Brief description of generic payload header fields is provided below, readers should refer to Section 3.2 of [RFC7296] for full description.

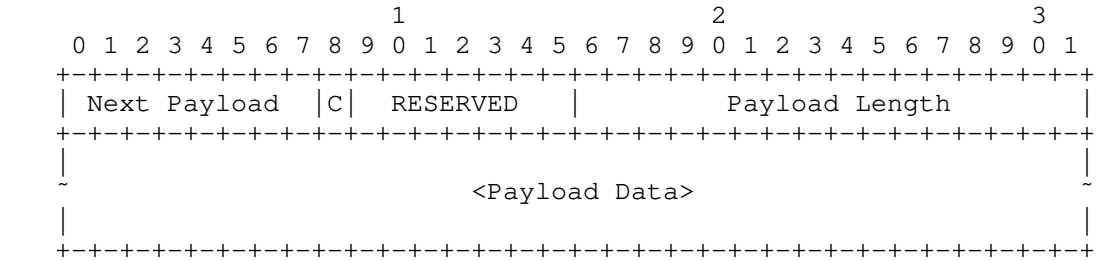


Figure 1: Generic Payload

- Next Payload (1 octet) - Identifier for the payload type of the next payload in the message.
- Critical (1 bit) - This bit is set if the current payload cannot be skipped in case the receiver doesn't understand its type and cleared if it is safe to skip this payload.
- RESERVED (7 bits) - MUST be sent as zero; MUST be ignored on receipt.
- Payload Length (2 octets, unsigned integer) - Length in octets of the current payload, including the generic payload header.
- Payload Data (variable) - Contains payload specific data.

Some payloads have extended headers following the generic payload header. For the purpose of compact payload encoding any extended header is treated as part of payload data. Complete description of IKEv2 Payload formats can be found in Section 3 of [RFC7296].

Figure 2 shows generic compact payload format.

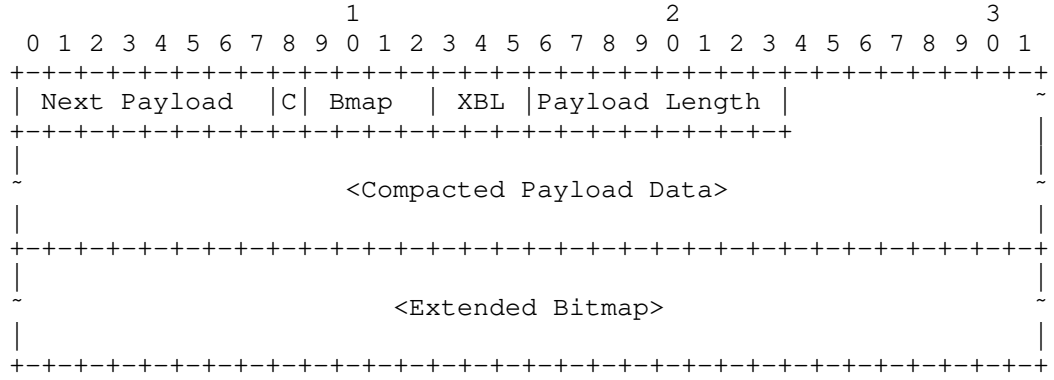


Figure 2: Compact Generic Payload

- Next Payload (1 octet) - Identifier for the payload type of the next payload in the message. The value is taken intact from original payload.

- Critical (1 bit) - This bit is set if the current payload cannot be skipped in case the receiver doesn't understand its type and cleared if it is safe to skip this payload. The value is taken intact from original payload.

- Bmap (4 bits) - Bitmap, contains bitmap where each bit indicates whether one of the four first octets of original Payload Data was zero and thus was omitted from Compacted Payload Data. If the bit is set, then the corresponding octet was zero and was omitted, if the bit is cleared, then either the corresponding octet was copied or it didn't present in the original Payload Data (in case the octet would be outside the original payload). The rightmost bit of the map corresponds to the first of the four octets and the leftmost bit corresponds to the last of these four octets.

- XBL (3 bits) - Extended Bitmap Length, specifies the number of Extended Bitmap octets plus one. Note, that by construction this field cannot be zero.

- Payload Length (1 octet) - Length in octets of compacted payload not including Extended Bitmap.

- Compacted Payload Data (variable) - Contains original payload data with some zero octets omitted. Up to 52 zero octets from the beginning of original payload data can be omitted, the rest of original payload data is always copied.

- Extended Bitmap (variable, 0 - 6 octets) - contains extended bitmap. Each octet of this bitmap represents how the corresponding eight octets of the original payload data were handled - original octets corresponding to the set bits were zero and thus were omitted from the compacted payload data, while octets corresponding to the cleared bits were copied. First octet in the Extended Bitmap corresponds to octets from 5 to 12 of the original payload data, next octet - to octets from 13 to 20, etc. Note, that the Bmap field indicates how octets from 1 to 4 were handled. In each octet of extended bitmap the rightmost bit represents the first of corresponding original octets and the leftmost bit represents the last one.

An IKEv2 payload can be compacted if it meets two requirements:

- The payload length is less than 256 octets. In other words, the high order octet of Payload Length field is zero.

- The RESERVED field in the generic payload header is zero. Section 3.2 of [RFC7296] requires that this field must always be zero when preparing payload.

If payload doesn't meet these requirement it is included into the message without modification. The regular and compact payloads can be present in any order in the compact IKEv2 message. The receiving side can distinguish between them by analyzing the least significant three bits of RESERVED field in generic payload header. These bits correspond to XBL field in compact generic payload header, and this field will always be non-zero in compact payload while these bits are required to be zero in regular payload.

If payload meets the above requirements then it is compacted as follows.

1. The Next Payload and Critical fields are copied from original payload.

2. The first 4 octets of the original payload data are scanned one by one. If the current octet is zero, then it is omitted from the Compacted Payload data and the corresponding bit (starting from the rightmost bit to the leftmost bit) in Bmap field is set. Otherwise the octet is copied and the corresponding bit is

cleared. If there are no more octets in the original payload data then the process stops and the rest of Bmap is zeroed.

3. If any more octets left in the original payload data, then the next 48 of them are scanned one by one. Since the Extended Bitmap position isn't known yet a temporary bitmap of six octets is used. If the current octet is zero, then it is omitted from the Compacted Payload Data and the corresponding bit (starting from the rightmost bit to the leftmost bit) in the corresponding octet of a temporary bitmap is set. Otherwise the original octet is copied and the corresponding bit in temporary bitmap is cleared. If there are no more octets in the original payload data then the process stops and the rest bits of the current temporary bitmap octet are zeroed.

4. If any more octets left in the original payload data, they are copied to the Compacted Payload Data.

5. The Payload Length field is set to indicate the size of Compacted Payload Data plus 3. It will indicate the offset of the Extended Bitmap from the beginning of compact payload.

6. The content of the temporary bitmap is copied to the Extended Bitmap field (after the Compacted Payload Data). The temporary bitmap octets are copied one by one up to the first zero octet (if any). In other words, the Extended Bitmap MUST NOT contain zero octets.

7. The XBL field is set to the number of octets placed in the Extended Bitmap plus one.

4.2. Compact SA Payload

SA payload is compacted differently than generic IKEv2 payload. The specific format for Compact SA payload allows achieving very high level of compression. High level of compression is important, because SA payload grows up quickly as more and more cryptographic transforms are defined, get widespread adoption and advertised by Initiators.

Unlike generic compact payload, which retains its payload type and is distinguished from regular IKEv2 payload by analyzing the leftmost three bits of RESERVED field of the generic payload header, the Compact SA payload has its own payload type. The reason is that its format (Figure 3) is completely different and the above method cannot be used. The Compact SA payload is denoted CSA, and its payload type is <TBA by IANA>.

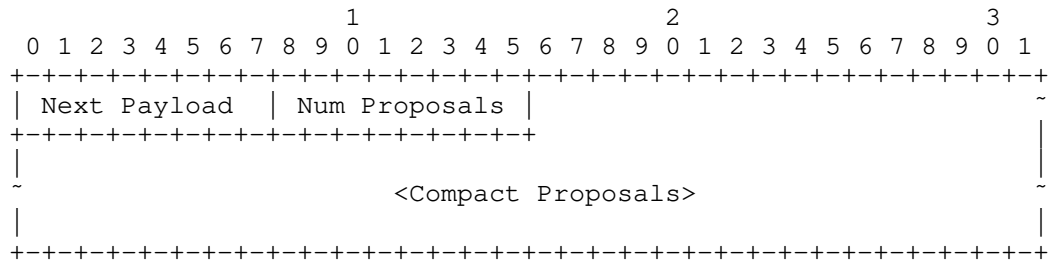


Figure 3: Compact SA Payload

- Next Payload (1 octet) - Identifier for the payload type of the next payload in the message. The value is taken intact from original payload.

- Num Proposals (1 octet) - Specifies the number of Compact Proposals in this SA payload.

- Compact Proposals (variable) - One or more compact proposal substructures.

Despite the fact that Compact SA payload has different payload type and different format than regular SA payload, the associated semantics MUST be the same. Regular SA payload can always be compacted if it is compliant with Section 3.3 of [RFC7296].

4.2.1. Compact Proposal Substructure

Compact Proposal substructure (Figure 4) resembles regular Proposal substructure lacking first four octets. The Compact Proposal substructure fields are briefly described below. Readers should refer to Section 3.3.1 of [RFC7296] for detailed description of Compact Proposal substructure fields with the same names, as in regular Proposal substructure.

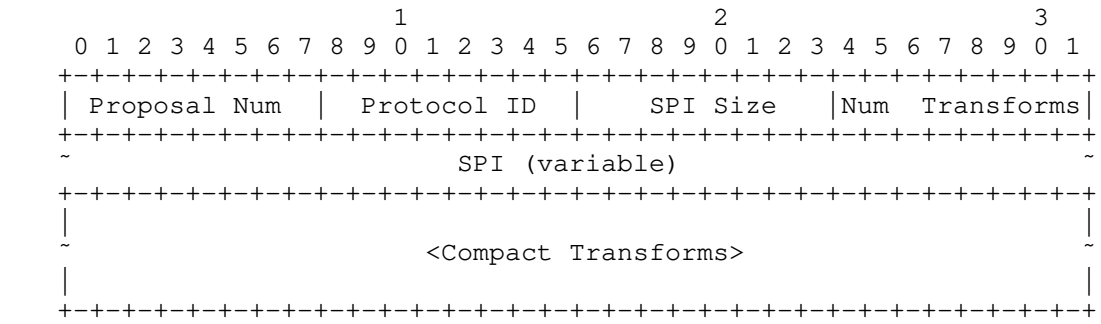


Figure 4: Compact Proposal Substructure

- Proposal Num (1 octet) - Proposal Number.
- Protocol ID (1 octet) - Specifies the IPsec protocol identifier for the current negotiation.
- SPI Size (1 octet) - Size of SPI.
- Num Transforms (1 octet) - Specifies the number of transforms in this proposal.
- SPI (variable) - The sending entity's SPI.
- Compact Transforms (variable) - One or more compact transform substructures.

4.2.2. Compact Transform Substructures

Compact Transform substructures are encoded differently depending on Transform Type, Transform ID and presence of attributes to get most effective encoding for common use cases. The leftmost bits of the first octet of the Compact Transform substructure are used to distinguish between different formats. These bits are called Tag. The table below shows how Tag value correlates with Compact Transform substructure format.

Tag	Compact Transform Format	Len	Trans Types	Trans IDs	Format
0tttvvvv	Short: Generic	1	6-13	0-15	Figure 5
100vvvvv	Short: Encryption (no Key Length attribute or 128-bit key)	1	1	11-42	Figure 6
101vvvvv	Short: Encryption (256-bit key)	1	1	11-42	Figure 6
110vvvvv	Short: Diffie-Hellman Group	1	4	0, 14-44	Figure 7
1110vvvv	Short: PRF	1	2	2-15	Figure 8
1110000v	Short: ESN	1	5	0-1	Figure 9
1111tttt	Long 1	2	1-15	0-127	Figure 10
1111tttt	Long 2	3	1-15	0-32767	Figure 11
11110000	Full	6+	0-255	0-65535	Figure 12

Table 1: Tag values and corresponding Compact Transform formats

Short formats are the most efficient Compact Transform formats, they occupy only one octet; long format occupies either two or three octets depending on the Transform ID value. Both short and long formats can be used only for some Transform Types and can represent only limited number of Transform IDs. Moreover, both short and long formats cannot be used if Transform contains any attributes, except if it is the Encryption Transform and it contains the Key Length attribute and the value of the attribute is either 128 or 256. The Table 1 summarizes the restrictions each format implies.

Full format imposes no constraints on the Transform Type and Transform ID, as well on the attributes the transform could contain.

4.2.2.1. Short Format

Short format allows encoding both Transform Type and Transform ID using single octet. It has several variations - a generic short format and a number of specific formats for different Transform Types that take advantage of the concrete Transform IDs defined in [IKEV2-IANA] for these Transform Types.

Figures 5-8 show short format encodings for different Transform Types.

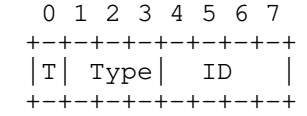


Figure 5: Generic Short Format

- T(ag) (1 bit) - MUST be 0.
- Type (3 bits) - Transform Type minus 6. This allows Transform Types from 6 to 13 to be encoded using this format.
- ID (4 bits) - Transform ID.

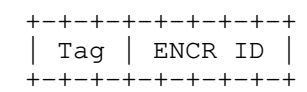


Figure 6: Short Format for Encryption

- Tag (3 bits) - MUST be either 100 or 101. Tag value 100 indicates that either no Key Length attribute is present in the original Transform or it is present and its value is 128. Tag values 101 indicates that Key Length attribute containing value 256 is present in the original Transform.
- ENCR ID (5 bits) - Encryption Algorithm Transform ID minus 11.

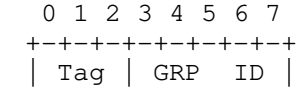


Figure 7: Short Format for Diffie-Hellman Group

- Tag (3 bits) - MUST be 110.

- GRP ID (5 bits) - Value 0 indicates NONE, other values are treated as Diffie-Hellman Group Transform ID minus 14.

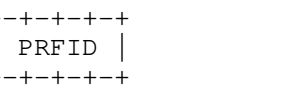


Figure 8: Short Format for PRF

- Tag (4 bits) - MUST be 1110.

- PRFID (4 bits) - Pseudo-random Function Transform ID. This value MUST never be 0 and 1.

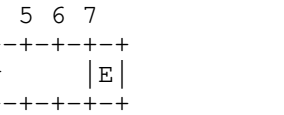


Figure 9: Short Format for ESN

- Tag (7 bits) - MUST be 1110000.

- E (bit) - Extended Sequence Numbers Transform ID (either 0 or 1).

4.2.2.2. Long Format

Long format (Figures 10 and 11) is used when Transform doesn't meet requirements for short format encoding, but still meets the following requirements:

1. Transform Type is between 1 and 15. At the time this document was written only Transform Types 1 to 5 were defined (see [IKEV2-IANA]).

2. Transform has no attributes.

3. Transform ID is less than or equal to 32767.

Long format allows to effectively encode Transform IDs for Transform Types that don't fit into the short format, e.g. private Transform IDs (if these transforms don't have associated attributes and its value is less than or equal to 32767).

Long format can occupy two or three octets depending on the Transform ID value. For values 0-127 only one octet is used to represent the

value, for values 128-32767 two octets are used. Values greater than 32767 require using full format.

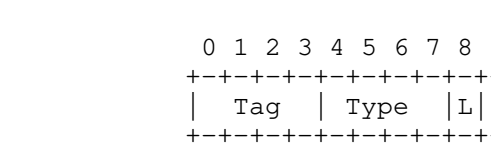


Figure 10: Long Format 1

- Tag (4 bits) - MUST be 1111.
- Type (4 bits) - The type of transform being specified in this substructure. This field is always non-zero, since Transform Type 0 is marked as Reserved in [IKEV2-IANA].
- L (1 bit) - MUST be 0.
- Trans ID (7 bits) - The specific instance of the Transform Type being specified.

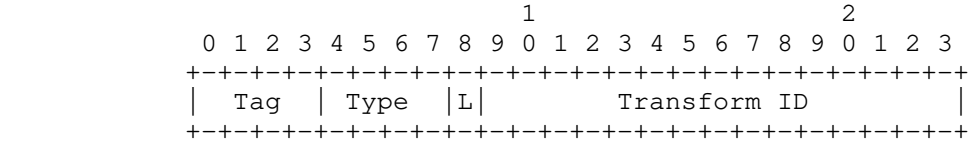


Figure 11: Long Format 2

- Tag (4 bits) - MUST be 1111.
- Type (4 bits) - The type of transform being specified in this substructure. This field is always non-zero, since Transform Type 0 is marked as Reserved in [IKEV2-IANA].
- L (1 bit) - MUST be 1.
- Transform ID (15 bits) - The specific instance of the Transform Type being specified.

4.2.2.3. Full Format

Full format of Compact Transform substructure allows encoding of any transform without restrictions. It is used when transform cannot be encoded neither in short format nor in long format. The format (Figure 12) resembles regular Transform Substructure with all RESERVED fields removed. The Compact Transform substructure fields

are briefly described below. Readers should refer to Section 3.3.2 of [RFC7296] for detailed description of Compact Transform substructure fields with the same names, as in regular Transform substructure.

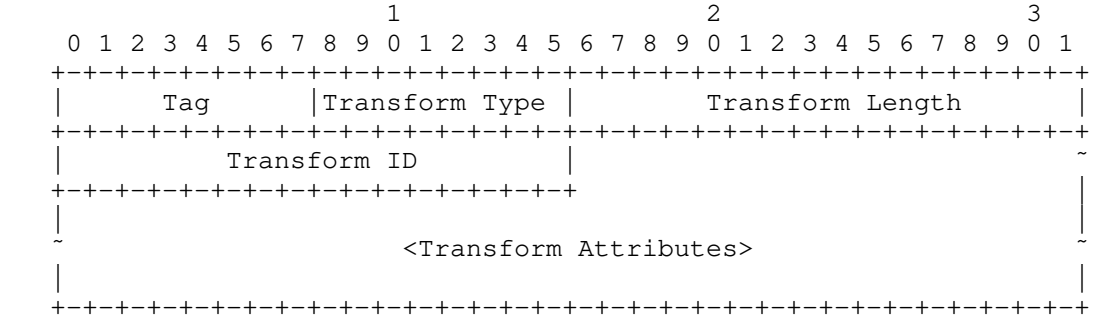


Figure 12: Full Format

- Tag (1 octet) - MUST be 11110000.
- Transform Type (1 octet) - The type of transform being specified in this substructure.
- Transform Length (2 octets, unsigned integer) - The length in octets of the Compact Transform Substructure including Header and Attributes.
- Transform ID (2 octets) - The specific instance of the Transform Type being specified.
- Transforms Attributes (variable) - Transform attributes.

4.3. Compact Notify Payload

Notify payloads containing status notifications with no data are often used in IKEv2. This is "de facto" standard way to negotiate various protocol extensions and for that reason usually several such Notify payloads are present in initial IKEv2 exchanges. It is anticipated that the number of IKEv2 extensions will increase and thus the size of initial exchange messages will increase too. This is the reason that this kind of Notify payload is encoded specifically, to get more effective message compression.

As well as Compact SA payload, Compact Notify payload has its own payload type. The Compact Notify payload is denoted CN, and its payload type is <TBA by IANA>.

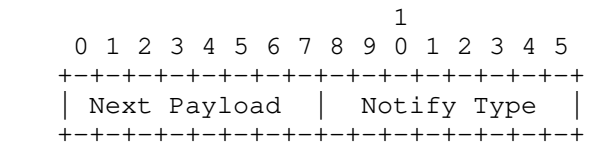


Figure 13: Compact Notify Payload for Status Notification

- Next Payload (1 octet) - Identifier for the payload type of the next payload in the message. The value is taken intact from original payload.

- Notify Type (1 octet) - The type of notification message minus 16384.

Despite the fact that Compact Notify payload has different payload type and different format than regular Notify payload, the associated semantics MUST be the same.

Notify payload can be encoded as Compact Notify payload if it meets the following requirements (see Section 3.10 of [RFC7296] for Notify payload format):

1. Notify Message Type is between 16384 and 16639 (inclusive). This corresponds to status notifications.

2. Protocol ID is zero.

3. SPI Size is zero, meaning no SPI is present.

4. Notification Data is empty.

At the time this document was written about 40 percent of status notifications defined in [IKEV2-IANA] met these requirements. If Notify payload doesn't meet these requirements, the generic compact format (Section 4.1) can be tried.

5. Compact Format Negotiation

Most IKEv2 extensions are negotiated in the following way. The Initiator announces its support for some extension by including corresponding Vendor ID payload or Notify payload containing status notification in the request message of IKE_SA_INIT or IKE_AUTH exchanges. If the Responder supports this extension it returns the same (or some specific) payload to the Initiator in response message. Responder that doesn't support compact format just ignores these payloads in accordance with IKEv2 specification.

This method is inappropriate for negotiation of compact format, because Initiator should be able to send an initial request message in compact form and thus it must inform Responder somehow that the message must be parsed differently than regular IKEv2 message. Since compact message may contain regular IKEv2 payloads, it is possible to define a new status notification and to include it without compacting as the very first payload in the initial request. However, this spoils the whole idea of reducing initial messages size, since this payload will increase message size for no good reason.

Instead this document specifies a different negotiation mechanism. An alternative initial exchange is defined, ALT_IKE_SA_INIT. Initiator wishing to use compact representations of IKEv2 payloads MUST start creating IKEv2 SA using ALT_IKE_SA_INIT exchange instead of IKE_SA_INIT. The very first ALT_IKE_SA_INIT request may contain compact payloads. If Responder receives ALT_IKE_SA_INIT request and doesn't support compact format, then according to Section 2.21.1 of [RFC7296] it discards the request. It may also send INVALID_SYNTAX notification. For the Initiator receiving no response after several retransmissions or receiving INVALID_SYNTAX notification is an indication that the Responder doesn't support compact format. In this case the Initiator MAY restart initial request using regular IKE_SA_INIT request.

If the Responder supports compact format then it response with ALT_IKE_SA_INIT response, confirming to the Initiator that using compact format is successfully negotiated. Once using compact format is negotiated, compact payloads may appear in any message of subsequent exchanges in the context of the IKE SA being negotiated. If IKE SA is rekeyed, then the flag whether compact format is allowed MUST be inherited by the new SA from the old one.

The semantics associated with ALT_IKE_SA_INIT exchange MUST be the same, as the semantics associated with IKE_SA_INIT exchange. In other words, when ALT_IKE_SA_INIT exchange is used, the endpoints must behave exactly as if it is IKE_SA_INIT exchange. The only difference (apart from different Exchange Types) is that IKE_SA_INIT messages MUST NOT contain compact payloads, while ALT_IKE_SA_INIT MAY contain them.

6. Interaction with other IKEv2 Extensions

When using compact encoding with IKEv2 Session Resumption [RFC5723], the flag indicating whether peers negotiated compact format MUST be stored in the resumption ticket and used in an SA created as a result of resumption.

7. Security Considerations

Compact format of IKEv2 payloads doesn't change security properties of the protocol, which are described in Section 5 of [RFC7296].

8. IANA Considerations

This document defines two new Payloads in the "IKEv2 Payload Types" registry:

<TBA>	Compact SA Payload	CSA
<TBA>	Compact Notify Payload	CN

This document also defines a new Exchange Type in the "IKEv2 Exchange Types" registry:

<TBA>	ALT_IKE_SA_INIT

9. References

9.1. 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, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[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, <https://www.rfc-editor.org/info/rfc7296>.

[RFC7383] Smyslov, V., "Internet Key Exchange Protocol Version 2 (IKEv2) Message Fragmentation", RFC 7383, DOI 10.17487/RFC7383, November 2014, <https://www.rfc-editor.org/info/rfc7383>.

9.2. Informative References

[RFC5723] Sheffer, Y. and H. Tschofenig, "Internet Key Exchange Protocol Version 2 (IKEv2) Session Resumption", RFC 5723, DOI 10.17487/RFC5723, January 2010, <https://www.rfc-editor.org/info/rfc5723>.

[IPSEC-IOT-REQS]
Migault, D., Guggemos, T., and C. Bormann, "Requirements for Diet-ESP the IPsec/ESP protocol for IoT", draft-mglt-6lo-diet-esp-requirements-02 (work in progress), July 2016.

[IKEV2-IANA]
"Internet Key Exchange Version 2 (IKEv2) Parameters", <http://www.iana.org/assignments/ikev2-parameters>.

Author's Address

Valery Smyslov
ELVIS-PLUS
PO Box 81
Moscow (Zelenograd) 124460
RU

Phone: +7 495 276 0211
Email: svan@elvis.ru

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 12, 2018

B. Weis
Cisco Systems
U. Mangla
Juniper Networks Inc.
T. Karl
Deutsche Telekom
N. Maheshwari

September 8, 2017

GDOI GROUPKEY-PUSH Acknowledgement Message
draft-weis-gdoi-rekey-ack-07

Abstract

The Group Domain of Interpretation (GDOI) includes the ability for a Group Controller/Key Server (GCKS) to provide a set of current Group Member (GM) devices with additional security associations (e.g., to rekey expiring security associations). This memo adds the ability of a GCKS to request the GM devices to return an acknowledgement of receipt of its rekey message, and specifies the acknowledgement method.

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 March 12, 2018.

Copyright Notice

Table of Contents

1. Introduction

The Group Domain of Interpretation (GDOI) [RFC6407] is a group key management method by which a Group Controller/Key Server (GCKS) distributes security associations (i.e., cryptographic policy and keying material) to a set of Group Member (GM) devices. GDOI meets the requirement of the Multicast Security (MSEC) Group Key Management Architecture [RFC4046], and defines both a Registration Protocol and Rekey Protocol. GDOI describes the Rekey Protocol as a GROUPKEY-PUSH message.

A GDOI GCKS uses a GROUPKEY-PUSH message (Section 4 of [RFC6407]) to alert group members to updates in policy for the group, including new policy and keying material, replacement policy and keying material, and indications of deleted policy and keying material. Usually the GCKS does not require a notification that the group member actually received the policy. However, in some cases it is beneficial for a GCKS to be told by each receiving GM that it received the rekey message and by implication has reacted to the policy contained within. For example, a GCKS policy can use the acknowledgements to determine which GMs are receiving the current group policy and which GMs are no longer participating in the group.

This memo introduces a method by which a GM returns an acknowledgment message to the GCKS. Initially a GCKS requests GM to acknowledge GROUPKEY-PUSH messages as part of distributed group policy. Then (shown in Figure 1) when the GCKS delivers a GROUPKEY-PUSH message, each GM that honors the GCKS request returns a GROUPKEY-PUSH Acknowledgement Message. The rest of this memo describes this method in detail.

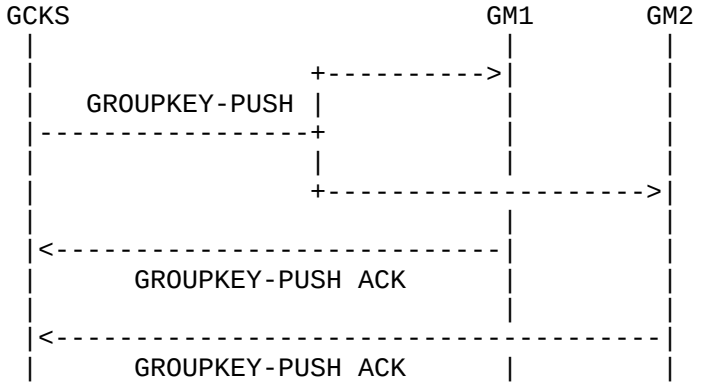


Figure 1: GROUPKEY-PUSH Rekey Event

Implementation of the GROUPKEY-PUSH Acknowledgement Message is OPTIONAL.

1.1. Requirements notation

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].

1.2. Acronyms and Abbreviations

The following acronyms and abbreviations are used throughout this document.

D Delete Payload

GCKS Group Controller/Key Server

GDOI Group Domain of Interpretation

GM Group Member

HDR Header Payload

IV Initialization Vector

KD Key Download Payload

KDF Key Derivation Function

KEK Key Encryption Key

LKH Logical Key Hierarchy

MSEC Multicast Security

SA Security Association

SEQ Sequence Number Payload

SIG Signature Payload

SPI Security Parameter Index

2. Acknowledgement Message Request

When a GM is ready to join a group, it contacts the GCKS with a GROUPKEY-PULL Registration Protocol. When the GCKS has authenticated and verified that the GM is an authorized member of the group it downloads several sets of policy in a Security Association (SA) payload. If the group includes the use of a GROUPKEY-PUSH Rekey Protocol, the SA payload includes an SA Key Encryption Key (KEK) payload (Section 5.3 of [RFC6407]). When necessary the GROUPKEY-PUSH Rekey Protocol also contains an SA payload that includes SA KEK policy. The SA KEK policy indicates how the GM will be receiving and handling the GROUPKEY-PUSH Rekey Protocol.

When the GCKS policy includes the use of the GROUPKEY-PUSH Acknowledgement Message, the GCKS reports this policy to the GM within the SA KEK policy. The GCKS includes a new KEK Attribute with the name KEK_ACK_REQUESTED (value TBD-1), which indicates that the GM is requested to return a GROUPKEY-PUSH Acknowledgement Message.

As part of the SA KEK policy, the GCKS specifies information on the keying material, that is used to protect the GROUPKEY-PUSH Rekey Protocol (e.g., presence of KEK Management Algorithm). Parts of these information are used by a GM to derive the ack_key (defined in Section 3.2), which protects the GROUPKEY-PUSH Acknowledgement Message. There are different types of Rekey Acknowledgement messages, which share an identical message format but differ in the used keying material.

The following values of the KEK_ACK_REQUESTED attribute are defined in this memo.

2.1. REKEY_ACK_KEK_SHA256 Type

This type of Rekey ACK is used when the KEK KD Type (Section 5.6.2 of [RFC6407]) is part of the group policy. The prf (defined in Section 3.2) is PRF-HMAC-SHA-256 [RFC4868]. The base_key (also defined in Section 3.2) is the KEK_ALGORITHM_KEY used to decrypt the GROUPKEY-PUSH message. Note that for some algorithms the KEK_ALGORITHM_KEY will include an explicit Initialization Vector (IV) before the actual key (Section 5.6.2.1 of [RFC6407]), but it is not used in the definition of the base_key.

2.2. REKEY_ACK_LKH_SHA256 Type

This type of Rekey ACK can be used when the KEK_MANAGEMENT_ALGORITHM KEK attribute with a value representing Logical Key Hierarchy (LKH) is part of the group policy (Section 5.3.1.1 of [RFC6407]). The prf is PRF-HMAC-SHA-256. The base_key is the Key Data taken from the first LKH Key structure in an LKH_DOWNLOAD_ARRAY attribute (see Section 5.6.3.1 of [RFC6407]). This is a secret symmetric key that the GCKS shares with the group member. Note that for some algorithms the LKH Key structure will include an explicit IV before the actual key (Section 5.6.3.1 of [RFC6407]), but it is not used in the definition of the base_key.

2.3. REKEY_ACK_KEK_SHA512 Type

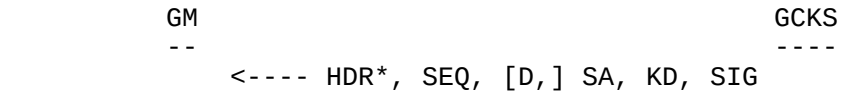
This type of Rekey ACK is identical to the REKEY_ACK_KEK_SHA256 Type, except that the prf is PRF-HMAC-SHA-512 (defined in [RFC4868]).

2.4. REKEY_ACK_LKH_SHA512 Type

This type of Rekey ACK is identical to the REKEY_ACK_LKH_SHA256 Type, except that the prf is PRF-HMAC-SHA-512 (defined in [RFC4868]).

3. GROUPKEY-PUSH Acknowledgement Message

The GROUPKEY-PUSH message defined in [RFC6407] is reproduced in Figure 2. The SA and Key Download (KD) payloads contain the actual policy and keying material being distributed to the GM. The Sequence Number (SEQ) payload contains a sequence number that is used by the GM for replay protection. This sequence number defines a unique rekey message delivered to that GM. One or more Delete (D) payloads optionally specify the deletion of existing group policy. The Signature (SIG) payload includes a signature of a hash of the entire GROUPKEY-PUSH message (excepting the SIG payload octets) before it has been encrypted



* Protected by the Rekey SA KEK; encryption occurs after HDR

Figure 2: GROUPKEY-PUSH from RFC 6407

When the GM has received a KEK_ACK_REQUESTED attribute in an SA KEK and it chooses to respond, it returns the value of the Sequence Number taken from the GROUPKEY-PUSH message to the GCKS along with its identity. This tuple alerts the GCKS that the GM has received the GROUPKEY-PUSH message and implemented the policy contained therein. The GROUPKEY-PUSH Acknowledgement Message is shown in Figure 3.

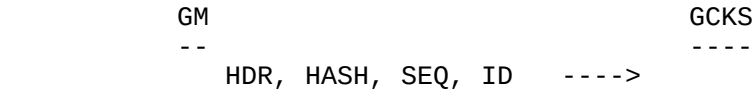


Figure 3: GROUPKEY-PUSH Acknowledgement Message

The IP header for the GROUPKEY-PUSH Acknowledgement Message is constructed as if it were a reply to the GROUPKEY-PUSH message. That is, the Source Address of the GROUPKEY-PUSH message becomes the Destination Address of the GROUPKEY-PUSH Acknowledgement Message and the GM includes its own IP address as the Source Address of the GROUPKEY-PUSH Acknowledgement Message. The Source port in the GROUPKEY-PUSH message UDP header becomes the Destination port of the GROUPKEY-PUSH Acknowledgement Message UDP header, and the Destination

port of the GROUPKEY-PUSH message UDP header becomes the Source port of the GROUPKEY-PUSH Acknowledgement Message UDP header.

The following sections describe the payloads in the GROUPKEY-PUSH Acknowledgement Message.

3.1. HDR

The message begins with a header as defined for the GDOI GROUPKEY-PUSH message in Section 4.1 of [RFC6407]. The fields in the HDR MUST be initialized as follows. The Cookies of a GROUPKEY-PUSH message act as a Security Parameter Index (SPI) and are copied to the Acknowledgement Message. Next Payload identifies a Hash payload (value 8) [ISAKMP-NP]. Major Version is 1 and Minor Version is 0. The Exchange Type has value 35 for the GDOI GROUPKEY-PUSH Acknowledgment Message. Flags are set to 0. Message ID MUST be set to zero. Length is according to Section 4.1 of [RFC6407]).

3.2. HASH

The HASH payload is the same one used in the GDOI GROUPKEY-PULL exchange defined in Section 3.2 of [RFC6407]. The hash data in the HASH payload is created as follows:

```
HASH = prf(ack_key, SEQ | ID)
```

where:

- prf is specific to the KEK_ACK_REQUESTED value, and is described as part of that description.
- "|" indicates concatenation.
- SEQ and ID represent the bytes comprising the Sequence Number and Identification Payloads

The ack_key is computed from a Key Derivation Function (KDF) that conforms to KDF in Feedback Mode as defined in NIST SP800-108 [SP800-108] where the length of the derived keying material is the same as the output of the prf, there is no initialization vector, and the optional counter is not used. Note: When the derived ack_key is smaller than the prf block size (i.e., 512 bits for PRF-HMAC-SHA-256), it is zero filled to the right, as specified in Section 2.1.2 of [RFC4868].

```
ack_key = prf(base_key, "GROUPKEY-PUSH ACK" | SPI | L)
```

where:

- o prf is specific to the KEK_ACK_REQUESTED value, and is described as part of that description.

- o base_key is specific to the KEK_ACK_REQUESTED value, and is described as part of that description. If the base_key is smaller than the prf block size (i.e., 512 bits for PRF-HMAC-SHA-256), then it is zero filled to the right, as specified in Section 2.1.2 of [RFC4868].

- o "|" indicates concatenation.

- o "GROUPKEY-PUSH ACK" is a label encoded as a null terminated ASCII string.

- o SPI is the Initiator Cookie followed by the Responder Cookie taken from the GROUPKEY-PUSH message HDR, which describes the Context of the key usage.

- o L is a length field matching the number of bits in the ack_key. L MUST match the length of the base_key (i.e., 512 bits for PRF-HMAC-SHA-256). The value L is represented as two octets in network byte order (that is, most significant byte first).

3.3. SEQ

The Sequence Number Payload is defined in [RFC6407]. The value in the GROUPKEY-PUSH SEQ payload is copied to the SEQ payload.

3.4. ID

The Identification payload is used as defined in Section 5.1 of [RFC6407]. The ID payload contains an ID Type of ID_IPV4_ADDR, ID_IPV6_ADDR, or ID_OID as defined for GDOI exchanges [RFC8052]. Protocol ID and Port fields MUST be set to 0. The address provided in the ID payload represents the IP address of the GM, and MUST match the source IP address used for the most recent GROUPKEY-PULL exchange.

4. Group Member Operations

When a GM receives an SA KEK payload (in a GROUPKEY-PULL exchange or GROUPKEY-PUSH message) including a KEK_ACK_REQUESTED attribute, it records in its group state some indication that it is expected to return a GROUPKEY-PUSH ACK message. A GM recognizing the attribute MUST honor the KEK_ACK_REQUESTED attribute by returning Acknowledgments, because it can be expected that the GCKS is likely to take some policy-specific action regarding non-responsive GMs, including ceasing to deliver GROUPKEY-PUSH messages to it.

If a GM cannot respond with the requested type of Acknowledgement, it continues with protocol exchange and participates in the group. In any case, if a GM stops receiving GROUPKEY-PUSH messages from a GCKS it will re-register before existing security associations expire, so omitting sending Acknowledgements should not be critical.

When a GM receives a GROUPKEY-PUSH message that contains a KEK_ACK_REQUESTED attribute in the SA KEK payload, it processes the message according to RFC 6407. When it concludes successful processing of the message, it formulates the GROUPKEY-PUSH ACK messages as described in Section 3 and delivers the message to the GCKS from which the GROUPKEY-PUSH message was received. A GROUPKEY-PUSH ACK message is sent even if the GROUPKEY-PUSH message contains a Delete payload for the KEK used to protect the GROUPKEY-PUSH message.

5. GCKS Operations

When a GCKS policy includes requesting a GROUPKEY-PUSH ACK message from Group Members, it includes the KEK_ACK_REQUESTED attribute in the SA KEK payload. It does this each time the SA KEK is delivered, in both GROUPKEY-PULL exchanges and GROUPKEY-PUSH messages. The value of the KEK_ACK_REQUESTED attribute will depend upon the type SA KEK, as described in Section 2.

When a GCKS receives a GROUPKEY-PUSH ACK message (identified by an Exchange type of GROUPKEY-PUSH-ACK), it first verifies that the group policy includes receiving GROUPKEY-PUSH ACK messages. If not, the message is discarded. GCKS implementations SHOULD keep a record (e.g., a hash value) of recently received GROUPKEY-PUSH Acknowledgment messages and reject duplicate messages prior to performing cryptographic operations. This enables an early discard of the replayed messages.

If the message is expected, the GCKS validates the format of the message, and verifies that the HASH has been properly constructed as described in Section 3.2. If validation fails, the message is discarded. The GCKS extracts the sequence number and identity of the GM from the SEQ and ID payloads respectively, and records the fact that the GM received the GROUPKEY-PUSH message represented by its serial number.

6. Management Considerations

The GCKS manages both group policy and group membership of a group. Group membership policy includes a strategy to ensure that rekey messages with current group policy reach all live group members. This is discussed briefly in Section 5.3 of the MSEC Group Key Management Architecture [RFC4046]. The GROUPKEY-PUSH Acknowledgement

message specified in this memo provides the GCKS an additional method to assess if a group member is live and has received the current group policy. But it is possible for a rekey message or GROUPKEY-PUSH Acknowledgement message to be discarded in the network, which results in a live GM to appear unresponsive. Also a GM might not be able to respond with an GROUPKEY-PUSH ACK. So the GCKS should use caution in using a lack of Acknowledgment as the only factor in determining whether a GM is live. In particular, a GCKS SHOULD NOT consider a GM to have left the group until it has received at least one ACK from the GM.

Some management considerations determining how a Group Member handle Acknowledgement messages is as follows:

- A GM MUST respond with Acknowledgement messages when requested, as a GCKS can subsequently determine when a GM becomes unexpectedly non-responsive.

- A GM receiving GROUPKEY-PUSH message as a multicast message MAY introduce a jitter to the timing of its Acknowledgement message to help the GCKS better manage replies from group members. A GM MUST NOT delay sending an Acknowledgment for more than 5 seconds. a GCKS SHOULD NOT declare an Acknowledgment as missing until it has waited at least 10 seconds. Implementations SHOULD make these timers configurable.

Some management considerations determining how the GCKS handles Acknowledgement messages is as follows:

- A non-receipt of an Acknowledgement is an indication that a GM is unable to respond. A GCKS SHOULD wait at least several seconds before determining non-receipt, as GMs could add jitter to the response time before sending an acknowledgement.

- If the GCKS is aware that GMs are expected to respond, then a non-receipt of an Acknowledgement SHOULD trigger a logging event. The GCKS MAY be configured with additional policy actions such as transmitting the GROUPKEY-PUSH message several times in a short period of time (as suggested in [RFC4046]), which mitigates a packet loss of either the GROUPKEY-PUSH message or an Acknowledgement message. Another policy action could be to alerting GCKS administrators of GMs that do not return several consecutive acknowledgement messages or even removing unresponsive GMs from the group. However, a GCKS with a policy of removing GMs from the group needs to be aware that a GM that has not responded will not receive newer group policy until it initiates contact with the GCKS again.

- When a GROUPKEY-PUSH message includes a Delete payload for the KEK used to protect the GROUPKEY-PUSH message, the GCKS SHOULD NOT itself delete the KEK until it has given GMs the opportunity to acknowledge receipt of the GROUPKEY-PUSH message. This could be several seconds, as GMs could add jitter to the response time before sending an acknowledgement.

- A GCKS SHOULD log failure events, such as receiving Acknowledgement messages for a group in which the GCKS has not requested Acknowledgements, receiving malformed Acknowledgement, and Acknowledgements that fail validation.

7. Security Considerations

There are three areas of security considerations to consider: the protection of the GROUPKEY-PUSH ACK message, whether the GM should transmit a GROUPKEY-PUSH ACK, and whether a GCKS should accept a GROUPKEY-PUSH ACK. These are addressed in the following subsections.

The construction of the HASH defined in this memo uses PRF-HMAC-SHA-256 or PRF-HMAC-SHA-512. The strength of these PRFs were unquestioned at the time this memo was developed. When a HASH construction is necessary using a different prf, a new KEK_ACK_REQUESTED value will be defined in a new specification.

7.1. Protection of the GROUPKEY-PUSH ACK

The GROUPKEY-PUSH ACK message is an ISAKMP [RFC2408] message. Message authentication and Man-in-the-Middle Attack Protection is provided by the inclusion of a HASH payload, which includes the output of an HMAC computation over the bytes of the message.

When the value of REKEY_ACK_KEK is specified, because the KEK is a group secret impersonation of a victim GM by another authorized GM is possible. However, security considerations of the impersonation are limited to a false claim that a victim GM has received a GROUPKEY-PUSH when the victim GM has in fact not received it (e.g., because an active attacker has discarded the GROUPKEY-PUSH). If a GCKS policy includes sending retransmissions of the GROUPKEY-PUSH message to that victim GM, then the victim GM might not receive replacement security associations. However, this adds no additional threats over a use case where the GROUPKEY-PUSH ACK is not deployed and GROUPKEY-PUSH messages are withheld from a victim GM by an active attacker. These threats can be mitigated by using a value of REKEY_ACK_LKH, due to the use of a secret pairwise key shared between the GCKS and individual GM.

Confidentiality is not provided for the GROUPKEY-PUSH ACK message. The contents of the message can be observed by a passive attacker, which includes the hash value, the sequence number of in the GROUPKEY-PUSH message to which it is acknowledging receipt, and the identity of the GM. Observation of a hash value or set of hash values will not compromise the hash key. The identity of the GM is also available to the passive attacker as the source IP address of the packet. The sequence number does reveal the sequence number that was included in the GROUPKEY-PUSH, which was previously not available to the attacker. However, the attacker is assumed to not be in possession of the key used to encrypt the message, and thus cannot create a spoofed GROUPKEY-PUSH message. Therefore, there is no direct value that the attacker derives from the knowledge of the sequence number.

7.2. Transmitting a GROUPKEY-PUSH ACK

A GM transmits an ACK only when the policy of the most recently received SA KEK includes a request by the GCKS for ACKs, and only is returned after processing the GROUPKEY-PUSH message according to Section 4.4 of [RFC6407]. In other words, the form of the GROUPKEY-PUSH message will have been validated, replay protection completed, and the digital signature verified as being genuine. Therefore, the threats of a GM responding to a spoofed or resent GROUPKEY-PUSH message, and the possibility of the GM being used to propagate a Distributed Denial of Service (DDoS) attack on a GCKS are mitigated. For more information, see the security considerations of a GROUPKEY-PUSH message described in Section 7.3 of [RFC6407].

7.3. Receiving a GROUPKEY-PUSH ACK

A GCKS receiving ACK messages will follow the validation steps described in Section 5 before interpreting the contents of the message. The GCKS will then be sure to operate only on messages that have been sent by an authorized GM.

A GCKS SHOULD be prepared to receive GROUPKEY-PUSH ACK messages from each GM to which it was sent. That is, needs to ensure it has sufficient resources (e.g., receive queue size) so that it does not unnecessarily drop ACK messages. An GCKS should be aware that a large number of replayed or invalid GROUPKEY-PUSH messages could be addressed to it. However, this is no worse a threat than if it received a large number of other types of replayed or invalid GDOI or other messages containing a HASH payload.

How a GCKS processes the serial number and identity included in an ACK message is a matter of local policy and is outside the scope of this memo.

8. IANA Considerations

The following additions are made to the GDOI Payloads [GDOI-REG] registry.

A new attribute is added to the SA KEK Payload Values - KEK Attributes registry. The ID Class name is KEK_ACK_REQUESTED with a value of TBD-1, and is a Basic attribute.

A new registry defining values for KEK_ACK_REQUESTED is needed, and these values are shown in the following table. The terms Reserved, Unassigned, and Private Use are to be applied as defined in [RFC8126]. The registration procedure is Specification Required.

Value	Type
0	Reserved
1	REKEY_ACK_KEK_SHA256
2	REKEY_ACK_LKH_SHA256
3	REKEY_ACK_KEK_SHA512
4	REKEY_ACK_LKH_SHA512
5-128	Unassigned
129-255	Private Use

A new registry describing ISAKMP Exchange Types for GDOI is added to GDOI Payloads [GDOI-REG]. This registry defines DOI Specific Use values [ISAKMP-EXCH], which are Exchange type values used with the ISAKMP GDOI DOI. Its name is "GDOI DOI Exchange Types". The registration procedure is Specification Required. The terms Known Unregistered Use and Unassigned are to be applied as defined in [RFC8126].

Value	Phase	Reference
GROUPKEY-PULL	32	RFC 6407
GROUPKEY-PUSH	33	RFC 6407
Known Unregistered Use	34	
GROUPKEY-PUSH-ACK	35	RFC XXXX
Unassigned	36-239	

[Note to RFC Editor: Please replace XXXX with the number of the RFC resulting from this memo, and delete this note.]

9. Acknowledgements

Mike Hamada, Adrian Farrel, and Yaron Sheffer provided many useful technical and editorial comments and suggestions for improvement.

10. References

10.1. 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, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4868] Kelly, S. and S. Frankel, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec", RFC 4868, DOI 10.17487/RFC4868, May 2007, <https://www.rfc-editor.org/info/rfc4868>.

[RFC6407] Weis, B., Rowles, S., and T. Hardjono, "The Group Domain of Interpretation", RFC 6407, DOI 10.17487/RFC6407, October 2011, <https://www.rfc-editor.org/info/rfc6407>.

[RFC8052] Weis, B., Seewald, M., and H. Falk, "Group Domain of Interpretation (GDOI) Protocol Support for IEC 62351 Security Services", RFC 8052, DOI 10.17487/RFC8052, June 2017, <https://www.rfc-editor.org/info/rfc8052>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

10.2. Informative References

[GDOI-REG]
Internet Assigned Numbers Authority, "Group Domain of Interpretation (GDOI) Payload Type Values", IANA Registry, November 2016, <http://www.iana.org/assignments/gdoi-payloads/gdoi-payloads.xml>.

[ISAKMP-EXCH]
Internet Assigned Numbers Authority, "Internet Key Exchange (IKE) Attributes Exchange Type Values", IANA Registry, May 2013, <http://www.iana.org/assignments/ipsec-registry/ipsec-registry.xhtml#ipsec-registry-17>.

[ISAKMP-NP]
Internet Assigned Numbers Authority, "Internet Key Exchange (IKE) Attributes Next Protocol Types", IANA Registry, May 2013, <http://www.iana.org/assignments/ipsec-registry/ipsec-registry.xhtml#ipsec-registry-21>.

[RFC2408] Maughan, D., Schertler, M., Schneider, M., and J. Turner, "Internet Security Association and Key Management Protocol (ISAKMP)", RFC 2408, DOI 10.17487/RFC2408, November 1998, <https://www.rfc-editor.org/info/rfc2408>.

[RFC4046] Baugher, M., Canetti, R., Dondeti, L., and F. Lindholm, "Multicast Security (MSEC) Group Key Management Architecture", RFC 4046, DOI 10.17487/RFC4046, April 2005, <https://www.rfc-editor.org/info/rfc4046>.

[SP800-108] Chen, L., "Recommendation for Key Derivation Using Pseudorandom Functions", United States of America, National Institute of Science and Technology, NIST Special Publication 800-108, October 2009, <http://dx.doi.org/10.6028/NIST.SP.800-108>.

Authors' Addresses

Brian Weis
Cisco Systems
170 W. Tasman Drive
San Jose, California 95134-1706
USA

Phone: +1-408-526-4796
Email: bew@cisco.com

Umesh Mangla
Juniper Networks Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA

Phone: +1-408-936-1022
Email: umangla@juniper.net

Thomas Karl
Deutsche Telekom
Landgrabenweg 151
Bonn 53227
Germany

Phone: +49 228 18138122
Email: thomas.karl@telekom.de

Nilesh Maheshwari

Email: nileshm@gmail.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 13, 2018

X. Xu
D. Zhang
L. Xia
Huawei

January 9, 2018

Encapsulating IPsec ESP in UDP for Load-balancing
draft-xu-ipsecme-esp-in-udp-lb-02

Abstract

IPsec Virtual Private Network (VPN) is widely used by enterprises to interconnect their geographical dispersed branch office locations across IP Wide Area Network (WAN) or the Internet, especially in the Software-Defined-WAN (SD-WAN) era. To fully utilize the bandwidth available in IP WAN or the Internet, load balancing of traffic between different IPsec VPN sites over Equal Cost Multi-Path (ECMP) and/or Link Aggregation Group (LAG) is attractive to those enterprises deploying IPsec VPN solutions. This document defines a method to encapsulate IPsec Encapsulating Security Payload (ESP) packets over UDP tunnels for improving load-balancing of IPsec ESP traffic.

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 https://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 July 13, 2018.

Copyright Notice

Table of Contents

1. Introduction

IPsec Virtual Private Network (VPN) is widely used by enterprises to interconnect their geographical dispersed branch office locations across IP Wide Area Network (WAN) or the Internet, especially in the Software-Defined-WAN (SD-WAN) era. To fully utilize the bandwidth available in IP WAN or the Internet, load balancing of traffic between different IPsec VPN sites over Equal Cost Multi-Path (ECMP) and/or Link Aggregation Group (LAG) is much attractive to those enterprises that deploy IPsec VPN solutions. Since most existing core routers within IP WAN or the Internet can already support balancing IP traffic flows based on the hash of the five-tuple of UDP packets, by encapsulating IPsec Encapsulating Security Payload (ESP) packets over UDP tunnels with the UDP source port being used as an entropy field, it will enable existing core routers to perform efficient load-balancing of the IPsec ESP traffic without requiring any change to them. Therefore, this specification defines a method of encapsulating IPsec ESP packets over UDP tunnels for improving load-balancing of IPsec ESP traffic.

Encapsulating ESP in UDP, as defined in this document, can be used in both IPv4 and IPv6 networks. IPv6 flow label has been proposed as an entropy field for load balancing in IPv6 network environment

[RFC6438]. However, as stated in [RFC6936], the end-to-end use of flow labels for load balancing is a long-term solution and therefore the use of load balancing using the transport header fields would continue until any widespread deployment is finally achieved. As such, ESP-in-UDP encapsulation would still have a practical application value in the IPv6 networks during this transition timeframe.

Note that the difference between the ESP-in-UDP encapsulation as proposed in this document and the ESP-in-UDP encapsulation as described in [RFC3948] is that the former uses the UDP tunnel for load-balancing improvement purpose and therefore the source port is used as an entropy field while the latter uses the UDP tunnel for NAT traverse purpose and therefore the source port is set to a constant value (i.e., 4500). In addition, this document only discusses about the tunnel mode ESP encapsulation.

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. Terminology

This memo makes use of the terms defined in [RFC2401]and [RFC2406].

3. Encapsulation in UDP

ESP-in-UDP encapsulation format is shown as follows:

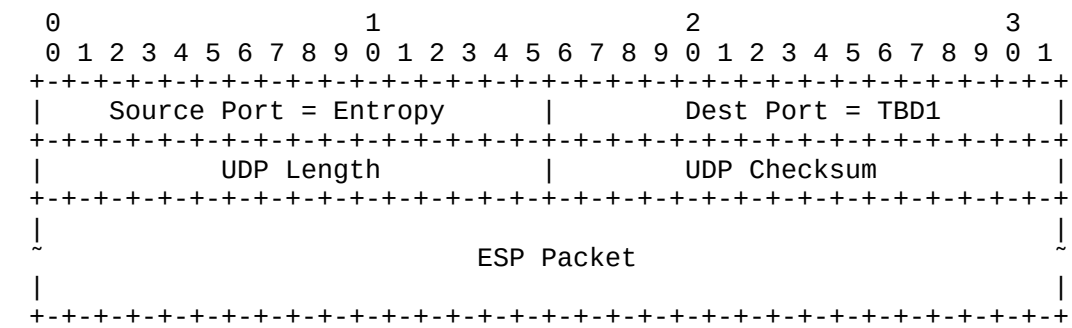


Figure 1: ESP-in-UDP Encapsulation Format

Source Port of UDP:

This field contains a 16-bit entropy value that is generated by the encapsulator to uniquely identify a flow. What constitutes

a flow is locally determined by the encapsulator and therefore is outside the scope of this document. What algorithm is actually used by the encapsulator to generate an entropy value is outside the scope of this document.

In case the tunnel does not need entropy, this field of all packets belonging to a given flow SHOULD be set to a randomly selected constant value so as to avoid packet reordering.

To ensure that the source port number is always in the range 49152 to 65535 (Note that those ports less than 49152 are reserved by IANA to identify specific applications/protocols) which may be required in some cases, instead of calculating a 16-bit hash, the encapsulator SHOULD calculate a 14-bit hash and use those 14 bits as the least significant bits of the source port field while the most significant two bits SHOULD be set to binary 11. That still conveys 14 bits of entropy information which would be enough as well in practice.

Destination Port of UDP:

This field is set to a value (TBD1) allocated by IANA to indicate that the UDP tunnel payload is an ESP packet.

UDP Length:

The usage of this field is in accordance with the current UDP specification [RFC0768].

UDP Checksum:

For IPv4 UDP encapsulation, this field is RECOMMENDED to be set to zero for performance or implementation reasons because the IPv4 header includes a checksum and use of the UDP checksum is optional with IPv4. For IPv6 UDP encapsulation, the IPv6 header does not include a checksum, so this field MUST contain a UDP checksum that MUST be used as specified in [RFC0768] and [RFC2460] unless one of the exceptions that allows use of UDP zero-checksum mode (as specified in [RFC6935]) applies.

ESP Packet:

This field contains one ESP packet.

4. Processing Procedures

This ESP-in-UDP encapsulation causes ESP [RFC2406] packets to be forwarded across IP WAN via "UDP tunnels". When performing ESP-in-UDP encapsulation by an IPsec VPN gateway, ordinary ESP encapsulation procedure is performed and then a formatted UDP header is inserted between ESP header and IP header. The Source Port field of the UDP header is filled with an entropy value which is generated by the IPsec VPN gateway. Upon receiving these UDP encapsulated packets, remote IPsec VPN gateway MUST decapsulate these packets by removing the UDP header and then perform ordinary ESP decapsulation procedure consequently.

Similar to all other IP-based tunneling technologies, ESP-in-UDP encapsulation introduces overheads and reduces the effective Maximum Transmission Unit (MTU) size. ESP-in-UDP encapsulation may also impact Time-to-Live (TTL) or Hop Count (HC) and Differentiated Services (DSCP). Hence, ESP-in-UDP MUST follow the corresponding procedures defined in [RFC2003].

Encapsulators MUST NOT fragment ESP packet, and when the outer IP header is IPv4, encapsulators MUST set the DF bit in the outer IPv4 header. It is strongly RECOMMENDED that IP transit core be configured to carry an MTU at least large enough to accommodate the added encapsulation headers. Meanwhile, it is strongly RECOMMENDED that Path MTU Discovery [RFC1191] [RFC1981] or Packetization Layer Path MTU Discovery (PLPMTUD) [RFC4821] is used to prevent or minimize fragmentation.

5. Congestion Considerations

TBD.

6. Applicability Statements

TBD.

7. Acknowledgements

8. IANA Considerations

One UDP destination port number indicating ESP needs to be allocated by IANA:

Service Name: ESP-in-UDP Transport Protocol(s):UDP
Assignee: IESG <iesg@ietf.org>
Contact: IETF Chair <chair@ietf.org>.
Description: Encapsulate ESP packets in UDP tunnels.
Reference: This document.
Port Number: TBD1 -- To be assigned by IANA.

9. Security Considerations

TBD.

10. References

10.1. Normative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <https://www.rfc-editor.org/info/rfc1191>.

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, DOI 10.17487/RFC1981, August 1996, <https://www.rfc-editor.org/info/rfc1981>.

[RFC2003] Perkins, C., "IP Encapsulation within IP", RFC 2003, DOI 10.17487/RFC2003, October 1996, <https://www.rfc-editor.org/info/rfc2003>.

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

[RFC2401] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, DOI 10.17487/RFC2401, November 1998, <https://www.rfc-editor.org/info/rfc2401>.

[RFC2406] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, DOI 10.17487/RFC2406, November 1998, <https://www.rfc-editor.org/info/rfc2406>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <https://www.rfc-editor.org/info/rfc2460>.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <https://www.rfc-editor.org/info/rfc4821>.

[RFC6438] Carpenter, B. and S. Amante, "Using the IPv6 Flow Label for Equal Cost Multipath Routing and Link Aggregation in Tunnels", RFC 6438, DOI 10.17487/RFC6438, November 2011, <https://www.rfc-editor.org/info/rfc6438>.

[RFC6935] Eubanks, M., Chimento, P., and M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets", RFC 6935, DOI 10.17487/RFC6935, April 2013, <https://www.rfc-editor.org/info/rfc6935>.

[RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, DOI 10.17487/RFC6936, April 2013, <https://www.rfc-editor.org/info/rfc6936>.

10.2. Informative References

[RFC3948] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", RFC 3948, DOI 10.17487/RFC3948, January 2005, <https://www.rfc-editor.org/info/rfc3948>.

Authors' Addresses

Xiaohu Xu
Huawei

Email: xuxh.mail@gmail.com

Dacheng Zhang
Huawei

Email: dacheng.zhang@huawei.com

Liang Xia
Huawei

Email: frank.xialiang@huawei.com