

ADD
Internet-Draft
Intended status: Standards Track
Expires: May 12, 2022

M. Boucadair
Orange
T. Reddy
McAfee
D. Wing
Citrix
V. Smyslov
ELVIS-PLUS
November 8, 2021

Internet Key Exchange Protocol Version 2 (IKEv2) Configuration for
Encrypted DNS
draft-btw-add-ipsecme-ike-04

Abstract

This document specifies a new Internet Key Exchange Protocol Version 2 (IKEv2) Configuration Payload Attribute Types for encrypted DNS protocols such as DNS-over-HTTPS (DoH), DNS-over-TLS (DoT), and DNS-over-QUIC (DoQ).

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 May 12, 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. IKEv2 Configuration Payload Attribute Types for Encrypted DNS	3
3.1. ENCDNS_IP* Configuration Payload Attributes	3
3.2. ENCDNS_DIGEST_INFO Configuration Payload Attribute	5
4. IKEv2 Protocol Exchange	7
5. Security Considerations	8
6. IANA Considerations	9
6.1. Configuration Payload Attribute Types	9
7. Acknowledgements	9
8. References	9
8.1. Normative References	9
8.2. Informative References	10
Appendix A. Sample Deployment Scenarios	11
A.1. Roaming Enterprise Users	11
A.2. VPN Service Provider	12
A.3. DNS Offload	12
Authors' Addresses	12

1. Introduction

This document specifies encrypted DNS configuration for an Internet Key Exchange Protocol Version 2 (IKEv2) [RFC7296] initiator, particularly the Authentication Domain Name (ADN) of encrypted DNS protocols such as DNS-over-HTTPS (DoH) [RFC8484], DNS-over-TLS (DoT) [RFC7858], or DNS-over-QUIC (DoQ) [I-D.ietf-dprive-dnssoquic].

This document introduces new IKEv2 Configuration Payload Attribute Types (Section 3) for the support of encrypted DNS servers. These attributes can be used to provision authentication domain names, a list of IP addresses, and a set of service parameters.

Sample use cases are discussed in Appendix A. The Configuration Payload Attribute Types defined in this document are not specific to these deployments, but can also be used in other deployment contexts. It is out of the scope of this document to provide a comprehensive list of deployment contexts.

The encrypted DNS server hosted by the VPN provider can get a domain-validate certificate from a public CA. The VPN client does not need

to be provisioned with the root certificate of a private CA to authenticate the certificate of the encrypted DNS server. The encrypted DNS server can run on private IP addresses and its access can be restricted to clients connected to the VPN.

Note that, for many years, typical designs have often considered that the DNS server was usually located inside the protected domain, but could be located outside of it. With encrypted DNS, the latter option becomes plausible.

2. Terminology

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.

This document uses of the terms defined in [RFC8499].

Also, this document uses of the terms defined in [RFC7296]. In particular, readers should be familiar with "initiator" and "responder" terms used in that document.

This document makes use of the following terms:

'Do53': refers to unencrypted DNS.

'Encrypted DNS': refers to a scheme where DNS messages are sent over an encrypted channel. Examples of encrypted DNS are DoT, DoH, and DoQ.

'ENCDNS_IP*': refers to any IKEv2 Configuration Payload Attribute Types defined in Section 3.1.

3. IKEv2 Configuration Payload Attribute Types for Encrypted DNS

3.1. ENCDNS_IP* Configuration Payload Attributes

The ENCDNS_IP* IKEv2 Configuration Payload Attribute Types are used to configure encrypted DNS servers. All these attributes share the format shown in Figure 1.

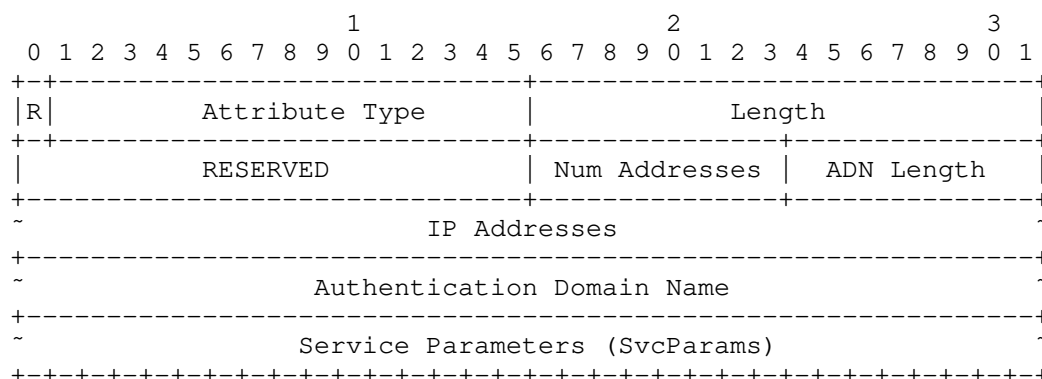


Figure 1: Attributes Format

The fields of the attribute shown in Figure 1 are as follows:

- o R (Reserved, 1 bit) - This bit MUST be set to zero and MUST be ignored on receipt (see Section 3.15.1 of [RFC7296] for details).
- o Attribute Type (15 bits) - Identifier for Configuration Attribute Type; is set to TBA1 or TBA2 values listed in Section 6.1.
- o Length (2 octets, unsigned integer) - Length of the data in octets. In particular, this field is set to:
 - * 0 if the Configuration payload has types CFG_REQUEST or CFG_ACK.
 - * (4 + Length of the ADN + N * 4 + Length of SvcParams) for ENCDNS_IP4 attributes if the Configuration payload has types CFG_REPLY or CFG_SET; N being the number of included IPv4 addresses ('Num addresses').
 - * (4 + Length of the ADN + N * 16 + Length of SvcParams) for ENCDNS_IP6 attributes if the Configuration payload has types CFG_REPLY or CFG_SET; N being the number of included IPv6 addresses ('Num addresses').
- o RESERVED (2 octets) - These bits are reserved for future use. These bits MUST be set to zero by the sender and MUST be ignored by the receiver.
- o Num Addresses (1 octet) - Indicates the number of enclosed IPv4 (for ENCDNS_IP4 attribute type) or IPv6 (for ENCDNS_IP6 attribute type) addresses. It MUST NOT be set to 0.

- o ADN Length (1 octet) - Indicates the length of the authentication-domain-name field in octets.
- o IP Address(es) (variable) - One or more IPv4 or IPv6 addresses to be used to reach the encrypted DNS server that is identified by the name in the Authentication Domain Name.
- o Authentication Domain Name (variable) - A fully qualified domain name of the encrypted DNS server following the syntax defined in [RFC5890]. The name MUST NOT contain any terminators (e.g., NULL, CR).

An example of a valid ADN for DoH server is "doh1.example.com".

- o Service Parameters (SvcParams) (variable) - Specifies a set of service parameters that are encoded following the rules in Section 2.1 of [I-D.ietf-dnsop-svcb-https]. Service parameters may include, for example, a list of ALPN protocol identifiers or alternate port numbers. The service parameters MUST NOT include "ipv4hint" or "ipv6hint" SvcParams as they are superseded by the included IP addresses.

If no port service parameter is included, this indicates that default port numbers should be used. As a reminder, the default port number is 853 for DoT and 443 for DoH.

The service parameters apply to all IP addresses in the ENCDNS_IP* Configuration Payload Attribute.

3.2. ENCDNS_DIGEST_INFO Configuration Payload Attribute

The format of ENCDNS_DIGEST_INFO configuration payload attribute is shown in Figure 2.

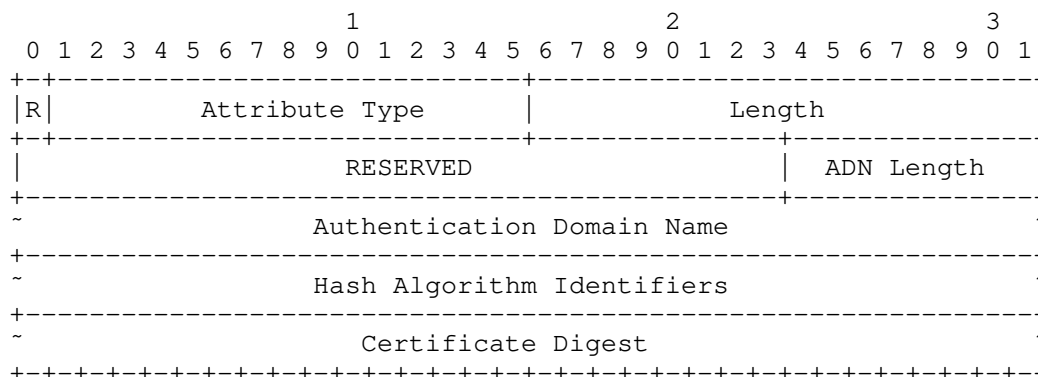


Figure 2: ENCDNS_DIGEST_INFO Attribute Format

- o R (Reserved, 1 bit) - This bit MUST be set to zero and MUST be ignored on receipt (see Section 3.15.1 of [RFC7296] for details).
- o Attribute Type (15 bits) - Identifier for Configuration Attribute Type; is set to TBA3 value listed in Section 6.1.
- o Length (2 octets, unsigned integer) - Length of the data in octets.
- o RESERVED (2 octets) - These bits are reserved for future use. These bits MUST be set to zero by the sender and MUST be ignored by the receiver.
- o ADN Length (1 octet) - Indicates the length of the authentication-domain-name field in octets. When set to '0', this means that the digest applies on the ADN conveyed in the ENCDNS_IP* Configuration Payload Attribute(s).
- o Authentication Domain Name (variable) - A fully qualified domain name of the encrypted DNS server following the syntax defined in [RFC5890]. The name MUST NOT contain any terminators (e.g., NULL, CR). A name is included only when multiple ADNs are included in the ENCDNS_IP* Configuration Payload Attributes.
- o Hash Algorithm Identifiers (variable) - In a request, this field specifies a list of 16-bit hash algorithm identifiers that are supported by the Encrypted DNS client. In a response, this field specified the 16-bit hash algorithm identifier selected by the server to generate the digest of its certificate. The values of this field are taken from the Hash Algorithm Identifiers of IANA's "Internet Key Exchange Version 2 (IKEv2) Parameters" registry [Hash].

There is no padding between the hash algorithm identifiers.

Note that SHA2-256 is mandatory to implement.

- o Certificate Digest (variable) - MUST only be present in a response. This field includes the digest of the Encrypted DNS server certificate using the algorithm identified in the 'Hash Algorithm Identifiers' field.

4. IKEv2 Protocol Exchange

This section describes how an initiator can be configured with an encrypted DNS server (e.g., DoH, DoT) using IKEv2.

Initiators indicate the support of an encrypted DNS in the CFG_REQUEST payloads by including one or two ENCDNS_IP* attributes, while responders supply the encrypted DNS configuration in the CFG_REPLY payloads. Concretely:

If the initiator supports encrypted DNS, it includes one or two ENCDNS_IP* attributes in the CFG_REQUEST. For each IP address family the initiator MUST include exactly one attribute with the Length field set to 0, so that no data is included for these attributes. The initiator MAY include the ENCDNS_DIGEST_INFO attribute with a list of hash algorithms that are supported by the Encrypted DNS client.

For each ENCDNS_IP* attribute from the CFG_REQUEST, if the responder supports the corresponding address family, and absent any policy, the responder sends back ENCDNS_IP* attribute(s) in the CFG_REPLY with an appropriate list of IP addresses, service parameters, and an ADN. The list of IP addresses MUST include at least one IP address. Multiple instances of the same ENCDNS_IP* attribute MAY be returned if distinct ADNs or service parameters are to be returned by the responder. The same or distinct IP addresses can be returned in such instances. In addition, the responder MAY return the ENCDNS_DIGEST_INFO attribute to convey a digest of the certificate of the Encrypted DNS and the identifier of the hash algorithm that is used to generate the digest.

The behavior of the responder if it receives both ENCDNS_IP* and INTERNAL_IP6_DNS (or INTERNAL_IP4_DNS) attributes is policy-based and deployment-specific. However, it is RECOMMENDED that if the responder includes at least one ENCDNS_IP* attribute in the reply, it should not include any of INTERNAL_IP4_DNS/INTERNAL_IP6_DNS attributes.

If the CFG_REQUEST includes an ENCDNS_IP* attribute but the CFG_REPLY does not include an ENCDNS_IP* matching the requested address family, this is an indication that requested address family is not supported by the responder or the responder is not configured to provide corresponding server addresses.

The DNS client establishes an encrypted DNS session (e.g., DoT, DoH, DoQ) with the address(es) conveyed in ENCDNS_IP* and uses the mechanism discussed in Section 8 of [RFC8310] to authenticate the DNS server certificate using the authentication domain name conveyed in ENCDNS_IP*.

If the CFG_REPLY includes an ENCDNS_DIGEST_INFO attribute, the DNS client has to create a digest of the DNS server certificate received in the TLS handshake using the negotiated hash algorithm in the ENCDNS_DIGEST_INFO attribute. If the computed digest for an ADN matches the one sent in the ENCDNS_DIGEST_INFO attribute, the encrypted DNS server certificate is successfully validated. If so, the client continues with the TLS connection as normal. Otherwise, the client MUST treat the server certificate validation failure as a non-recoverable error. This approach is similar to certificate usage PKIX-EE(1) defined in [RFC7671].

If the IPsec connection is a split-tunnel configuration and the initiator negotiated INTERNAL_DNS_DOMAIN as per [RFC8598], the DNS client MUST resolve the internal names using ENCDNS_IP* DNS servers.

Note: [RFC8598] requires INTERNAL_IP6_DNS (or INTERNAL_IP4_DNS) attribute to be mandatory present when INTERNAL_DNS_DOMAIN is included. This specification relaxes that constraint in the presence of ENCDNS_IP* attributes.

5. Security Considerations

This document adheres to the security considerations defined in [RFC7296]. In particular, this document does not alter the trust on the DNS configuration provided by a responder.

Networks are susceptible to internal attacks as discussed in Section 3.2 of [I-D.arkko-farrell-arch-model-t]. Hosting encrypted DNS server even in case of split-VPN configuration minimizes the attack vector (e.g., a compromised network device cannot monitor/modify DNS traffic). This specification describes a mechanism to restrict access to the DNS messages to only the parties that need to know.

The initiator may trust the encrypted DNS servers supplied by means of IKEv2 from a trusted responder more than the locally provided DNS

servers, especially in the case of connecting to unknown or untrusted networks (e.g., coffee shops or hotel networks).

If IKEv2 responder has used NULL Authentication method [RFC7619] to authenticate itself, the initiator MUST NOT use returned ENCDNS_IP* servers configuration unless it is pre-configured in the OS or the browser.

This specification does not extend the scope of accepting DNSSEC trust anchors beyond the usage guidelines defined in Section 6 of [RFC8598].

6. IANA Considerations

6.1. Configuration Payload Attribute Types

This document requests IANA to assign the following new IKEv2 Configuration Payload Attribute Types from the "IKEv2 Configuration Payload Attribute Types" namespace available at <https://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml#ikev2-parameters-21>.

Value	Attribute Type	Multi-Valued	Length	Reference
TBA1	ENCDNS_IP4	YES	0 or more	RFC XXXX
TBA2	ENCDNS_IP6	YES	0 or more	RFC XXXX
TBA3	ENCDNS_ENCDNS_DIGEST_INFO	YES	0 or more	RFC XXXX

7. Acknowledgements

Many thanks to Yoav Nir, Christian Jacquenet, Paul Wouters, and Tommy Pauly for the review and comments.

Yoav and Paul suggested the use of one single attribute carrying both the name and an IP address instead of depending on the existing INTERNAL_IP6_DNS and INTERNAL_IP4_DNS attributes.

Christian Huitema suggested to return a port number in the attributes.

8. References

8.1. Normative References

[Hash] "IKEv2 Hash Algorithms",
<<https://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml#hash-algorithms>>.

- [I-D.ietf-dnsop-svcb-https]
Schwartz, B., Bishop, M., and E. Nygren, "Service binding and parameter specification via the DNS (DNS SVCB and HTTPS RRs)", draft-ietf-dnsop-svcb-https-08 (work in progress), October 2021.
- [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>>.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <<https://www.rfc-editor.org/info/rfc5890>>.
- [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>>.
- [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>>.
- [RFC8310] Dickinson, S., Gillmor, D., and T. Reddy, "Usage Profiles for DNS over TLS and DNS over DTLS", RFC 8310, DOI 10.17487/RFC8310, March 2018, <<https://www.rfc-editor.org/info/rfc8310>>.

8.2. Informative References

- [I-D.arkko-farrell-arch-model-t]
Arkko, J. and S. Farrell, "Challenges and Changes in the Internet Threat Model", draft-arkko-farrell-arch-model-t-04 (work in progress), July 2020.
- [I-D.ietf-dprive-dnsquic]
Huitema, C., Dickinson, S., and A. Mankin, "DNS over Dedicated QUIC Connections", draft-ietf-dprive-dnsquic-06 (work in progress), October 2021.
- [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, <<https://www.rfc-editor.org/info/rfc7619>>.

- [RFC7671] Dukhovni, V. and W. Hardaker, "The DNS-Based Authentication of Named Entities (DANE) Protocol: Updates and Operational Guidance", RFC 7671, DOI 10.17487/RFC7671, October 2015, <<https://www.rfc-editor.org/info/rfc7671>>.
- [RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <<https://www.rfc-editor.org/info/rfc7858>>.
- [RFC8484] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, DOI 10.17487/RFC8484, October 2018, <<https://www.rfc-editor.org/info/rfc8484>>.
- [RFC8499] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", BCP 219, RFC 8499, DOI 10.17487/RFC8499, January 2019, <<https://www.rfc-editor.org/info/rfc8499>>.
- [RFC8598] Pauly, T. and P. Wouters, "Split DNS Configuration for the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 8598, DOI 10.17487/RFC8598, May 2019, <<https://www.rfc-editor.org/info/rfc8598>>.

Appendix A. Sample Deployment Scenarios

A.1. Roaming Enterprise Users

In this Enterprise scenario (Section 1.1.3 of [RFC7296]), a roaming user connects to the Enterprise network through an IPsec tunnel. The split-tunnel Virtual Private Network (VPN) configuration allows the endpoint to access hosts that resides in the Enterprise network [RFC8598] using that tunnel; other traffic not destined to the Enterprise does not traverse the tunnel. In contrast, a non-split-tunnel VPN configuration causes all traffic to traverse the tunnel into the enterprise.

For both split- and non-split-tunnel configurations, the use of encrypted DNS instead of Do53 provides privacy and integrity protection along the entire path (rather than just to the VPN termination device) and can communicate the encrypted DNS server policies.

For split-tunnel VPN configurations, the endpoint uses the Enterprise-provided encrypted DNS server to resolve internal-only domain names.

For non-split-tunnel VPN configurations, the endpoint uses the Enterprise-provided encrypted DNS server to resolve both internal and external domain names.

Enterprise networks are susceptible to internal and external attacks. To minimize that risk all enterprise traffic is encrypted (Section 2.1 of [I-D.arkko-farrell-arch-model-t]).

A.2. VPN Service Provider

Legacy VPN service providers usually preserve end-users' data confidentiality by sending all communication traffic through an encrypted tunnel. A VPN service provider can also provide guarantees about the security of the VPN network by filtering malware and phishing domains.

Browsers and OSes support DoH/DoT; VPN providers may no longer expect DNS clients to fallback to Do53 just because it is a closed network.

The encrypted DNS server hosted by the VPN service provider can be securely discovered by the endpoint using the IKEv2 Configuration Payload Attribute Type.

A.3. DNS Offload

VPN service providers typically allow split-tunnel VPN configuration in which users can choose applications that can be excluded from the tunnel. For example, users may exclude applications that restrict VPN access.

The encrypted DNS server hosted by the VPN service provider can be securely discovered by the endpoint using the IKEv2 Configuration Payload Attribute Type.

Authors' Addresses

Mohamed Boucadair
Orange
Rennes 35000
France

Email: mohamed.boucadair@orange.com

Tirumaleswar Reddy
McAfee, Inc.
Embassy Golf Link Business Park
Bangalore, Karnataka 560071
India

Email: TirumaleswarReddy_Konda@McAfee.com

Dan Wing
Citrix Systems, Inc.
USA

Email: dwing-ietf@fuggles.com

Valery Smyslov
ELVIS-PLUS
RU

Email: svan@elvis.ru

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 6 September 2022

V. Smyslov
ELVIS-PLUS
5 March 2022

Intermediate Exchange in the IKEv2 Protocol
draft-ietf-ipsecme-ikev2-intermediate-10

Abstract

This document defines a new exchange, called Intermediate Exchange, for the Internet Key Exchange protocol Version 2 (IKEv2). This exchange can be used for transferring large amounts of data in the process of IKEv2 Security Association (SA) establishment. An example of the need to do this is using Quantum Computer resistant key exchange methods for IKE SA establishment. Introducing the Intermediate Exchange allows re-using the existing IKE fragmentation mechanism, that helps to avoid IP fragmentation of large IKE messages, but cannot be used in the initial IKEv2 exchange.

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 6 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components

extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	2
2. Terminology and Notation	4
3. Intermediate Exchange Details	4
3.1. Support for Intermediate Exchange Negotiation	4
3.2. Using Intermediate Exchange	4
3.3. The IKE_INTERMEDIATE Exchange Protection and Authentication	5
3.3.1. Protection of the IKE_INTERMEDIATE Messages	5
3.3.2. Authentication of the IKE_INTERMEDIATE Exchanges	6
3.4. Error Handling in the IKE_INTERMEDIATE Exchange	10
4. Interaction with other IKEv2 Extensions	11
5. Security Considerations	11
6. IANA Considerations	12
7. Implementation Status	13
8. Acknowledgements	13
9. References	13
9.1. Normative References	13
9.2. Informative References	14
Appendix A. Example of IKE_INTERMEDIATE exchange	14
Author's Address	16

1. Introduction

The Internet Key Exchange protocol version 2 (IKEv2) defined in [RFC7296] uses UDP as a transport for its messages. If the size of a message is larger than the PMTU, IP fragmentation takes place, which has been shown to cause operational challenge in certain network configurations and devices. The problem is described in more detail in [RFC7383], which also defines an extension to IKEv2 called IKE fragmentation. This extension allows IKE messages to be fragmented at the IKE level, eliminating possible issues caused by IP fragmentation. However, IKE fragmentation cannot be used in the initial IKEv2 exchange (IKE_SA_INIT). This limitation in most cases is not a problem, since the IKE_SA_INIT messages are usually small enough not to cause IP fragmentation.

However, the situation has been changing recently. One example of the need to transfer large amount of data before an IKE SA is created is using Quantum Computer resistant key exchange methods in IKEv2. Recent progress in Quantum Computing has brought a concern that classical Diffie-Hellman key exchange methods will become insecure in a relatively near future and should be replaced with Quantum Computer

(QC) resistant ones. Currently, most QC-resistant key exchange methods have large public keys. If these keys are exchanged in the IKE_SA_INIT, then most probably IP fragmentation will take place, therefore all the problems caused by it will become inevitable.

A possible solution to the problem would be to use TCP as a transport for IKEv2, as defined in [RFC8229]. However, this approach has significant drawbacks and is intended to be a "last resort" when UDP transport is completely blocked by intermediate network devices.

This specification describes a way to transfer a large amount of data in IKEv2 using UDP transport. For this purpose the document defines a new exchange for the IKEv2 protocol, called Intermediate Exchange or IKE_INTERMEDIATE. One or more these exchanges may take place right after the IKE_SA_INIT exchange and prior to the IKE_AUTH exchange. The IKE_INTERMEDIATE exchange messages can be fragmented using the IKE fragmentation mechanism, so these exchanges may be used to transfer large amounts of data which don't fit into the IKE_SA_INIT exchange without causing IP fragmentation.

The Intermediate Exchange can be used to transfer large public keys of QC-resistant key exchange methods, but its application is not limited to this use case. This exchange can also be used whenever some data need to be transferred before the IKE_AUTH exchange and for some reason the IKE_SA_INIT exchange is not suited for this purpose. This document defines the IKE_INTERMEDIATE exchange without tying it to any specific use case. It is expected that separate specifications will define for which purposes and how the IKE_INTERMEDIATE exchange is used in IKEv2. Some considerations must be taken into account when designing such specifications:

- * The IKE_INTERMEDIATE exchange is not intended for bulk transfer. This document doesn't set a hard cap on the amount of data that can be safely transferred using this mechanism, as it depends on its application. But it is anticipated that in most cases the amount of data will be limited to tens of Kbytes (few hundred Kbytes in extreme cases), which is believed to cause no network problems (see [RFC6928] as an example of experiments with sending similar amounts of data in the first TCP flight). See also Section 5 for the discussion of possible DoS attack vectors when amount of data sent in IKE_INTERMEDIATE is too large.
- * It is expected that the IKE_INTERMEDIATE exchange will only be used for transferring data that is needed to establish IKE SA and not for data that can be send later when this SA is established.

2. Terminology and Notation

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.

It is expected that readers are familiar with the terms used in the IKEv2 specification [RFC7296].

3. Intermediate Exchange Details

3.1. Support for Intermediate Exchange Negotiation

The initiator indicates its support for Intermediate Exchange by including a notification of type `INTERMEDIATE_EXCHANGE_SUPPORTED` in the `IKE_SA_INIT` request message. If the responder also supports this exchange, it includes this notification in the response message.

Initiator	Responder
-----	-----
HDR, SAi1, KEi, Ni, [N(INTERMEDIATE_EXCHANGE_SUPPORTED)] -->	<-- HDR, SAR1, KEr, Nr, [CERTREQ], [N(INTERMEDIATE_EXCHANGE_SUPPORTED)]

The `INTERMEDIATE_EXCHANGE_SUPPORTED` is a Status Type IKEv2 notification. Its Notify Message Type is 16438, Protocol ID and SPI Size are both set to 0. This specification doesn't define any data that this notification may contain, so the Notification Data is left empty. However, future enhancements to this specification may override this. Implementations MUST ignore non-empty Notification Data if they don't understand its purpose.

3.2. Using Intermediate Exchange

If both peers indicated their support for the Intermediate Exchange, the initiator may use one or more these exchanges to transfer additional data. Using the Intermediate Exchange is optional; the initiator may find it unnecessary even when support for this exchanged has been negotiated.

The Intermediate Exchange is denoted as `IKE_INTERMEDIATE`, its Exchange Type is 43.

Initiator	Responder
-----	-----
HDR, ..., SK {...} -->	<-- HDR, ..., SK {...}

The initiator may use several IKE_INTERMEDIATE exchanges if necessary. Since window size is initially set to one for both peers (Section 2.3 of [RFC7296]), these exchanges MUST be sequential and MUST all be completed before the IKE_AUTH exchange is initiated. The IKE SA MUST NOT be considered as established until the IKE_AUTH exchange is successfully completed.

The Message IDs for IKE_INTERMEDIATE exchanges MUST be chosen according to the standard IKEv2 rule, described in the Section 2.2. of [RFC7296], i.e. it is set to 1 for the first IKE_INTERMEDIATE exchange, 2 for the next (if any) and so on. Implementations MUST verify that Message IDs in the IKE_INTERMEDIATE messages they receive actually follow this rule. The Message ID for the first pair of the IKE_AUTH messages is one more than the value used in the last IKE_INTERMEDIATE exchange.

If the presence of NAT is detected in the IKE_SA_INIT exchange via NAT_DETECTION_SOURCE_IP and NAT_DETECTION_DESTINATION_IP notifications, then the peers switch to port 4500 in the first IKE_INTERMEDIATE exchange and use this port for all subsequent exchanges, as described in Section 2.23 of [RFC7296].

The content of the IKE_INTERMEDIATE exchange messages depends on the data being transferred and will be defined by specifications utilizing this exchange. However, since the main motivation for the IKE_INTERMEDIATE exchange is to avoid IP fragmentation when large amounts of data need to be transferred prior to IKE_AUTH, the Encrypted payload MUST be present in the IKE_INTERMEDIATE exchange messages and payloads containing large data MUST be placed inside it. This will allow IKE fragmentation [RFC7383] to take place, provided it is supported by the peers and negotiated in the initial exchange.

Appendix A contains an example of using an IKE_INTERMEDIATE exchange in creating an IKE SA.

3.3. The IKE_INTERMEDIATE Exchange Protection and Authentication

3.3.1. Protection of the IKE_INTERMEDIATE Messages

The keys SK_e[i/r] and SK_a[i/r] for the IKE_INTERMEDIATE exchanges protection are computed in the standard fashion, as defined in the Section 2.14 of [RFC7296].

Every subsequent IKE_INTERMEDIATE exchange uses the most recently calculated IKE SA keys before this exchange is started. So, the first IKE_INTERMEDIATE exchange always uses SK_e[i/r] and SK_a[i/r] keys that were computed as a result of the IKE_SA_INIT exchange. If additional key exchange is performed in the first IKE_INTERMEDIATE exchange, resulting in the update of SK_e[i/r] and SK_a[i/r], then these updated keys are used for protection of the second IKE_INTERMEDIATE exchange. Otherwise, the original SK_e[i/r] and SK_a[i/r] keys are used again, and so on.

Once all the IKE_INTERMEDIATE exchanges are completed, the most recently calculated SK_e[i/r] and SK_a[i/r] keys are used for protection of the IKE_AUTH and all the subsequent exchanges.

3.3.2. Authentication of the IKE_INTERMEDIATE Exchanges

The IKE_INTERMEDIATE messages must be authenticated in the IKE_AUTH exchange, which is performed by adding their content into the AUTH payload calculation. It is anticipated that in many use cases IKE_INTERMEDIATE messages will be fragmented using IKE fragmentation [RFC7383] mechanism. According to [RFC7383], when IKE fragmentation is negotiated, the initiator may first send a request message in unfragmented form, but later turn on IKE fragmentation and re-send it fragmented if no response is received after a few retransmissions. In addition, peers may re-send fragmented message using different fragment sizes to perform simple PMTU discovery.

The requirement to support this behavior makes authentication challenging: it is not appropriate to add on-the-wire content of the IKE_INTERMEDIATE messages into the AUTH payload calculation, because implementations are generally unaware in which form these messages are received by peers. Instead, a more complex scheme is used -- authentication is performed by adding content of these messages before their encryption and possible fragmentation, so that data to be authenticated doesn't depend on the form the messages are delivered in.

If any IKE_INTERMEDIATE exchange took place, the definition of the blob to be signed (or MAC'ed) from the Section 2.15 of [RFC7296] is modified as follows:

```

InitiatorSignedOctets = RealMsg1 | NonceRData | MACedIDForI | IntAuth
ResponderSignedOctets = RealMsg2 | NonceIData | MACedIDForR | IntAuth

```

```

IntAuth = IntAuth_iN | IntAuth_rN | IKE_AUTH_MID

```

```

IntAuth_i1 = prf(SK_pi1, IntAuth_i1A [| IntAuth_i1P])
IntAuth_i2 = prf(SK_pi2, IntAuth_i1 | IntAuth_i2A [| IntAuth_i2P])
IntAuth_i3 = prf(SK_pi3, IntAuth_i2 | IntAuth_i3A [| IntAuth_i3P])
...
IntAuth_iN = prf(SK_piN, IntAuth_iN-1 | IntAuth_iNA [| IntAuth_iNP])

IntAuth_r1 = prf(SK_pr1, IntAuth_r1A [| IntAuth_r1P])
IntAuth_r2 = prf(SK_pr2, IntAuth_r1 | IntAuth_r2A [| IntAuth_r2P])
IntAuth_r3 = prf(SK_pr3, IntAuth_r2 | IntAuth_r3A [| IntAuth_r3P])
...
IntAuth_rN = prf(SK_prN, IntAuth_rN-1 | IntAuth_rNA [| IntAuth_rNP])

```

The essence of this modification is that a new chunk called IntAuth is appended to the string of octets that is signed (or MAC'ed) by the peers. IntAuth consists of three parts: IntAuth_iN, IntAuth_rN, and IKE_AUTH_MID.

The IKE_AUTH_MID chunk is a value of the Message ID field from the IKE Header of the first round of the IKE_AUTH exchange. It is represented as a four octet integer in network byte order (in other words, exactly as it appears on the wire).

The IntAuth_iN and IntAuth_rN chunks each represent the cumulative result of applying the negotiated prf to all IKE_INTERMEDIATE exchange messages sent during IKE SA establishment by the initiator and the responder respectively. After the first IKE_INTERMEDIATE exchange is completed peers calculate the IntAuth_i1 value by applying the negotiated prf to the content of the request message from this exchange and calculate the IntAuth_r1 value by applying the negotiated prf to the content of the response message. For every following IKE_INTERMEDIATE exchange (if any) peers re-calculate these values as follows. After the n-th exchange is completed they compute IntAuth_[i/r]n by applying the negotiated prf to the concatenation of IntAuth_[i/r](n-1) (computed for the previous IKE_INTERMEDIATE exchange) and the content of the request (for IntAuth_in) or response (for IntAuth_rn) messages from this exchange. After all IKE_INTERMEDIATE exchanges are over the resulted IntAuth_[i/r]N values (assuming N exchanges took place) are used in the computing the AUTH payload.

For the purpose of calculating the `IntAuth_[i/r]*` values the content of the `IKE_INTERMEDIATE` messages is represented as two chunks of data: mandatory `IntAuth_[i/r]*A` optionally followed by `IntAuth_[i/r]*P`.

The `IntAuth_[i/r]*A` chunk consists of the sequence of octets from the first octet of the IKE Header (not including prepended four octets of zeros, if UDP encapsulation or TCP encapsulation of ESP packets is used) to the last octet of the generic header of the Encrypted payload. The scope of `IntAuth_[i/r]*A` is identical to the scope of Associated Data defined for use of AEAD algorithms in IKEv2 (see Section 5.1 of [RFC5282]), which is stressed by using "A" suffix in its name. Note, that calculation of `IntAuth_[i/r]*A` doesn't depend on whether an AEAD algorithm or a plain cipher is used in IKE SA.

The `IntAuth_[i/r]*P` chunk is present if the Encrypted payload is not empty. It consists of the content of the Encrypted payload that is fully formed, but not yet encrypted. The Initialization Vector, the Padding, the Pad Length and the Integrity Checksum Data fields (see Section 3.14 of [RFC7296]) are not included into the calculation. In other words, the `IntAuth_[i/r]*P` chunk is the inner payloads of the Encrypted payload in plaintext form, which is stressed by using "P" suffix in its name.

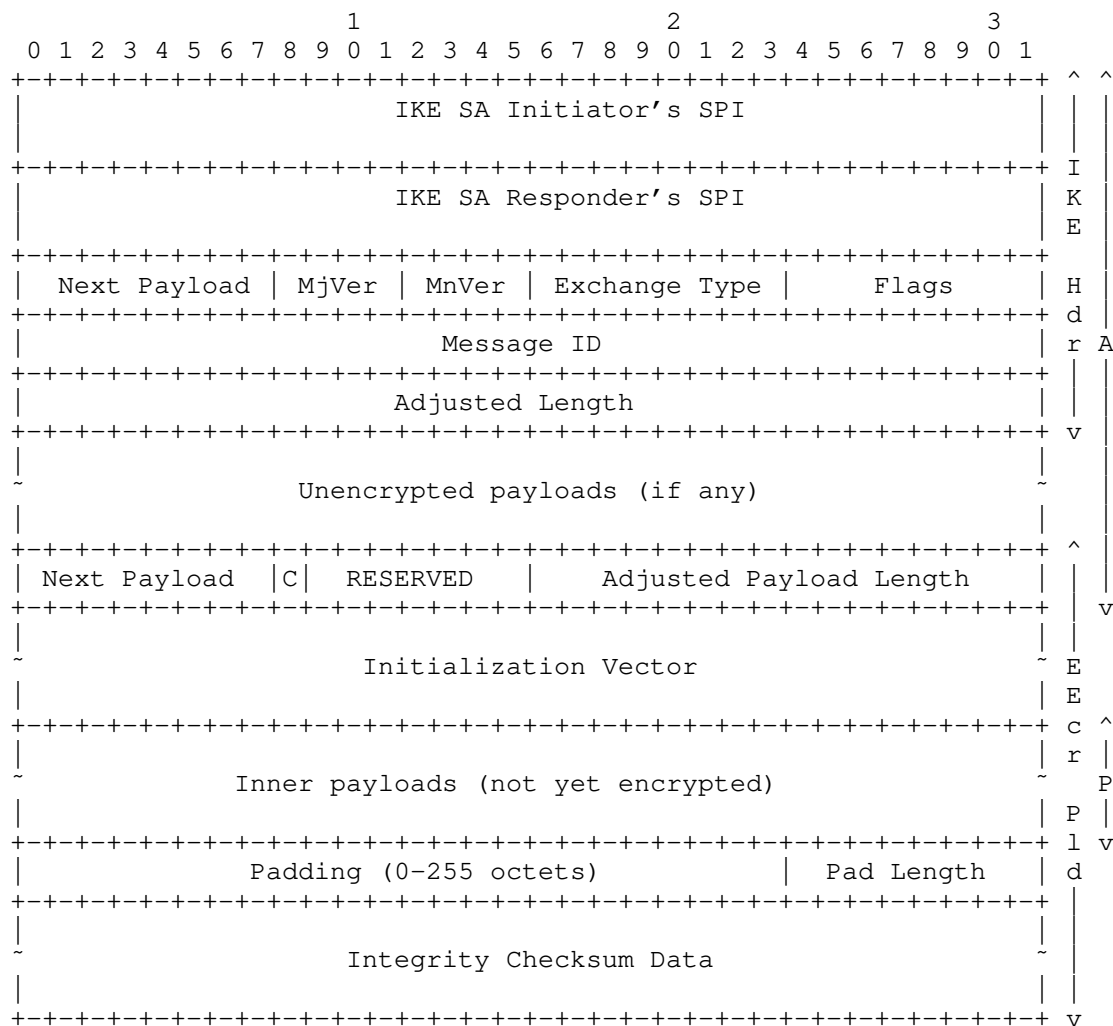


Figure 1: Data to Authenticate in the IKE_INTERMEDIATE Exchange Messages

Figure 1 illustrates the layout of the IntAuth__[i/r]*A (denoted as A) and the IntAuth__[i/r]*P (denoted as P) chunks in case the Encrypted payload is not empty.

For the purpose of prf calculation the Length field in the IKE Header and the Payload Length field in the Encrypted payload header are adjusted so that they don't count the lengths of Initialization Vector, Integrity Checksum Data, Padding and Pad Length fields. In other words, the Length field in the IKE Header (denoted as Adjusted

Length in Figure 1) is set to the sum of the lengths of `IntAuth_[i/r]*A` and `IntAuth_[i/r]*P`, and the Payload Length field in the Encrypted payload header (denoted as Adjusted Payload Length in Figure 1) is set to the length of `IntAuth_[i/r]*P` plus the size of the Encrypted payload header (four octets).

The prf calculations MUST be applied to whole messages only, before possible IKE fragmentation. This ensures that the `IntAuth` will be the same regardless of whether IKE fragmentation takes place or not. If the message was received in fragmented form, it MUST be reconstructed before calculating the prf as if it were received unfragmented. While reconstructing, the RESERVED field in the reconstructed Encrypted payload header MUST be set to the value of the RESERVED field in the Encrypted Fragment payload header from the first fragment (with Fragment Number field set to 1).

Note that it is possible to avoid actual reconstruction of the message by incrementally calculating prf on decrypted (or ready to be encrypted) fragments. However, care must be taken to properly replace the content of the Next Header and the Length fields so that the result of computing the prf is the same as if it were computed on the reconstructed message.

Each calculation of `IntAuth_[i/r]*` uses its own keys `SK_p[i/r]*`, which are the most recently updated `SK_p[i/r]` keys available before the corresponded `IKE_INTERMEDIATE` exchange is started. The first `IKE_INTERMEDIATE` exchange always uses the `SK_p[i/r]` keys that were computed in the `IKE_SA_INIT` as `SK_p[i/r]1`. If the first `IKE_INTERMEDIATE` exchange performs additional key exchange resulting in `SK_p[i/r]` update, then this updated `SK_p[i/r]` are used as `SK_p[i/r]2`, otherwise the original `SK_p[i/r]` are used, and so on. Note that if keys are updated, then for any given `IKE_INTERMEDIATE` exchange the keys `SK_e[i/r]` and `SK_a[i/r]` used for protection of its messages (see Section 3.3.1) and the keys `SK_p[i/r]` for its authentication are always from the same generation.

3.4. Error Handling in the `IKE_INTERMEDIATE` Exchange

Since messages of the `IKE_INTERMEDIATE` exchange are not authenticated until the `IKE_AUTH` exchange successfully completes, possible errors need to be handled with care. There is a trade-off between providing better diagnostics of the problem and risk of becoming part of DoS attack. Section 2.21.1 and 2.21.2 of [RFC7296] describe how errors are handled in initial IKEv2 exchanges; these considerations are also applied to the `IKE_INTERMEDIATE` exchange with a qualification, that not all error notifications may appear in the `IKE_INTERMEDIATE` exchange (for example, errors concerning authentication are generally only applicable to the `IKE_AUTH` exchange).

4. Interaction with other IKEv2 Extensions

The IKE_INTERMEDIATE exchanges MAY be used during the IKEv2 Session Resumption [RFC5723] between the IKE_SESSION_RESUME and the IKE_AUTH exchanges. To be able to use it peers MUST negotiate support for intermediate exchange by including INTERMEDIATE_EXCHANGE_SUPPORTED notifications in the IKE_SESSION_RESUME messages. Note, that a flag whether peers supported the IKE_INTERMEDIATE exchange is not stored in the resumption ticket and is determined each time from the IKE_SESSION_RESUME exchange.

5. Security Considerations

The data that is transferred by means of the IKE_INTERMEDIATE exchanges is not authenticated until the subsequent IKE_AUTH exchange is completed. However, if the data is placed inside the Encrypted payload, then it is protected from passive eavesdroppers. In addition, the peers can be certain that they receive messages from the party they performed the IKE_SA_INIT with if they can successfully verify the Integrity Checksum Data of the Encrypted payload.

The main application for the Intermediate Exchange is to transfer large amounts of data before an IKE SA is set up, without causing IP fragmentation. For that reason it is expected that in most cases IKE fragmentation will be employed in the IKE_INTERMEDIATE exchanges. Section 5 of [RFC7383] contains security considerations for IKE fragmentation.

Since authentication of the peers occurs only in the IKE_AUTH exchange, malicious initiator may use the Intermediate Exchange to mount Denial of Service attack on responder. In this case it starts creating IKE SA, negotiates using the Intermediate Exchanges and transfers a lot of data to the responder that may also require some computationally expensive processing. Then it aborts the SA establishment before the IKE_AUTH exchange. Specifications utilizing the Intermediate Exchange MUST NOT allow unlimited number of these exchanges to take place on initiator's discretion. It is recommended that these specifications are defined in such a way, that the responder would know (possibly via negotiation with the initiator) the exact number of these exchanges that need to take place. In other words: it is preferred that both the initiator and the responder know after the IKE_SA_INIT is completed the exact number of the IKE_INTERMEDIATE exchanges they have to perform; it is allowed that some IKE_INTERMEDIATE exchanges are optional and are performed on the initiator's discretion, but in this case the maximum number of optional exchanges must be hard capped by the corresponding specification. In addition, [RFC8019] provides guidelines for the responder of how to deal with DoS attacks during IKE SA establishment.

Note that if an attacker was able to break the key exchange in real time (e.g. by means of a Quantum Computer), then the security of the IKE_INTERMEDIATE exchange would degrade. In particular, such an attacker would be able both to read data contained in the Encrypted payload and to forge it. The forgery would become evident in the IKE_AUTH exchange (provided the attacker cannot break the employed authentication mechanism), but the ability to inject forged IKE_INTERMEDIATE exchange messages with valid ICV would allow the attacker to mount a Denial-of-Service attack. Moreover, if in this situation the negotiated prf was not secure against second preimage attack with known key, then the attacker could forge the IKE_INTERMEDIATE exchange messages without later being detected in the IKE_AUTH exchange. To do this the attacker would find the same IntAuth__[i/r]* value for the forged message as for original.

6. IANA Considerations

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

43	IKE_INTERMEDIATE
----	------------------

This document also defines a new Notify Message Type in the "IKEv2 Notify Message Types - Status Types" registry:

16438	INTERMEDIATE_EXCHANGE_SUPPORTED
-------	---------------------------------

7. Implementation Status

[Note to RFC Editor: please, remove this section before publishing RFC.]

At the time of writing the -05 version of the draft there were at least three independent interoperable implementations of this specification from the following vendors:

- * ELVIS-PLUS
- * strongSwan
- * libreswan (only one IKE_INTERMEDIATE exchange is supported)

8. Acknowledgements

The idea to use an intermediate exchange between IKE_SA_INIT and IKE_AUTH was first suggested by Tero Kivinen. He also helped with writing an example of using IKE_INTERMEDIATE exchange (shown in Appendix A). Scott Fluhrer and Daniel Van Geest identified a possible problem with authentication of the IKE_INTERMEDIATE exchange and helped to resolve it. Author is grateful to Tobias Brunner who raised good questions concerning authentication of the IKE_INTERMEDIATE exchange and proposed how to make the size of authentication chunk constant regardless of the number of exchanges. Author is also grateful to Paul Wouters and to Benjamin Kaduk who suggested a lot of text improvements for the document.

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

- [RFC5282] Black, D. and D. McGrew, "Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol", RFC 5282, DOI 10.17487/RFC5282, August 2008, <<https://www.rfc-editor.org/info/rfc5282>>.
- [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>>.
- [RFC6928] Chu, J., Dukkkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <<https://www.rfc-editor.org/info/rfc6928>>.
- [RFC8019] Nir, Y. and V. Smyslov, "Protecting Internet Key Exchange Protocol Version 2 (IKEv2) Implementations from Distributed Denial-of-Service Attacks", RFC 8019, DOI 10.17487/RFC8019, November 2016, <<https://www.rfc-editor.org/info/rfc8019>>.
- [RFC8229] Pauly, T., Touati, S., and R. Mantha, "TCP Encapsulation of IKE and IPsec Packets", RFC 8229, DOI 10.17487/RFC8229, August 2017, <<https://www.rfc-editor.org/info/rfc8229>>.

Appendix A. Example of IKE_INTERMEDIATE exchange

This appendix contains an example of the messages using IKE_INTERMEDIATE exchanges. This appendix is purely informative; if it disagrees with the body of this document, the other text is considered correct.

In this example there is one IKE_SA_INIT exchange and two IKE_INTERMEDIATE exchanges, followed by the IKE_AUTH exchange to authenticate all initial exchanges. The xxx in the HDR(XXX,MID=yyy) indicates the exchange type, and yyy tells the message id used for that exchange. The keys used for each SK {} payload are indicated in the parenthesis after the SK. Otherwise, the payload notation is the same as is used in [RFC7296].

Initiator	Responder
-----	-----
HDR(IKE_SA_INIT,MID=0), SAi1, KEi, Ni, N(INTERMEDIATE_EXCHANGE_SUPPORTED) -->	<-- HDR(IKE_SA_INIT,MID=0), SAr1, KEr, Nr, [CERTREQ], N(INTERMEDIATE_EXCHANGE_SUPPORTED)

At this point peers calculate SK_* and store them as SK_*1. SK_e[i/r]1 and SK_a[i/r]1 will be used to protect the first IKE_INTERMEDIATE exchange and SK_p[i/r]1 will be used for its authentication.

Initiator	Responder
-----	-----
HDR(IKE_INTERMEDIATE,MID=1), SK(SK_ei1,SK_ai1) {...} -->	<-- HDR(IKE_INTERMEDIATE,MID=1), SK(SK_er1,SK_ar1) {...}
<Calculate IntAuth_i1 = prf(SK_pi1, ...)>	
	<Calculate IntAuth_r1 = prf(SK_pr1, ...)>

If after completing this IKE_INTERMEDIATE exchange the SK_*1 keys are updated (e.g., as a result of a new key exchange), then the peers store the updated keys as SK_*2, otherwise they use SK_*1 as SK_*2. SK_e[i/r]2 and SK_a[i/r]2 will be used to protect the second IKE_INTERMEDIATE exchange and SK_p[i/r]2 will be used for its authentication.

Initiator	Responder
-----	-----
HDR(IKE_INTERMEDIATE,MID=2), SK(SK_ei2,SK_ai2) {...} -->	<-- HDR(IKE_INTERMEDIATE,MID=2), SK(SK_er2,SK_ar2) {...}
<Calculate IntAuth_i2 = prf(SK_pi2, ...)>	
	<Calculate IntAuth_r2 = prf(SK_pr2, ...)>

If after completing the second IKE_INTERMEDIATE exchange the SK_*2 keys are updated (e.g., as a result of a new key exchange), then the peers store the updated keys as SK_*3, otherwise they use SK_*2 as

SK_*3. SK_e[i/r]3 and SK_a[i/r]3 will be used to protect the IKE_AUTH exchange, SK_p[i/r]3 will be used for authentication, and SK_d3 will be used for derivation of other keys (e.g. for Child SAs).

Initiator -----	Responder -----
HDR(IKE_AUTH,MID=3),	
SK(SK_ei3,SK_ai3)	
{IDi, [CERT,] [CERTREQ,]	
[IDr,] AUTH, SAi2, TSi, TSr} -->	
	<-- HDR(IKE_AUTH,MID=3),
	SK(SK_er3,SK_ar3)
	{IDr, [CERT,] AUTH, SAr2, TSi, TSr}

In this example two IKE_INTERMEDIATE exchanges took place, therefore SK_*3 keys would be used as SK_* keys for further cryptographic operations in the context of the created IKE SA, as defined in [RFC7296].

Author's Address

Valery Smyslov
 ELVIS-PLUS
 PO Box 81
 Moscow (Zelenograd)
 124460
 Russian Federation
 Phone: +7 495 276 0211
 Email: svan@elvis.ru

Internet Engineering Task Force (IETF)
Internet-Draft
Updates: 7296 (if approved)
Intended status: Standards Track
Expires: 29 September 2022

C. Tjhai
M. Tomlinson
Post-Quantum
G. Bartlett
Quantum Secret
S. Fluhrer
Cisco Systems
D. Van Geest
ISARA Corporation
O. Garcia-Morchon
Philips
V. Smyslov
ELVIS-PLUS
28 March 2022

Multiple Key Exchanges in IKEv2
draft-ietf-ipsecme-ikev2-multiple-ke-05

Abstract

This document describes how to extend the Internet Key Exchange Protocol Version 2 (IKEv2) to allow multiple key exchanges to take place while computing a shared secret during a Security Association (SA) setup. The primary application of this feature in IKEv2 is the ability to perform one or more post-quantum key exchanges in conjunction with the classical (Elliptic Curve) Diffie-Hellman key exchange, so that the resulting shared key is resistant against quantum computer attacks. Another possible application is the ability to combine several key exchanges in situations when no single key exchange algorithm is trusted by both initiator and responder.

This document updates RFC7296 by renaming a transform type 4 from "Diffie-Hellman Group (D-H)" to "Key Exchange Method (KE)" and renaming a field in the Key Exchange Payload from "Diffie-Hellman Group Num" to "Key Exchange Method". It also renames an IANA registry for this transform type from "Transform Type 4 - Diffie-Hellman Group Transform IDs" to "Transform Type 4 - Key Exchange Method Transform IDs". These changes generalize key exchange algorithms that can be used in IKEv2.

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 29 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
1.1. Problem Description	3
1.2. Proposed Extension	3
1.3. Changes	5
1.4. Document Organization	7
2. Design Criteria	7
3. Multiple Key Exchanges	9
3.1. Design Overview	9
3.2. Protocol Details	11
3.2.1. IKE_SA_INIT Round: Negotiation	11
3.2.2. IKE_INTERMEDIATE Round: Additional Key Exchanges	15
3.2.3. IKE_AUTH Exchange	15
3.2.4. CREATE_CHILD_SA Exchange	16
3.2.5. Interaction with Childless IKE SA	19
4. IANA Considerations	19
5. Security Considerations	20
6. Acknowledgements	22
7. References	22
7.1. Normative References	22
7.2. Informative References	22

Appendix A. Sample Multiple Key Exchanges	24
A.1. No Additional Key Exchange Used	24
A.2. Additional Key Exchange in the CREATE_CHILD_SA Exchange only	25
A.3. Not Matching Proposal for Additional Key Exchanges	26
Appendix B. Alternative Design	27
Authors' Addresses	31

1. Introduction

1.1. Problem Description

Internet Key Exchange Protocol (IKEv2) as specified in [RFC7296] uses the Diffie-Hellman (DH) or Elliptic Curve Diffie-Hellman (ECDH) algorithm to establish a shared secret between an initiator and a responder. The security of the DH and ECDH algorithms relies on the difficulty to solve a discrete logarithm problem in multiplicative and elliptic curve groups respectively when the order of the group parameter is large enough. While solving such a problem remains difficult with current computing power, it is believed that general purpose quantum computers will be able to solve this problem, implying that the security of IKEv2 is compromised. There are, however, a number of cryptosystems that are conjectured to be resistant against quantum computer attack. This family of cryptosystems is known as post-quantum cryptography (PQC). It is sometimes also referred to as quantum-safe cryptography (QSC) or quantum-resistant cryptography (QRC).

1.2. Proposed Extension

This document describes a method to perform multiple successive key exchanges in IKEv2. It allows integration of QSC in IKEv2, while maintaining backwards compatibility, to derive a set of IKE keys that is resistant to quantum computer attacks. This extension allows the negotiation of one or more QSC algorithm to exchange data, in addition to the existing DH or ECDH key exchange data. We believe that the feature of using more than one post-quantum algorithms is important as many of these algorithms are relatively new and there may be a need to hedge the security risk with multiple key exchange data from several distinct QSC algorithms.

The secrets established from each key exchange are combined in a way such that should the post-quantum secrets not be present, the derived shared secret is equivalent to that of the standard IKEv2; on the other hand, a post-quantum shared secret is obtained if both classical and post-quantum key exchange data are present. This extension also applies to key exchanges in IKE Security Associations (SAs) for Encapsulating Security Payload (ESP) [RFC4303] or Authentication Header (AH) [RFC4302], i.e. Child SAs, in order to provide a stronger guarantee of forward security.

Some post-quantum key exchange payloads may have sizes larger than the standard maximum transmission unit (MTU) size, and therefore there could be issues with fragmentation at the IP layer. IKE does allow transmission over TCP where fragmentation is not an issue [RFC8229]; however, we believe that a UDP-based solution will be required too. IKE does have a mechanism to handle fragmentation within UDP [RFC7383], however that is only applicable to messages exchanged after the IKE_SA_INIT exchange. To use this mechanism, this specification relies on the IKE_INTERMEDIATE exchange as outlined in [I-D.ietf-ipsecme-ikev2-intermediate]. With this mechanism, we do an initial key exchange, using a smaller, possibly non-quantum resistant primitive, such as ECDH. Then, before we do the IKE_AUTH exchange, we perform one or more IKE_INTERMEDIATE exchanges, each of which contains an additional key exchange. As the IKE_INTERMEDIATE exchange is encrypted, the IKE fragmentation protocol [RFC7383] can be used. The IKE SK_* values are updated after each exchange, and so the final IKE SA keys depend on all the key exchanges, hence they are secure if any of the key exchanges are secure.

Note that readers should consider the approach defined in this document as providing a long term solution in upgrading the IKEv2 protocol to support post-quantum algorithms. A short term solution to make IKEv2 key exchange quantum secure is to use post-quantum pre-shared keys as discussed in [RFC8784].

Note also, that the proposed approach of performing multiple successive key exchanges in such a way that resulting session keys depend on all of them is not limited to achieving quantum resistance only. It can also be used when all the performed key exchanges are classical (EC)DH ones, where for some reasons (e.g. policy requirements) it is essential to perform multiple of them.

This draft does not attempt to address key exchanges with KE payloads longer than 64k; the current IKE payload format does not allow that as a possibility. At the current time, it appears likely that there are a number of key exchanges available that would not require such a requirement. However, if such a requirement is needed, [I-D.tjhai-ikev2-beyond-64k-limit] discusses approaches that should be taken to exchange huge payloads.

1.3. Changes

RFC EDITOR PLEASE DELETE THIS SECTION.

Changes in this draft in each version iterations.

draft-ietf-ipsecme-ikev2-multiple-ke-04

- * Introduction and initial sections are reorganized.
- * More clarifications for error handling added.
- * ASCII arts displaying SA payload are added.
- * Clarification for handling multiple round trips key exchange methods added.
- * DoS concerns added into Security Considerations section.
- * Explicitly allow scenario when additional key exchanges are performed only after peers are authenticated.

draft-ietf-ipsecme-ikev2-multiple-ke-03

- * More clarifications added.
- * Figure illustrating initial exchange added.
- * Minor editorial changes.

draft-ietf-ipsecme-ikev2-multiple-ke-02

- * Added a reference on the handling of KE payloads larger than 64KB.

draft-ietf-ipsecme-ikev2-multiple-ke-01

- * References are updated.

draft-ietf-ipsecme-ikev2-multiple-ke-00

- * Draft name changed as result of WG adoption and generalization of the approach.
- * New exchange IKE_FOLLOWUP_KE is defined for additional key exchanges performed after CREATE_CHILD_SA.
- * Nonces are removed from all additional key exchanges.
- * Clarification that IKE_INTERMEDIATE must be negotiated is added.

draft-tjhai-ipsecme-hybrid-qske-ikev2-04

- * Clarification about key derivation in case of multiple key exchanges in CREATE_CHILD_SA is added.
- * Resolving rekey collisions in case of multiple key exchanges is clarified.

draft-tjhai-ipsecme-hybrid-qske-ikev2-03

- * Using multiple key exchanges CREATE_CHILD_SA is defined.

draft-tjhai-ipsecme-hybrid-qske-ikev2-02

- * Use new transform types to negotiate additional key exchanges, rather than using the KE payloads of IKE SA.

draft-tjhai-ipsecme-hybrid-qske-ikev2-01

- * Use IKE_INTERMEDIATE to perform multiple key exchanges in succession.
- * Handle fragmentation by keeping the first key exchange (a standard IKE_SA_INIT with a few extra notifies) small, and encrypting the rest of the key exchanges.
- * Simplify the negotiation of the 'extra' key exchanges.

draft-tjhai-ipsecme-hybrid-qske-ikev2-00

- * We added a feature to allow more than one post-quantum key exchange algorithms to be negotiated and used to exchange a post-quantum shared secret.
- * Instead of relying on TCP encapsulation to deal with IP level fragmentation, we introduced a new key exchange payload that can be sent as multiple fragments within IKE_SA_INIT message.

1.4. Document Organization

The remainder of this document is organized as follows. Section 2 summarizes design criteria. Section 3 describes how multiple key exchanges are performed between two IKE peers and how keying materials are derived for both SAs and Child SAs. A summary of alternative approaches that have been considered, but later discarded, are described in Appendix B. Section 4 discusses IANA considerations for the namespaces introduced in this document, and lastly Section 5 discusses security considerations.

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.

2. Design Criteria

The design of the proposed extension is driven by the following criteria:

- 1) Need for post-quantum cryptography in IPsec. Quantum computers might become feasible in the near future. If current Internet communications are monitored and recorded today (D), the communications could be decrypted as soon as a quantum- computer is available (e.g., year Q) if key negotiation only relies on non post-quantum primitives. This is a high threat for any information that must remain confidential for a long period of time $T > Q-D$. The need is obvious if we assume that Q is 2040, D is 2020, and T is 30 years. Such a value of T is typical in classified or healthcare data.
- 2) Hybrid. Currently, there does not exist a post-quantum key exchange that is trusted at the level that ECDH is trusted against conventional (non-quantum) adversaries. A hybrid post-quantum algorithm to be introduced next to well-established primitives, since the overall security is at least as strong as each individual primitive.
- 3) Focus on quantum-resistant confidentiality. A passive attacker

can eavesdrop on IPsec communication today and decrypt it once a quantum computer is available in the future. This is a very serious attack for which we do not have a solution. An attacker can only perform active attacks such as impersonation of the communicating peers once a quantum computer is available, sometime in the future. Thus, our design focuses on quantum-resistant confidentiality due to the urgency of this problem. This document does not address quantum-resistant authentication since it is less urgent at this stage.

- 4) Limit amount of exchanged data. The protocol design should be such that the amount of exchanged data, such as public-keys, is kept as small as possible even if initiator and responder need to agree on a hybrid group or multiple public-keys need to be exchanged.
- 5) Future proof. Any cryptographic algorithm could be potentially broken in the future by currently unknown or impractical attacks: quantum computers are merely the most concrete example of this. The design does not categorize algorithms as "post-quantum" or "non post-quantum" nor does it create assumptions about the properties of the algorithms, meaning that if algorithms with different properties become necessary in the future, this extension can be used unchanged to facilitate migration to those algorithms.
- 6) Limited amount of changes. A key goal is to limit the number of changes required when enabling a post-quantum handshake. This ensures easier and quicker adoption in existing implementations.
- 7) Localized changes. Another key requirement is that changes to the protocol are limited in scope, in particular, limiting changes in the exchanged messages and in the state machine, so that they can be easily implemented.
- 8) Deterministic operation. This requirement means that the hybrid post-quantum exchange, and thus, the computed keys, will be based on algorithms that both client and server wish to support.
- 9) Fragmentation support. Some PQC algorithms could be relatively bulky and they might require fragmentation. Thus, a design goal is the adaptation and adoption of an existing fragmentation method or the design of a new method that allows for the fragmentation of the key shares.
- 10) Backwards compatibility and interoperability. This is a fundamental requirement to ensure that hybrid post-quantum IKEv2 and non-post-quantum IKEv2 implementations are interoperable.

- 11) Federal Information Processing Standards (FIPS) compliance. IPsec is widely used in Federal Information Systems and FIPS certification is an important requirement. However, algorithms that are believed to be post-quantum are not FIPS compliant yet. Still, the goal is that the overall hybrid post-quantum IKEv2 design can be FIPS compliant.
- 12) Ability to use this method with multiple classical (EC)DH key exchanges. In some situations peers have no single mutually trusted key exchange algorithm (e.g., due to local policy restrictions). The ability to combine two (or more) key exchange methods in such a way that the resulting shared key depends on all of them allows peers to communicate in this situation.

3. Multiple Key Exchanges

3.1. Design Overview

Most post-quantum key agreement algorithms are relatively new, and thus are not fully trusted. There are also many proposed algorithms, with different trade-offs and relying on different hard problems. The concern is that some of these hard problems may turn out to be easier to solve than anticipated and thus the key agreement algorithm may not be as secure as expected. A hybrid solution, when multiple key exchanges are performed and the calculated shared key depends on all of them, allows us to deal with this uncertainty by combining a classical key exchange with a post-quantum one, as well as leaving open the possibility of multiple post-quantum key exchanges.

In order to be able to use IKE fragmentation [RFC7383] for those key exchanges that may have long public keys, the proposed framework utilizes the IKE_INTERMEDIATE exchange defined in [I-D.ietf-ipsecme-ikev2-intermediate]. The initial IKE_INIT messages do not have any inherent fragmentation support within IKE; however that can include a relatively short KE payload. The additional key exchanges are performed using IKE_INTERMEDIATE messages; because these messages are encrypted, the standard IKE fragmentation mechanism is available.

In order to minimize communication overhead, only the key shares that are agreed to be used are actually exchanged. To negotiate additional key exchanges seven new Transform Types are defined. These transforms share allowed Transform IDs with Transform Type 4.

We assume that new Transform Type 4 identifiers will be assigned later to the various post-quantum key exchanges. We specifically do not make a distinction between classical (DH and ECDH) and post-

quantum key exchanges, nor post-quantum algorithms which are true key exchanges versus post-quantum algorithms that act as key transport mechanisms; all are treated equivalently by the protocol. To be more specific, this document renames Transform Type 4 from "Diffie-Hellman Group (D-H)" to "Key Exchange Method (KE)" and renames a field in the Key Exchange Payload from "Diffie-Hellman Group Num" to "Key Exchange Method". The corresponding IANA registry is also renamed from "Diffie-Hellman Group Transform IDs" to "Key Exchange Method Transform IDs".

The fact, that newly defined transforms share the same registry for possible Transform IDs with Transform Type 4, allows additional key exchanges to be of any type - either post-quantum or classical (EC)DH one. This approach allows any combination of defined key exchange methods to take place. This also allows performing a single post-quantum key exchange in the IKE_SA_INIT without additional key exchanges, provided that IP fragmentation is not an issue and that hybrid key exchange is not needed.

The SA payload in the IKE_SA_INIT message includes one or more newly defined transforms which represent the extra key exchange policy required by the initiator. The responder follows the usual IKEv2 negotiation rules: it selects a single transform of each type, and returns all of them in the IKE_SA_INIT response message.

Then, provided that additional key exchanges are negotiated, the initiator and the responder perform one or more IKE_INTERMEDIATE exchanges. Then the IKE_AUTH exchange authenticates peers and completes IKE SA establishment.

Initiator	Responder

<-- IKE_SA_INIT (additional key exchanges negotiation) -->	
<-- {IKE_INTERMEDIATE (additional key exchange)} -->	
...	
<-- {IKE_INTERMEDIATE (additional key exchange)} -->	
<-- {IKE_AUTH} -->	

Note, that this document assumes, that each key exchange method requires one round trip and consumes exactly one `IKE_INTERMEDIATE` exchange. This assumption is valid for all classic key exchange methods defined so far and for all post-quantum methods currently known. For hypothetical future key exchange methods requiring multiple round trips to complete, a separate document should define how such methods are splitted into several `IKE_INTERMEDIATE` exchanges.

3.2. Protocol Details

In the simplest case, the initiator is happy with a single key exchange (and has no interest in supporting multiple), and it is not concerned with possible fragmentation of the `IKE_SA_INIT` messages (either because the key exchange it selects is small enough not to fragment, or the initiator is confident that fragmentation will be handled either by IP fragmentation, or transport via TCP).

In this case, the initiator performs the `IKE_SA_INIT` as usual, inserting a preferred key exchange (which is possibly a post-quantum algorithm) as the listed Transform Type 4, and including the initiator KE payload. If the responder accepts the policy, it responds with an `IKE_SA_INIT` response, and IKE continues as usual.

If the initiator desires to negotiate multiple key exchanges, then the initiator uses the protocol listed below.

3.2.1. `IKE_SA_INIT` Round: Negotiation

Multiple key exchanges are negotiated using the standard IKEv2 mechanism, via SA payload. For this purpose seven new transform types, namely Additional Key Exchange 1 (<TBA by IANA>), Additional Key Exchange 2 (<TBA by IANA>), Additional Key Exchange 3 (<TBA by IANA>), Additional Key Exchange 4 (<TBA by IANA>), Additional Key Exchange 5 (<TBA by IANA>), Additional Key Exchange 6 (<TBA by IANA>) and Additional Key Exchange 7 (<TBA by IANA>) are defined. They are collectively called Additional Key Exchange transforms in this document and have slightly different semantics than existing IKEv2 transform types. They are interpreted as an indication of additional key exchanges methods that peers agreed to perform in a series of `IKE_INTERMEDIATE` exchanges following the `IKE_SA_INIT` exchange. The allowed transform IDs for these transform types are the same as IDs for the Transform Type 4, so they all share a single IANA registry for transform IDs.

Key exchange method negotiated via Transform Type 4 always takes place in the `IKE_SA_INIT` exchange, as defined in [RFC7296]. Additional key exchanges negotiated via newly defined transforms MUST

take place in a series of IKE_INTERMEDIATE exchanges following the IKE_SA_INIT exchange, performed in an order of the values of their transform types, so that key exchange negotiated using Transform Type n always precedes that of Transform Type $n + 1$. Each additional key exchange method MUST be fully completed before the next one is started.

Note that with this semantics, Additional Key Exchanges transforms are not associated with any particular type of key exchange and do not have any specific per transform type transform IDs IANA registry. Instead they all share a single registry for transform IDs - "Key Exchange Method Transform IDs", as well as Transform Type 4. All new key exchange algorithms (both classical or post-quantum) should be added to this registry. This approach gives peers flexibility in defining the ways they want to combine different key exchange methods.

When forming a proposal the initiator adds transforms for the IKE_SA_INIT exchange using Transform Type 4. In most cases they will contain classical key exchange methods (DH or ECDH), however it is not a requirement. Additional key exchange methods are proposed using Additional Key Exchanges transform types. All these transform types are optional, the initiator is free to select any of them for proposing additional key exchange methods. Consequently, if none of Additional Key Exchange transforms are included in the proposal, then this proposal indicates performing standard IKEv2, as defined in [RFC7296]. If the initiator includes any Additional Key Exchanges transform in the proposal, the responder MUST select one of the algorithms proposed using this type. A transform ID NONE MAY be added to those transform types which contain key exchange methods that the initiator believes are optional according to its local policy.

The responder performs negotiation using standard IKEv2 procedure described in Section 3.3 of [RFC7296]. However, for the Additional Key Exchange types the responder's choice MUST NOT contain equal algorithms, except for transform ID of NONE. An algorithm is represented as a transform, in some cases the transform could include a set of associated attributes that define details of the algorithm. In this case two transforms can be the same, but the attributes must be different. Additionally, the order of the attributes does not affect the equality of the algorithm, so two transforms (ID=alg1,ATTR1=attr1,ATTR2=attr2) and (ID=alg1,ATTR2=attr2,ATTR1=attr1) define the same algorithm.

If the responder selected NONE for some Additional Key Exchange types (provided they were proposed by the initiator), then the corresponding IKE_INTERMEDIATE exchanges should not take place. The

IKE_INTERMEDIATE exchanges MUST only be performed for Additional Key Exchange types containing non-NONE responders choices. It means that if the initiator includes NONE in all Additional Key Exchange transforms and the responder selects this value for all of them, then no IKE_INTERMEDIATE exchanges will take place between the peers. perform additional key exchanges will take place (note that they still may take place for other purposes).

Below is an example of the SA payload in the initiator's IKE_SA_INIT request message. Here we use an abbreviation AKE1, AKE 2 etc. to denote Additional Key Exchange 1, Additional Key Exchange 2 etc. transforms, that this document defines, and an abbreviation KE for the Key Exchange transform, that this document renames from the Diffie-Hellman Group transform. We also use not yet defined Transform IDs PQ_KEM_1, PQ_KEM_2 and PQ_KEM_3 to denote some of popular post-quantum key exchange methods.

SA Payload

```

+--- Proposal #1 ( Proto ID = IKE(1), SPI size = 8,
                    9 transforms,          SPI = 0x35a1d6f22564f89d )
    |
    +--- Transform ENCR ( ID = ENCR_AES_GCM_16 )
    |     +--- Attribute ( Key Length = 256 )
    |
    +--- Transform KE ( ID = 4096-bit MODP Group )
    |
    +--- Transform PRF ( ID = PRF_HMAC_SHA2_256 )
    |
    +--- Transform AKE2 ( ID = PQ_KEM_1 )
    |
    +--- Transform AKE2 ( ID = PQ_KEM_2 )
    |
    +--- Transform AKE3 ( ID = PQ_KEM_1 )
    |
    +--- Transform AKE3 ( ID = PQ_KEM_2 )
    |
    +--- Transform AKE5 ( ID = PQ_KEM_3 )
    |
    +--- Transform AKE5 ( ID = NONE )

```

In this example the initiator proposes to perform initial key exchange using 4096-bit MODP group following by two mandatory additional key exchanges using PQ_KEM_1 and PQ_KEM_2 methods in any order, following by additional key exchange using PQ_KEM_3 method that may be omitted.

The responder might return the following SA payload, indicating that it agrees to perform two additional key exchanges PQ_KEM_2 followed by PQ_KEM_1 and doesn't want to perform PQ_KEM_3 additionally.

SA Payload

```

+--- Proposal #1 ( Proto ID = IKE(1), SPI size = 8,
                    6 transforms,          SPI = 0x8df52b331a196e7b )
    |
    +-- Transform ENCR ( ID = ENCR_AES_GCM_16 )
    |   +-- Attribute ( Key Length = 256 )
    |
    +-- Transform KE ( ID = 4096-bit MODP Group )
    |
    +-- Transform PRF ( ID = PRF_HMAC_SHA2_256 )
    |
    +-- Transform AKE2 ( ID = PQ_KEM_2 )
    |
    +-- Transform AKE3 ( ID = PQ_KEM_1 )
    |
    +-- Transform AKE5 ( ID = NONE )

```

If the initiator includes any Additional Key Exchanges transform types into SA payload in the IKE_SA_INIT exchange request message, it MUST also negotiate using IKE_INTERMEDIATE exchange as described in [I-D.ietf-ipsecme-ikev2-intermediate], by including INTERMEDIATE_EXCHANGE_SUPPORTED notification in the same message. If the responder agrees to use additional key exchanges while establishing initial IKE SA, it MUST also return this notification in the IKE_SA_INIT response message, thus confirming that IKE_INTERMEDIATE exchange is supported and will be used for transferring additional key exchange data. If the IKE_INTERMEDIATE exchange is not negotiated, then the peers MUST treat any Additional Key Exchange transforms in the IKE_SA_INIT exchange messages as unknown transform types and skip the proposals they appear in. If no other proposals are present in the SA payload, the peers will proceed as when no proposal is chosen (i.e. the responder will send NO_PROPOSAL_CHOSEN notification).

Initiator	Responder

HDR, SAi1(.. AKE*...), KEi1, Ni, N(INTERMEDIATE_EXCHANGE_SUPPORTED)	--->
	HDR, SAR1(.. AKE*...), KER1, Nr, [CERTREQ],
	<--- N(INTERMEDIATE_EXCHANGE_SUPPORTED)

3.2.2. IKE_INTERMEDIATE Round: Additional Key Exchanges

For each additional key exchange agreed to in the IKE_SA_INIT exchange, the initiator and the responder perform IKE_INTERMEDIATE exchange, as described in [I-D.ietf-ipsecme-ikev2-intermediate].

Initiator	Responder
HDR, SK {KEi(n)} -->	<-- HDR, SK {KEr(n)}

The initiator sends key exchange data in the KEi(n) payload. This packet is protected with the current SK_ei/SK_ai keys.

On receiving this, the responder sends back key exchange payload KEr(n); again, this packet is protected with the current SK_er/SK_ar keys.

The former "Diffie-Hellman Group Num" (now called "Key Exchange Method") field in the KEi(n) and KEr(n) payloads MUST match the n-th negotiated additional key exchange.

Once this exchange is done, both sides compute an updated keying material:

$$\text{SKEYSEED}(n) = \text{prf}(\text{SK}_d(n-1), \text{SK}(n) \mid \text{Ni} \mid \text{Nr})$$

where SK(n) is the resulting shared secret of this key exchange, Ni and Nr are nonces from the IKE_SA_INIT exchange and SK_d(n-1) is the last generated SK_d, (derived from the previous IKE_INTERMEDIATE exchange, or the IKE_SA_INIT if there have not already been any IKE_INTERMEDIATE exchanges). Then, SK_d, SK_ai, SK_ar, SK_ei, SK_er, SK_pi, SK_pr are updated as:

$$\{\text{SK}_d(n) \mid \text{SK}_{ai}(n) \mid \text{SK}_{ar}(n) \mid \text{SK}_{ei}(n) \mid \text{SK}_{er}(n) \mid \text{SK}_{pi}(n) \mid \text{SK}_{pr}(n)\} = \text{prf}^+(\text{SKEYSEED}(n), \text{Ni} \mid \text{Nr} \mid \text{SPIi} \mid \text{SPIr})$$

Both the initiator and the responder use these updated key values in the next exchange (IKE_INTERMEDIATE or IKE_AUTH).

3.2.3. IKE_AUTH Exchange

After all IKE_INTERMEDIATE exchanges have completed, the initiator and the responder perform an IKE_AUTH exchange. This exchange is the standard IKE exchange, except that the initiator and responder signed octets are modified as described in [I-D.ietf-ipsecme-ikev2-intermediate].

3.2.4. CREATE_CHILD_SA Exchange

The CREATE_CHILD_SA exchange is used in IKEv2 for the purposes of creating additional Child SAs, rekeying them and rekeying IKE SA itself. When creating or rekeying Child SAs, the peers may optionally perform a Diffie-Hellman key exchange to add a fresh entropy into the session keys. In case of IKE SA rekey, the key exchange is mandatory. Peers supporting this specification may want to use multiple key exchanges in these situations.

Using multiple key exchanges with CREATE_CHILD_SA exchange is negotiated similarly as in initial exchange, see Section 3.2.1. If the initiator includes any Additional Key Exchanges transform in the SA payload (along with Transform Type 4) and the responder agrees to perform additional key exchanges, then the additional key exchanges are performed in a series of new IKE_FOLLOWUP_KE exchanges that follows the CREATE_CHILD_SA exchange. The IKE_FOLLOWUP_KE exchange is introduced as a dedicated exchange for transferring data of additional key exchanges following the key exchange performed in the CREATE_CHILD_SA. Its Exchange Type is <TBA by IANA>.

Key exchange negotiated via Transform Type 4 always takes place in the CREATE_CHILD_SA exchange, as per IKEv2 specification. Additional key exchanges are performed in an order of the values of their transform types, so that key exchange negotiated using Transform Type n always precedes key exchange negotiated using Transform Type $n + 1$. Each additional key exchange method MUST be fully completed before the next one is started. Note, that this document assumes, that each key exchange method consumes exactly one IKE_FOLLOWUP_KE exchange. For the methods requiring multiple round trips, a separate document should define how such methods are splitted into several IKE_FOLLOWUP_KE exchanges.

Since after IKE SA is created the window size may be greater than one and multiple concurrent exchanges may be in progress, it is essential to link the IKE_FOLLOWUP_KE exchanges together and with the corresponding CREATE_CHILD_SA exchange. A new status type notification ADDITIONAL_KEY_EXCHANGE is used for this purpose. Its Notify Message Type is <TBA by IANA>, Protocol ID and SPI Size are both set to 0. The data associated with this notification is a blob meaningful only to the responder, so that the responder can correctly link successive exchanges. For the initiator the content of this notification is an opaque blob.

The responder MUST include this notification in a CREATE_CHILD_SA or IKE_FOLLOWUP_KEY_EXCHANGE response message in case the next IKE_FOLLOWUP_KEY_EXCHANGE exchange is expected, filling it with some data that would allow linking current exchange to the next one. The initiator MUST send back this notification intact in the request message of the next IKE_FOLLOWUP_KEY_EXCHANGE exchange.

Below is an example of CREATE_CHILD_SA exchange followed by three additional key exchanges.

Initiator	Responder

HDR(CREATE_CHILD_SA), SK {SA, Ni, KEi} -->	<-- HDR(CREATE_CHILD_SA), SK {SA, Nr, KEr, N(ADDITIONAL_KEY_EXCHANGE) (link1)}
HDR(IKE_FOLLOWUP_KEY_EXCHANGE), SK {KEi(1), N(ADDITIONAL_KEY_EXCHANGE) (link1)} -->	<-- HDR(IKE_FOLLOWUP_KEY_EXCHANGE), SK {KEr(1), N(ADDITIONAL_KEY_EXCHANGE) (link2)}
HDR(IKE_FOLLOWUP_KEY_EXCHANGE), SK {KEi(2), N(ADDITIONAL_KEY_EXCHANGE) (link2)} -->	<-- HDR(IKE_FOLLOWUP_KEY_EXCHANGE), SK {KEr(2), N(ADDITIONAL_KEY_EXCHANGE) (link3)}
HDR(IKE_FOLLOWUP_KEY_EXCHANGE), SK {KEi(3), N(ADDITIONAL_KEY_EXCHANGE) (link3)} -->	<-- HDR(IKE_FOLLOWUP_KEY_EXCHANGE), SK {KEr(3)}

The former "Diffie-Hellman Group Num" (now called "Key Exchange Method") field in the KEi(n) and KEr(n) payloads MUST match the n-th negotiated additional key exchange.

It is possible that due to some unexpected events (e.g. reboot) the initiator may lose its state and forget that it is in the process of performing additional key exchanges and thus never start the remaining IKE_FOLLOWUP_KEY_EXCHANGE exchanges. The responder MUST handle this situation gracefully and delete the associated state if it does not receive the next expected IKE_FOLLOWUP_KEY_EXCHANGE request after some reasonable period of time.

If responder receives IKE_FOLLOWUP_KEY_EXCHANGE request containing ADDITIONAL_KEY_EXCHANGE notification and the content of this notify does not correspond to any active key exchange state the responder has, it MUST send back a new error type notification STATE_NOT_FOUND. This is a non-fatal error notification, its Notify Message Type is <TBA by IANA>, Protocol ID and SPI Size are both set to 0 and the

data is empty. If the initiator receives this notification in response to IKE_FOLLOWUP_KE exchange performing additional key exchange, it MUST cancel this exchange and MUST treat the whole series of exchanges started from the CREATE_CHILD_SA exchange as failed. In most cases, the receipt of this notification is caused by premature deletion of the corresponding state on the responder (the time period between IKE_FOLLOWUP_KE exchanges appeared too long from the responder's point of view, e.g. due to a temporary network failure). After receiving this notification the initiator MAY start a new CREATE_CHILD_SA exchange (eventually followed by the IKE_FOLLOWUP_KE exchanges) to retry the failed attempt. If the initiator continues to receive STATE_NOT_FOUND notifications after several retries, it MUST treat this situation as a fatal error and delete IKE SA by sending a DELETE payload.

When rekeying IKE SA or Child SA, it is possible that the peers start doing this at the same time, which is called simultaneous rekeying. Sections 2.8.1 and 2.8.2 of [RFC7296] describe how IKEv2 handles this situation. In a nutshell IKEv2 follows the rule that if in case of simultaneous rekeying two identical new IKE SAs (or two pairs of Child SAs) are created, then one of them should be deleted. Which one is to be deleted is determined by comparing the values of four nonces, that were used in the colliding CREATE_CHILD_SA exchanges - the IKE SA (or pair of Child SAs) that was created by the exchange in which the smallest nonce was used should be deleted by the initiator of this exchange.

With multiple key exchanges the SAs are not yet created when the CREATE_CHILD_SA is completed, they would be created only after the series of IKE_FOLLOWUP_KE exchanges is finished. For this reason if additional key exchanges were negotiated in the CREATE_CHILD_SA initiated by the losing side, there is nothing to delete and this side just stops the rekeying process - this side MUST NOT initiate IKE_FOLLOWUP_KE exchange with next key exchange.

In most cases, rekey collisions are resolved in the CREATE_CHILD_SA exchange. However, a situation may occur when due to packet loss, one of the peers receives the CREATE_CHILD_SA message requesting rekey of SA that is already being rekeyed by this peer (i.e. the CREATE_CHILD_SA exchange initiated by this peer has been already completed and the series of IKE_FOLLOWUP_KE exchanges is in progress). In this case, TEMPORARY_FAILURE notification MUST be sent in response to such a request.

If multiple key exchanges were negotiated in the CREATE_CHILD_SA exchange, then the resulting keys are computed as follows. In case of IKE SA rekey:

$$\text{SKEYSEED} = \text{prf}(\text{SK_d}, \text{SK}(0) \mid \text{Ni} \mid \text{Nr} \mid \text{SK}(1) \mid \dots \text{SK}(n))$$

In case of Child SA creation or rekey:

$$\text{KEYMAT} = \text{prf+}(\text{SK_d}, \text{SK}(0) \mid \text{Ni} \mid \text{Nr} \mid \text{SK}(1) \mid \dots \text{SK}(n))$$

In both cases SK_d is from existing IKE SA; SK(0), Ni, Nr are the shared key and nonces from the CREATE_CHILD_SA respectively; SK(1)...SK(n) are the shared keys from additional key exchanges.

3.2.5. Interaction with Childless IKE SA

It is also possible to establish a fully quantum-resistant IKE SAs from additional key exchanges without using IKE_INTERMEDIATE exchanges. In this case, the IKE SA created from IKE_SA_INIT exchange can be immediately rekeyed with CREATE_CHILD_SA using additional key exchanges and IKE_FOLLOWUP_KE message to carry the key exchange payload. If only classical key exchange method is used in the IKE_SA_INIT message, the very first Child SA created in IKE_AUTH will not be quantum resistant. Consequently, if the peers' local policy requires that all Child SAs should be fully-protected, then the peers can avoid creating the very first Child SA by adopting [RFC6023]. In this case, the peers exchange CHILDLess_IKEV2_SUPPORTED notification in the IKE_SA_INIT exchange and a fully-protected Child SA can be created with CREATE_CHILD_SA using additional key exchanges.

Note that if the initial IKE SA is used to transfer sensitive information, then this information will not be protected using the additional (e.g. quantum safe) key exchanges, so this scenario may be inappropriate. One such example is in G-IKEv2 protocol [I-D.ietf-ipsecme-g-ikev2] where cryptographic materials are exchanged in IKE_SA_INIT messages between group member and the group controller.

4. IANA Considerations

This document adds new exchange type into the "IKEv2 Exchange Types" registry:

<TBA> IKE_FOLLOWUP_KE

This document renames Transform Type 4 defined in "Transform Type Values" registry from "Diffie-Hellman Group (D-H)" to "Key Exchange Method (KE)".

This document renames IKEv2 registry "Transform Type 4 - Diffie-Hellman Group Transform IDs" to "Transform Type 4 - Key Exchange Method Transform IDs".

This document adds the following Transform Types to the "Transform Type Values" registry:

Type	Description	Used In
<TBA>	Additional Key Exchange 1	(optional in IKE, AH, ESP)
<TBA>	Additional Key Exchange 2	(optional in IKE, AH, ESP)
<TBA>	Additional Key Exchange 3	(optional in IKE, AH, ESP)
<TBA>	Additional Key Exchange 4	(optional in IKE, AH, ESP)
<TBA>	Additional Key Exchange 5	(optional in IKE, AH, ESP)
<TBA>	Additional Key Exchange 6	(optional in IKE, AH, ESP)
<TBA>	Additional Key Exchange 7	(optional in IKE, AH, ESP)

This document defines a new Notify Message Type in the "Notify Message Types - Status Types" registry:

<TBA> ADDITIONAL_KEY_EXCHANGE

and a new Notify Message Type in the "Notify Message Types - Error Types" registry:

<TBA> STATE_NOT_FOUND

5. Security Considerations

The key length of the Encryption Algorithm (Transform Type 1), the Pseudorandom Function (Transform Type 2) and the Integrity Algorithm (Transform Type 3), all have to be of sufficient length to prevent attacks using Grover's algorithm [GROVER]. In order to use the extension proposed in this document, the key lengths of these transforms MUST be at least 256 bits long in order to provide sufficient resistance to quantum attacks. Accordingly the post-quantum security level achieved is at least 128 bits.

SKEYSEED is calculated from shared SK(x) using an algorithm defined in Transform Type 2. While a quantum attacker may learn the value of SK(x), if this value is obtained by means of a classical key exchange, other SK(x) values generated by means of a quantum-resistant algorithm ensure that the final SKEYSEED is not compromised. This assumes that the algorithm defined in the Transform Type 2 is post-quantum.

The main focus of this document is to prevent a passive attacker performing a "harvest and decrypt" attack. In other words, an attacker that records messages exchanged today and proceeds to decrypt them once he owns a quantum computer. This attack is prevented due to the hybrid nature of the key exchange. Other attacks involving an active attacker using a quantum-computer are not completely solved by this document. This is for two reasons.

The first reason is because the authentication step remains classical. In particular, the authenticity of the SAs established under IKEv2 is protected using a pre-shared key, RSA, DSA, or ECDSA algorithms. Whilst the pre-shared key option, provided the key is long enough, is post-quantum, the other algorithms are not. Moreover, in implementations where scalability is a requirement, the pre-shared key method may not be suitable. Quantum-safe authenticity may be provided by using a quantum-safe digital signature and several quantum-safe digital signature methods are being explored by IETF. For example, if the implementation is able to reliably track state, the hash based method, XMSS has the status of an RFC, see [RFC8391]. Currently, quantum-safe authentication methods are not specified in this document, but are planned to be incorporated in due course.

It should be noted that the purpose of post-quantum algorithms is to provide resistance to attacks mounted in the future. The current threat is that encrypted sessions are subject to eavesdropping and archived with decryption by quantum computers taking place at some point in the future. Until quantum computers become available there is no point in attacking the authenticity of a connection because there are no possibilities for exploitation. These only occur at the time of the connection, for example by mounting a man-in-the-middle (MitM) attack. Consequently there is not such a pressing need for quantum-safe authenticity.

Performing multiple key exchanges while establishing IKEv2 SA increases the responder's susceptibility to DoS attacks, because of an increased amount of resources needed to spend before the initiator is authenticated. This is especially true for post-quantum key exchange methods, where many of them are more memory and/or CPU intensive than the classical counterparts.

Responders may consider recommendations from [RFC8019] to deal with increased DoS attack susceptibility. It is also possible that the responder only agrees to create initial IKE SA without performing additional key exchanges, provided the initiator includes such an option in its proposals. Then peers immediately rekey initial IKE SA with the CREATE_CHILD_SA exchange and additional key exchanges performed via the IKE_FOLLOWUP_KEY exchanges. In this case at the point when resource-intensive operations are required, peers have

already authenticated each other. However, in the context of hybrid post-quantum key exchange this scenario would leave initial IKE SA (and initial Child SA if it is created) unprotected against quantum computers. Nevertheless the rekeyed IKE SA (and Child SAs that will be created over it) will have full protection. This is similar to the scenario described in [RFC8784]. Depending on peers' policy, this scenario may or may not be appropriate.

6. Acknowledgements

The authors would like to thank Frederic Detienne and Olivier Pelerin for their comments and suggestions, including the idea to negotiate the post-quantum algorithms using the existing KE payload. The authors are also grateful to Tobias Heider and Tobias Guggemos for valuable comments. Thanks to Paul Wouters for reviewing the document.

7. References

7.1. Normative References

- [I-D.ietf-ipsecme-ikev2-intermediate] Smyslov, V., "Intermediate Exchange in the IKEv2 Protocol", Work in Progress, Internet-Draft, draft-ietf-ipsecme-ikev2-intermediate-10, 5 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-ipsecme-ikev2-intermediate-10.txt>>.
- [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>>.
- [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>>.
- [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>>.

7.2. Informative References

- [GROVER] Grover, L., "A Fast Quantum Mechanical Algorithm for Database Search", Proc. of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing (STOC 1996), 1996.

- [I-D.ietf-ipsecme-g-ikev2]
Smyslov, V. and B. Weis, "Group Key Management using IKEv2", Work in Progress, Internet-Draft, draft-ietf-ipsecme-g-ikev2-05, 18 March 2022, <<https://www.ietf.org/internet-drafts/draft-ietf-ipsecme-g-ikev2-05.txt>>.
- [I-D.tjhai-ikev2-beyond-64k-limit]
Tjhai, C., Heider, T., and V. Smyslov, "Beyond 64KB Limit of IKEv2 Payloads", Work in Progress, Internet-Draft, draft-tjhai-ikev2-beyond-64k-limit-01, 9 July 2021, <<https://www.ietf.org/archive/id/draft-tjhai-ikev2-beyond-64k-limit-01.txt>>.
- [RFC4302] Kent, S., "IP Authentication Header", RFC 4302, DOI 10.17487/RFC4302, December 2005, <<https://www.rfc-editor.org/info/rfc4302>>.
- [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <<https://www.rfc-editor.org/info/rfc4303>>.
- [RFC6023] Nir, Y., Tschofenig, H., Deng, H., and R. Singh, "A Childless Initiation of the Internet Key Exchange Version 2 (IKEv2) Security Association (SA)", RFC 6023, DOI 10.17487/RFC6023, October 2010, <<https://www.rfc-editor.org/info/rfc6023>>.
- [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>>.
- [RFC8019] Nir, Y. and V. Smyslov, "Protecting Internet Key Exchange Protocol Version 2 (IKEv2) Implementations from Distributed Denial-of-Service Attacks", RFC 8019, DOI 10.17487/RFC8019, November 2016, <<https://www.rfc-editor.org/info/rfc8019>>.
- [RFC8229] Pauly, T., Touati, S., and R. Mantha, "TCP Encapsulation of IKE and IPsec Packets", RFC 8229, DOI 10.17487/RFC8229, August 2017, <<https://www.rfc-editor.org/info/rfc8229>>.
- [RFC8391] Huelsing, A., Butin, D., Gazdag, S., Rijneveld, J., and A. Mohaisen, "XMSS: eXtended Merkle Signature Scheme", RFC 8391, DOI 10.17487/RFC8391, May 2018, <<https://www.rfc-editor.org/info/rfc8391>>.

[RFC8784] Fluhrrer, S., Kampanakis, P., McGrew, D., and V. Smyslov, "Mixing Preshared Keys in the Internet Key Exchange Protocol Version 2 (IKEv2) for Post-quantum Security", RFC 8784, DOI 10.17487/RFC8784, June 2020, <<https://www.rfc-editor.org/info/rfc8784>>.

Appendix A. Sample Multiple Key Exchanges

This appendix shows some examples of multiple key exchanges. These examples are purely for information purposes and they describe some message flow scenarios that may occur in establishing an IKE or CHILD SA. Note that some payloads that are not relevant to multiple key exchanges may be omitted for brevity.

A.1. No Additional Key Exchange Used

The initiator proposes two sets of optional additional key exchanges, but the responder does not support any of them. The responder chooses NONE for each set and consequently, IKE_INTERMEDIATE exchange does not take place and the exchange proceeds to IKE_AUTH phase. The resulting keying materials are the same as those derived with [RFC7296].

Initiator	Responder
<hr/>	
HDR(IKE_SA_INIT), SAI1(.. AKE*...), --->	
KEi1, Ni, N(IKEV2_FRAG_SUPPORTED),	
N(INTERMEDIATE_EXCHANGE_SUPPORTED)	
Proposal #1	
Transform ECR (ID = ENCR_AES_GCM_16,	
256-bit key)	
Transform PRF (ID = PRF_HMAC_SHA2_512)	
Transform KE (ID = Curve25519)	
Transform AKE1 (ID = PQ_KEM_1)	
Transform AKE1 (ID = PQ_KEM_2)	
Transform AKE1 (ID = NONE)	
Transform AKE2 (ID = PQ_KEM_3)	
Transform AKE2 (ID = PQ_KEM_4)	
Transform AKE2 (ID = NONE)	
	<--- HDR(IKE_SA_INIT), SAR1(.. AKE*...),
	KER1, Nr, N(IKEV2_FRAG_SUPPORTED),
	N(INTERMEDIATE_EXCHANGE_SUPPORTED)
	Proposal #1
	Transform ECR (ID = ENCR_AES_GCM_16,
	256-bit key)
	Transform PRF (ID = PRF_HMAC_SHA2_512)
	Transform KE (ID = Curve25519)
	Transform AKE1 (ID = NONE)
	Transform AKE2 (ID = NONE)
HDR(IKE_AUTH), SK{ IDi, AUTH, SAI2, TSi, TSr } --->	
	<--- HDR(IKE_AUTH), SK{ IDr, AUTH, SAR2,
	TSi, TSr }

A.2. Additional Key Exchange in the CREATE_CHILD_SA Exchange only

The exchanges below show that the initiator does not propose the use of additional key exchanges to establish an IKE SA, but they are required in order to establish a Child SA. In order to establish a fully quantum-resistant IPsec SA, both peers include CHILDLESS_IKEV2_SUPPORTED notification in their exchange so that the first Child SA is not created in IKE_AUTH, but instead the IKE SA is immediately rekeyed using CREATED_CHILD_SA. Any Child SA will have to be created via subsequent CREATED_CHILD_SA exchange.

```

Initiator                                     Responder
-----
HDR(IKE_SA_INIT), Sai1, --->
KEi1, Ni, N(IKEV2_FRAG_SUPPORTED),
N(CHILDLESS_IKEV2_SUPPORTED)
      <--- HDR(IKE_SA_INIT), Sar1,
      Ker1, Nr, N(IKEV2_FRAG_SUPPORTED),
      N(CHILDLESS_IKEV2_SUPPORTED)
HDR(IKE_AUTH), SK{ IDi, AUTH } --->
      <--- HDR(IKE_AUTH), SK{ IDr, AUTH }
HDR(CREATE_CHILD_SA), SK{ Sai(.. AKE*...), Ni, KEi } --->
  Proposal #1
    Transform ECR (ID = ENCR_AES_GCM_16,
                  256-bit key)
    Transform PRF (ID = PRF_HMAC_SHA2_512)
    Transform KE (ID = Curve25519)
    Transform AKE1 (ID = PQ_KEM_1)
    Transform AKE1 (ID = PQ_KEM_2)
    Transform AKE2 (ID = PQ_KEM_5)
    Transform AKE2 (ID = PQ_KEM_6)
    Transform AKE2 (ID = NONE)
      <--- HDR(CREATE_CHILD_SA), SK{ Sar(.. AKE*...),
      Nr, Ker,
      N(ADDITIONAL_KEY_EXCHANGE)(link1) }
        Proposal #1
          Transform ECR (ID = ENCR_AES_GCM_16,
                        256-bit key)
          Transform PRF (ID = PRF_HMAC_SHA2_512)
          Transform KE (ID = Curve25519)
          Transform AKE1 (ID = PQ_KEM_2)
          Transform AKE2 (ID = PQ_KEM_5)

HDR(IKE_FOLLOWUP_KEY), SK{ KEi(1), --->
N(ADDITIONAL_KEY_EXCHANGE)(link1) }
      <--- HDR(IKE_FOLLOWUP_KEY), SK{ Ker(1),
      N(ADDITIONAL_KEY_EXCHANGE)(link2) }
HDR(IKE_FOLLOWUP_KEY), SK{ KEi(2), --->
N(ADDITIONAL_KEY_EXCHANGE)(link2) }
      <--- HDR(IKE_FOLLOWUP_KEY), SK{ Ker(2) }

```

A.3. Not Matching Proposal for Additional Key Exchanges

The initiator proposes the combination of PQ_KEM_1, PQ_KEM_2, PQ_KEM_3, and PQ_KEM_4 as the additional key exchanges. The initiator indicates, using the key exchange method NONE, that either PQ_KEM_1 or PQ_KEM_2 must be used to establish a security association. The responder, although supports the optional PQ_KEM_3 and PQ_KEM_4 method, does not support either PQ_KEM_1 or PQ_KEM_2

mandatory method and therefore responds with NO_PROPOSAL_CHOSEN notification.

Initiator	Responder

HDR(IKE_SA_INIT), SAi1(.. AKE*...), --->	
KEi1, Ni, N(IKEV2_FRAG_SUPPORTED),	
N(INTERMEDIATE_EXCHANGE_SUPPORTED)	
Proposal #1	
Transform ECR (ID = ENCR_AES_GCM_16,	
256-bit key)	
Transform PRF (ID = PRF_HMAC_SHA2_512)	
Transform KE (ID = Curve25519)	
Transform AKE1 (ID = PQ_KEM_1)	
Transform AKE1 (ID = PQ_KEM_2)	
Transform AKE2 (ID = PQ_KEM_3)	
Transform AKE2 (ID = PQ_KEM_4)	
Transform AKE2 (ID = NONE)	
	<---- HDR(IKE_SA_INIT), N(NO_PROPOSAL_CHOSEN)

Appendix B. Alternative Design

This section gives an overview on a number of alternative approaches that we have considered, but later discarded. These approaches are:

- * Sending the classical and post-quantum key exchanges as a single transform

We considered combining the various key exchanges into a single large KE payload; this effort is documented in a previous version of this draft (draft-tjhai-ipsecme-hybrid-qske-ikev2-01). This does allow us to cleanly apply hybrid key exchanges during the child SA; however it does add considerable complexity, and requires an independent fragmentation solution.

- * Sending post-quantum proposals and policies in KE payload only

With the objective of not introducing unnecessary notify payloads, we considered communicating the hybrid post-quantum proposal in the KE payload during the first pass of the protocol exchange. Unfortunately, this design is susceptible to the following downgrade attack. Consider the scenario where there is an MitM attacker sitting between an initiator and a responder. The initiator proposes, through SAi payload, to use a hybrid post-quantum group and as a backup a Diffie-Hellman group, and through KEi payload, the initiator proposes a list of hybrid post-quantum proposals and policies. The MitM attacker intercepts this traffic and replies with N(INVALID_KEY_PAYLOAD) suggesting to downgrade to

the backup Diffie-Hellman group instead. The initiator then resends the same SA_i payload and the KE_i payload containing the public value of the backup Diffie-Hellman group. Note that the attacker may forward the second IKE_SA_INIT message only to the responder, and therefore at this point in time, the responder will not have the information that the initiator prefers the hybrid group. Of course, it is possible for the responder to have a policy to reject an IKE_SA_INIT message that (a) offers a hybrid group but not offering the corresponding public value in the KE_i payload; and (b) the responder has not specifically acknowledged that it does not support the requested hybrid group. However, the checking of this policy introduces unnecessary protocol complexity. Therefore, in order to fully prevent any downgrade attacks, using KE payload alone is not sufficient and that the initiator MUST always indicate its preferred post-quantum proposals and policies in a notify payload in the subsequent IKE_SA_INIT messages following a N(INVALID_KEY_PAYLOAD) response.

- * New payload types to negotiate hybrid proposal and to carry post-quantum public values

Semantically, it makes sense to use a new payload type, which mimics the SA payload, to carry a hybrid proposal. Likewise, another new payload type that mimics the KE payload, could be used to transport hybrid public value. Although, in theory a new payload type could be made backwards compatible by not setting its critical flag as per Section 2.5 of RFC7296, we believe that it may not be that simple in practice. Since the original release of IKEv2 in RFC4306, no new payload type has ever been proposed and therefore, this creates a potential risk of having a backward compatibility issue from non-conforming RFC IKEv2 implementations. Since we could not see any other compelling advantages apart from a semantic one, we use the existing transform type and notify payloads instead. In fact, as described above, we use the KE payload in the first IKE_SA_INIT request round and the notify payload to carry the post-quantum proposals and policies. We use one or more of the existing KE payloads to carry the hybrid public values.

- * Hybrid public value payload

One way to transport the negotiated hybrid public payload, which contains one classical Diffie-Hellman public value and one or more post-quantum public values, is to bundle these into a single KE payload. Alternatively, these could also be transported in a single new hybrid public value payload, but following the same reasoning as above, this may not be a good idea from a backward compatibility perspective. Using a single KE payload would

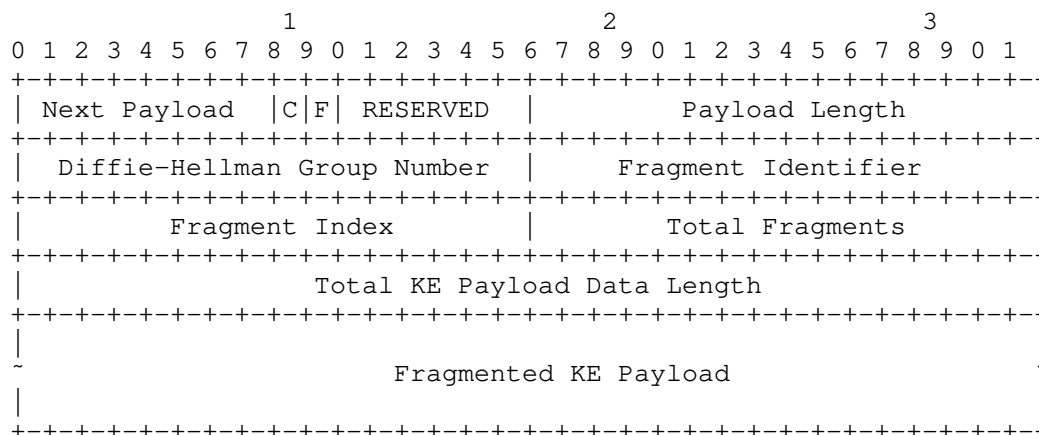
require an encoding or formatting to be defined so that both peers are able to compose and extract the individual public values. However, we believe that it is cleaner to send the hybrid public values in multiple KE payloads--one for each group or algorithm. Furthermore, at this point in the protocol exchange, both peers should have indicated support of handling multiple KE payloads.

* Fragmentation

Handling of large IKE_SA_INIT messages has been one of the most challenging tasks. A number of approaches have been considered and the two prominent ones that we have discarded are outlined as follows.

The first approach was to treat the entire IKE_SA_INIT message as a stream of bytes, which we then split it into a number of fragments, each of which is wrapped onto a payload that would fit into the size of the network MTU. The payload that wraps each fragment is a new payload type and it was envisaged that this new payload type will not cause a backward compatibility issue because at this stage of the protocol, both peers should have indicated support of fragmentation in the first pass of the IKE_SA_INIT exchange. The negotiation of fragmentation is performed using a notify payload, which also defines supporting parameters such as the size of fragment in octets and the fragment identifier. The new payload that wraps each fragment of the messages in this exchange is assigned the same fragment identifier. Furthermore, it also has other parameters such as a fragment index and total number of fragments. We decided to discard this approach due to its blanket approach to fragmentation. In cases where only a few payloads need to be fragmented, we felt that this approach is overly complicated.

Another idea that was discarded was fragmenting an individual payload without introducing a new payload type. The idea was to use the 9-th bit (the bit after the critical flag in the RESERVED field) in the generic payload header as a flag to mark that this payload is fragmented. As an example, if a KE payload is to be fragmented, it may look as follows.



When the flag F is set, this means the current KE payload is a fragment of a larger KE payload. The Payload Length field denotes the size of this payload fragment in octets--including the size of the generic payload header. The two-octet RESERVED field following Diffie-Hellman Group Number was to be used as a fragment identifier to help assembly and disassembly of fragments. The Fragment Index and Total Fragments fields are self-explanatory. The Total KE Payload Data Length indicates the size of the assembled KE payload data in octets. Finally, the actual fragment is carried in Fragment KE Payload field.

We discarded this approach because we believe that the working group may not be happy using the RESERVED field to change the format of a packet and that implementers may not like the complexity added from checking the fragmentation flag in each received payload. More importantly, fragmenting the messages in this way may leave the system to be more prone to denial of service (DoS) attacks. By using IKE_INTERMEDIATE to transport the large post-quantum key exchange payloads, there is no longer any issue with fragmentation.

* Group sub-identifier

As discussed before, each group identifier is used to distinguish a post-quantum algorithm. Further classification could be made on a particular post-quantum algorithm by assigning additional value alongside the group identifier. This sub-identifier value may be used to assign different security parameter sets to a given post-quantum algorithm. However, this level of details does not fit the principles of the document where it should deal with generic hybrid key exchange protocol, not a specific ciphersuite. Furthermore, there are enough Diffie-Hellman group identifiers should this be required in the future.

Authors' Addresses

C. Tjhai
Post-Quantum
Email: cjt@post-quantum.com

M. Tomlinson
Post-Quantum
Email: mt@post-quantum.com

G. Bartlett
Quantum Secret
Email: graham.ietf@gmail.com

S. Fluhrer
Cisco Systems
Email: sfluhrer@cisco.com

D. Van Geest
ISARA Corporation
Email: daniel.vangeest@isara.com

O. Garcia-Morchon
Philips
Email: oscar.garcia-morchon@philips.com

Valery Smyslov
ELVIS-PLUS
Email: svan@elvis.ru

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 22 May 2022

D. Fedyk
E. Kinzie
LabN Consulting, L.L.C.
18 November 2021

Definitions of Managed Objects for IP Traffic Flow Security
draft-ietf-ipsecme-mib-iptfs-03

Abstract

This document describes managed objects for the the management of IP Traffic Flow Security additions to IKEv2 and IPsec. This document provides a read only version of the objects defined in the YANG module for the same purpose.

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 22 May 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	2
2. Terminology & Concepts	3
3. Overview	3
4. Management Objects	3
4.1. MIB Tree	3
4.2. SNMP	5
5. IANA Considerations	19
6. Security Considerations	19
7. Acknowledgements	20
8. References	20
8.1. Normative References	20
8.2. Informative References	22
Authors' Addresses	22

1. Introduction

This document defines a Management Information Base (MIB) module for use with network management protocols in the Internet community. Traffic Flow Security (IP-TFS) extensions as defined in [I-D.ietf-ipsecme-iptfs]. IP-TFS provides enhancements to an IPsec tunnel Security Association to provide improved traffic confidentiality.

For a detailed overview of the documents that describe the current Internet-Standard Management Framework, please refer to section 7 of [RFC3410].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. MIB objects are generally accessed through the Simple Network Management Protocol (SNMP). Objects in the MIB are defined using the mechanisms defined in the Structure of Management Information (SMI). This memo specifies a MIB module that is compliant to the SMIV2, which is described in STD 58, [RFC2578], STD 58, [RFC2579] and STD 58, [RFC2580].

The objects defined here are the same as [I-D.ietf-ipsecme-yang-iptfs] with the exception that only operational data is supported. This module uses the YANG model as a reference point for managed objects. Note an IETF MIB model for IPsec was never standardized however the structures here could be adapted to existing MIB implementations.

2. Terminology & Concepts

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 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Overview

This document defines configuration and operational parameters of IP traffic flow security (IP-TFS). IP-TFS, defined in [I-D.ietf-ipsecme-iptfs], configures a security association for tunnel mode IPsec with characteristics that improve traffic confidentiality and reduce bandwidth efficiency loss.

This document is based on the concepts and management model defined in [I-D.ietf-ipsecme-yang-iptfs]. This documents assume familiarity with IP security concepts described in [RFC4301], IP-TFS as described in [I-D.ietf-ipsecme-iptfs] and the IP-TFS management model described in [I-D.ietf-ipsecme-yang-iptfs].

This document specifies an extensible operational model for IP-TFS. It reuses the management model defined in [I-D.ietf-ipsecme-yang-iptfs]. It allows SNMP systems to read configured and operational objects of IPTFS.

4. Management Objects

4.1. MIB Tree

The following is the MIB registration tree diagram for the IP-TFS extensions.

IETF-IPTFS-MIB registration tree (generated by smidump 0.4.8)

```
--iptfsMIB(1.3.6.1.3.500)
+--iptfsMIBObjects(1)
|   +--iptfsGroup(1)
|   |   +--iptfsConfigTable(1)
|   |   |   +--iptfsConfigTableEntry(1) [iptfsConfigSaIndex]
|   |   |   |   +-- --- Integer32      iptfsConfigSaIndex(1)
|   |   |   |   +-- r-n TruthValue    congestionControl(2)
|   |   |   |   +-- r-n TruthValue    usePathMtu(3)
|   |   |   |   +-- r-n UnsignedShort outerPacketSize(4)
|   |   |   |   +-- r-n Counter64      l2FixedRate(5)
|   |   |   |   +-- r-n Counter64      l3FixedRate(6)
|   |   |   |   +-- r-n TruthValue     dontFragment(7)
```

```

    +--- r-n NanoSeconds    maxAggregationTime(8)
    +--- r-n Unsigned32     windowSize(9)
    +--- r-n TruthValue     sendImmediately(10)
    +--- r-n NanoSeconds    lostPktTimerInt(11)
+---ipsecStatsGroup(2)
  +---ipsecStatsTable(1)
    +---ipsecStatsTableEntry(1) [ipsecSaIndex]
      +--- --- Integer32 ipsecSaIndex(1)
      +--- r-n Counter64 txPackets(2)
      +--- r-n Counter64 txOctets(3)
      +--- r-n Counter64 txDropPackets(4)
      +--- r-n Counter64 rxPackets(5)
      +--- r-n Counter64 rxOctets(6)
      +--- r-n Counter64 rxDropPackets(7)
+---iptfsInnerStatsGroup(3)
  +---iptfsInnerStatsTable(1)
    +---iptfsInnerStatsTableEntry(1) [iptfsInnerSaIndex]
      +--- --- Integer32 iptfsInnerSaIndex(1)
      +--- r-n Counter64 txInnerPackets(2)
      +--- r-n Counter64 txInnerOctets(3)
      +--- r-n Counter64 rxInnerPackets(4)
      +--- r-n Counter64 rxInnerOctets(5)
      +--- r-n Counter64 rxIncompleteInnerPackets(6)
+---iptfsOuterStatsGroup(4)
  +---iptfsOuterStatsTable(1)
    +---iptfsOuterStatsTableEntry(1) [iptfsSaIndex]
      +--- --- Integer32 iptfsSaIndex(1)
      +--- r-n Counter64 txExtraPadPackets(2)
      +--- r-n Counter64 txExtraPadOctets(3)
      +--- r-n Counter64 txAllPadPackets(4)
      +--- r-n Counter64 txAllPadOctets(5)
      +--- r-n Counter64 rxExtraPadPackets(6)
      +--- r-n Counter64 rxExtraPadOctets(7)
      +--- r-n Counter64 rxAllPadPackets(8)
      +--- r-n Counter64 rxAllPadOctets(9)
      +--- r-n Counter64 rxErroredPackets(10)
      +--- r-n Counter64 rxMissedPackets(11)
+---iptfsMIBConformance(2)
  +---iptfsMIBConformances(1)
    | +---iptfsMIBCompliance(1)
  +---iptfsMIBGroups(2)
    +---iptfsMIBConfGroup(1)
    +---ipsecStatsConfGroup(2)
    +---iptfsInnerStatsConfGroup(3)
    +---iptfsOuterStatsConfGroup(4)

```


4.2. SNMP

The following is the MIB for IP-TFS. The Congestion control algorithm in [RFC5348] is referenced in the MIB text.

```
-- *-----
-- *
-- *-----
```

```
IETF-IPTFS-MIB DEFINITIONS ::= BEGIN
```

```
  IMPORTS
```

```
    MODULE-IDENTITY, OBJECT-TYPE,
    Integer32, Unsigned32, Counter64, experimental
      FROM SNMPv2-SMI
    MODULE-COMPLIANCE, OBJECT-GROUP
      FROM SNMPv2-CONF
    TEXTUAL-CONVENTION,
    TruthValue
      FROM SNMPv2-TC;
```

```
  iptfsMIB MODULE-IDENTITY
```

```
    LAST-UPDATED "202111180000Z"
    ORGANIZATION "IETF IPsecme Working Group"
    CONTACT-INFO
```

```
      "
```

```
        Author: Don Fedyk
                <mailto:dfedyk@labn.net>
```

```
        Author: Eric Kinzie
                <mailto:ekinzie.labn.net>"
```

```
  DESCRIPTION
```

```
    "This module defines the configuration and operational
    state for managing the IP Traffic Flow Security
    functionality [RFC XXXX]. Copyright (c) 2021 IETF
    Trust and the persons identified as authors of the
    code. All rights reserved.
```

```
    Redistribution and use in source and binary forms,
    with or without modification, is permitted pursuant
    to, and subject to the license terms contained in,
    the Simplified BSD License set forth in Section 4.c
    of the IETF Trust's Legal Provisions Relating to IETF
    Documents (https://trustee.ietf.org/license-info).
```

```
    This version of this SNMP MIB module is part of RFC XXXX
    (https://tools.ietf.org/html/rfcXXXX); see the RFC
```

itself for full legal notices."

```
REVISION "202111180000Z"
DESCRIPTION
    "Initial revision. Derived from the IPTFS Yang Model."
::= { experimental 500 }
--
-- Textual Conventions
--

UnsignedShort ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "d"
    STATUS current
    DESCRIPTION "xs:unsignedShort"
    SYNTAX      Unsigned32 (0 .. 65535)

NanoSeconds ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "d"
    STATUS current
    DESCRIPTION
        "Represents time unit value in nanoseconds."
    SYNTAX      Counter64

-- Objects, Notifications & Conformances

iptfsMIBObjects      OBJECT IDENTIFIER
    ::= { iptfsMIB 1 }
iptfsMIBConformance OBJECT IDENTIFIER
    ::= { iptfsMIB 2 }

--
-- IPTFS MIB Object Groups
--

iptfsGroup OBJECT IDENTIFIER
    ::= { iptfsMIBObjects 1 }

ipsecStatsGroup OBJECT IDENTIFIER
    ::= { iptfsMIBObjects 2 }

iptfsInnerStatsGroup OBJECT IDENTIFIER
    ::= { iptfsMIBObjects 3 }

iptfsOuterStatsGroup OBJECT IDENTIFIER
    ::= { iptfsMIBObjects 4 }
```

```

iptfsConfigTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF IptfsConfigTableEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The table containing configuration information for
        IPTFS."
    ::= { iptfsGroup 1 }

```

```

iptfsConfigTableEntry OBJECT-TYPE
    SYNTAX      IptfsConfigTableEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry (conceptual row) containing the information on
        a particular IPTFS SA."
    INDEX       { iptfsConfigSaIndex }
    ::= { iptfsConfigTable 1 }

```

```

IptfsConfigTableEntry ::= SEQUENCE {
    iptfsConfigSaIndex      Integer32,

    -- identifier information
    congestionControl        TruthValue,
    usePathMtu               TruthValue,
    outerPacketSize          UnsignedShort,
    l2FixedRate              Counter64,
    l3FixedRate              Counter64,
    dontFragment             TruthValue,
    maxAggregationTime       NanoSeconds,
    windowSize               Unsigned32,
    sendImmediately         TruthValue,
    lostPktTimerInt          NanoSeconds
}

```

```

iptfsConfigSaIndex OBJECT-TYPE
    SYNTAX      Integer32 (1..16777215)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A unique value, greater than zero, for each SA.
        It is recommended that values are assigned contiguously
        starting from 1.

        The value for each entry must remain constant at least
        from one re-initialization of entity's network management
        system to the next re-initialization."
    ::= { iptfsConfigTableEntry 1 }

```

```
congestionControl OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "When set to true, the default, this enables the
        congestion control on-the-wire exchange of data that is
        required by congestion control algorithms as defined by
        RFC 5348.  When set to false, IP-TFS sends fixed-sized
        packets over an IP-TFS tunnel at a constant rate."
    DEFVAL { false }
    ::= { iptfsConfigTableEntry 2 }

usePathMtu OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "Packet size is either auto-discovered or manually
        configured.  If usePathMtu is true the system utilizes
        path-mtu to determine maximum IPTFS packet size.  If
        the packet size is explicitly configured then it will
        only be adjusted downward if use-path-mtu is set."
    ::= { iptfsConfigTableEntry 3 }

outerPacketSize OBJECT-TYPE
    SYNTAX      UnsignedShort
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "On Transmission, the size of the outer encapsulating
        tunnel packet (i.e., the IP packet containing the ESP
        payload)."
    ::= { iptfsConfigTableEntry 4 }

12FixedRate OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "TFS bit rate may be specified at layer 2 wire rate.  On
        transmission, target bandwidth/bit rate in bps for iptfs
        tunnel.  This rate is the nominal timing for the fixed
        size packet.  If congestion control is enabled the rate
        may be adjusted down (or up if unset)."
    ::= { iptfsConfigTableEntry 5 }

13FixedRate OBJECT-TYPE
```

```
SYNTAX          Counter64
MAX-ACCESS      read-only
STATUS          current
DESCRIPTION
    "TFS bit rate may be specified at layer 3 packet rate.
    On Transmission, target bandwidth/bit rate in bps for
    iptfs tunnel. This rate is the nominal timing for the
    fixed size packet. If congestion control is enabled the
    rate may be adjusted down (or up if unset)."
```

```
 ::= { iptfsConfigTableEntry 6 }
```

```
dontFragment OBJECT-TYPE
SYNTAX          TruthValue
MAX-ACCESS      read-only
STATUS          current
DESCRIPTION
    "On transmission, disable packet fragmentation across
    consecutive iptfs tunnel packets; inner packets larger
    than what can be transmitted in outer packets will be
    dropped."
```

```
 ::= { iptfsConfigTableEntry 7 }
```

```
maxAggregationTime OBJECT-TYPE
SYNTAX          NanoSeconds
MAX-ACCESS      read-only
STATUS          current
DESCRIPTION
    "On transmission, maximum aggregation time is the
    maximum length of time a received inner packet can be
    held prior to transmission in the iptfs tunnel. Inner
    packets that would be held longer than this time, based
    on the current tunnel configuration will be dropped
    rather than be queued for transmission."
```

```
 ::= { iptfsConfigTableEntry 8 }
```

```
windowSize OBJECT-TYPE
SYNTAX          Unsigned32(0..65535)
MAX-ACCESS      read-only
STATUS          current
DESCRIPTION
    "On reception, the maximum number of out-of-order
    packets that will be reordered by an iptfs receiver
    while performing the reordering operation. The value 0
    disables any reordering."
```

```
 ::= { iptfsConfigTableEntry 9 }
```

```
sendImmediately OBJECT-TYPE
SYNTAX          TruthValue
```

```

MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
    "On reception, send inner packets as soon as possible, do
    not wait for lost or misordered outer packets.
    Selecting this option reduces the inner (user) packet
    delay but can amplify out-of-order delivery of the inner
    packet stream in the presence of packet aggregation and
    any reordering."
 ::= { iptfsConfigTableEntry 10 }

lostPktTimerInt OBJECT-TYPE
    SYNTAX      NanoSeconds
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "On reception, this interval defines the length of time
        an iptfs receiver will wait for a missing packet before
        considering it lost.  If not using send-immediately,
        then each lost packet will delay inner (user) packets
        until this timer expires.  Setting this value too low can
        impact reordering and reassembly."
    ::= { iptfsConfigTableEntry 11 }

ipsecStatsTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF IpsecStatsTableEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The table containing basic statistics on IPsec."
    ::= { ipsecStatsGroup 1 }

ipsecStatsTableEntry OBJECT-TYPE
    SYNTAX      IpsecStatsTableEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry (conceptual row) containing the information on
        a particular IKE SA."
    INDEX       { ipsecSaIndex }
    ::= { ipsecStatsTable 1 }

IpsecStatsTableEntry ::= SEQUENCE {
    ipsecSaIndex      Integer32,
    -- packet statistics information
    txPackets         Counter64,
    txOctets           Counter64,

```

```
    txDropPackets          Counter64,  
    rxPackets              Counter64,  
    rxOctets               Counter64,  
    rxDropPackets          Counter64  
}
```

ipsecSaIndex OBJECT-TYPE

SYNTAX Integer32 (1..16777215)

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"A unique value, greater than zero, for each SA.

It is recommended that values are assigned contiguously starting from 1.

The value for each entry must remain constant at least from one re-initialization of entity's network management system to the next re-initialization."

::= { ipsecStatsTableEntry 1 }

txPackets OBJECT-TYPE

SYNTAX Counter64

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Outbound Packet count."

::= { ipsecStatsTableEntry 2 }

txOctets OBJECT-TYPE

SYNTAX Counter64

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Outbound Packet bytes."

::= { ipsecStatsTableEntry 3 }

txDropPackets OBJECT-TYPE

SYNTAX Counter64

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Outbound dropped packets count."

::= { ipsecStatsTableEntry 4 }

rxPackets OBJECT-TYPE

SYNTAX Counter64

MAX-ACCESS read-only

```

    STATUS      current
    DESCRIPTION
        "Inbound Packet count."
    ::= { ipsecStatsTableEntry 5 }

rxOctets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS   read-only
    STATUS      current
    DESCRIPTION
        "Inbound Packet bytes."
    ::= { ipsecStatsTableEntry 6 }

rxDropPackets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS   read-only
    STATUS      current
    DESCRIPTION
        "Inbound Dropped packets"
    ::= { ipsecStatsTableEntry 7 }

iptfsInnerStatsTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF IptfsInnerSaEntry
    MAX-ACCESS   not-accessible
    STATUS      current
    DESCRIPTION
        "The table containing information on IPTFS
        Inner Packets."
    ::= { iptfsInnerStatsGroup 1 }

iptfsInnerStatsTableEntry OBJECT-TYPE
    SYNTAX      IptfsInnerSaEntry
    MAX-ACCESS   not-accessible
    STATUS      current
    DESCRIPTION
        "An entry containing the information on
        a particular tfs SA."
    INDEX       { iptfsInnerSaIndex }
    ::= { iptfsInnerStatsTable 1 }

IptfsInnerSaEntry ::= SEQUENCE {
    iptfsInnerSaIndex      Integer32,

    txInnerPackets         Counter64,
    txInnerOctets          Counter64,
    rxInnerPackets         Counter64,
    rxInnerOctets          Counter64,
    rxIncompleteInnerPackets Counter64
}
```



```
}

iptfsInnerSaIndex OBJECT-TYPE
    SYNTAX      Integer32 (1..16777215)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A unique value, greater than zero, for each SA.
        It is recommended that values are assigned contiguously
        starting from 1.

        The value for each entry must remain constant at least
        from one re-initialization of entity's network management
        system to the next re-initialization."
    ::= { iptfsInnerStatsTableEntry 1 }

txInnerPackets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Total number of IP-TFS inner packets sent. This count
        is whole packets only. A fragmented packet counts as
        one packet."
    ::= { iptfsInnerStatsTableEntry 2 }

txInnerOctets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Total number of IP-TFS inner octets sent. This is
        inner packet octets only. Does not count padding."
    ::= { iptfsInnerStatsTableEntry 3 }

rxInnerPackets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Total number of IP-TFS inner packets received."
    ::= { iptfsInnerStatsTableEntry 4 }

rxInnerOctets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
```

```
        "Total number of IP-TFS inner octets received. Does
        not include padding or overhead."
 ::= { iptfsInnerStatsTableEntry 5 }

rxIncompleteInnerPackets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "Total number of IP-TFS inner packets that were
        incomplete. Usually this is due to fragments not
        received. Also, this may be due to misordering or
        errors in received outer packets."
 ::= { iptfsInnerStatsTableEntry 6 }

iptfsOuterStatsTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF IptfsOuterSaEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION
        "The table containing information on IPTFS."
 ::= { iptfsOuterStatsGroup 1 }

iptfsOuterStatsTableEntry OBJECT-TYPE
    SYNTAX      IptfsOuterSaEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION
        "An entry containing the information on
        a particular tfs SA."
    INDEX       { iptfsSaIndex }
 ::= { iptfsOuterStatsTable 1 }

IptfsOuterSaEntry ::= SEQUENCE {
    iptfsSaIndex      Integer32,

-- iptfs packet statistics information
    txExtraPadPackets Counter64,
    txExtraPadOctets  Counter64,
    txAllPadPackets   Counter64,
    txAllPadOctets     Counter64,
    rxExtraPadPackets  Counter64,
    rxExtraPadOctets   Counter64,
    rxAllPadPackets    Counter64,
    rxAllPadOctets     Counter64,
    rxErroredPackets   Counter64,
    rxMissedPackets    Counter64
}
```

iptfsSaIndex OBJECT-TYPE
SYNTAX Integer32 (1..16777215)
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
 "A unique value, greater than zero, for each SA.
 It is recommended that values are assigned contiguously
 starting from 1.

 The value for each entry must remain constant at least
 from one re-initialization of entity's network management
 system to the next re-initialization."
 ::= { iptfsOuterStatsTableEntry 1 }

txExtraPadPackets OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "Total number of transmitted outer IP-TFS packets that
 included some padding."
 ::= { iptfsOuterStatsTableEntry 2 }

txExtraPadOctets OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "Total number of transmitted octets of padding added to
 outer IP-TFS packets with data."
 ::= { iptfsOuterStatsTableEntry 3 }

txAllPadPackets OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "Total number of transmitted IP-TFS packets that were
 all padding with no inner packet data."
 ::= { iptfsOuterStatsTableEntry 4 }

txAllPadOctets OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "Total number transmitted octets of padding added to
 IP-TFS packets with no inner packet data."

```
 ::= { iptfsOuterStatsTableEntry 5 }

rxExtraPadPackets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "Total number of received outer IP-TFS packets that
        included some padding."
    ::= { iptfsOuterStatsTableEntry 6 }

rxExtraPadOctets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "Total number of received octets of padding added to
        outer IP-TFS packets with data."
    ::= { iptfsOuterStatsTableEntry 7 }

rxAllPadPackets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "Total number of received IP-TFS packets that were all
        padding with no inner packet data."
    ::= { iptfsOuterStatsTableEntry 8 }

rxAllPadOctets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "Total number received octets of padding added to
        IP-TFS packets with no inner packet data."
    ::= { iptfsOuterStatsTableEntry 9 }

rxErroredPackets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "Total number of IP-TFS outer packets dropped due to
        errors."
    ::= { iptfsOuterStatsTableEntry 10 }

rxMissedPackets OBJECT-TYPE
```

```
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "Total number of IP-TFS outer packets missing indicated
    by missing sequence number."
 ::= { iptfsOuterStatsTableEntry 11 }

--
-- Iptfs Module Compliance
--

iptfsMIBConformances OBJECT IDENTIFIER
    ::= { iptfsMIBConformance 1 }

iptfsMIBGroups OBJECT IDENTIFIER
    ::= { iptfsMIBConformance 2 }

iptfsMIBCompliance MODULE-COMPLIANCE
    STATUS      current
    DESCRIPTION
        "The compliance statement for entities which
        implement the IPTFS MIB"
    MODULE      -- this module
        MANDATORY-GROUPS {
            iptfsMIBConfGroup,
            ipsecStatsConfGroup,
            iptfsInnerStatsConfGroup,
            iptfsOuterStatsConfGroup
        }

    ::= { iptfsMIBConformances 1 }

--
-- MIB Groups (Units of Conformance)
--

iptfsMIBConfGroup OBJECT-GROUP
    OBJECTS {
        congestionControl,
        usePathMtu,
        outerPacketSize ,
        l2FixedRate ,
        l3FixedRate ,
        dontFragment,
        maxAggregationTime,
        windowSize,
        sendImmediately,
```

```
        lostPktTimerInt
    }
    STATUS    current
    DESCRIPTION
        "A collection of objects providing per SA IPTFS
        Configuration."
    ::= { iptfsMIBGroups 1 }

ipsecStatsConfGroup OBJECT-GROUP
    OBJECTS {
        txPackets,
        txOctets,
        txDropPackets,
        rxPackets,
        rxOctets,
        rxDropPackets
    }
    STATUS    current
    DESCRIPTION
        "A collection of objects providing per SA Basic
        Stats."
    ::= { iptfsMIBGroups 2 }

iptfsInnerStatsConfGroup OBJECT-GROUP
    OBJECTS {
        txInnerPackets,
        txInnerOctets,
        rxInnerPackets,
        rxInnerOctets,
        rxIncompleteInnerPackets
    }
    STATUS    current
    DESCRIPTION
        "A collection of objects providing per SA IPTFS
        Inner Packet Statistics."
    ::= { iptfsMIBGroups 3 }

iptfsOuterStatsConfGroup OBJECT-GROUP
    OBJECTS {
        txExtraPadPackets,
        txExtraPadOctets,
        txAllPadPackets,
        txAllPadOctets,
        rxExtraPadPackets,
        rxExtraPadOctets,
        rxAllPadPackets,
        rxAllPadOctets,
```

```
        rxErroredPackets,
        rxMissedPackets
    }
    STATUS    current
    DESCRIPTION
        "A collection of objects providing per SA IPTFS
        Outer Packet Statistics."
    ::= { iptfsMIBGroups 4 }
```

END

5. IANA Considerations

The MIB module in this document uses the following IANA-assigned OBJECT IDENTIFIER value, recorded in the SMI Numbers registry:

Descriptor	OBJECT IDENTIFIER value
iptfs	TBA IANA
ipsec	TBA IANA

6. Security Considerations

The MIB specified in this document can read the operational and configured behavior of IP traffic flow security, for the implications regarding write configuration consult the [I-D.ietf-ipsecme-iptfs] which defines the functionality.

There are no management objects defined in this MIB module that have a MAX-ACCESS clause of read-write and/or read-create. So, if this MIB module is implemented correctly, then there is no risk that an intruder can alter or create any management objects of this MIB module via direct SNMP SET operations.

Some of the objects in this MIB module may be considered sensitive or vulnerable in some network environments. This includes INDEX objects with a MAX-ACCESS of not-accessible, and any indices from other modules exposed via AUGMENTS. It is thus important to control even GET and/or NOTIFY access to these objects and possibly to even encrypt the values of these objects when sending them over the network via SNMP. These are the tables and objects and their sensitivity/vulnerability:

- * iptfsOuterStatsTable - IPTFS hides the traffic flows through the network, anywhere that access to read SNMP statistics is enabled needs to be protected from third party observation.

SNMP versions prior to SNMPv3 did not include adequate security. Even if the network itself is secure (for example by using IPsec), there is no control as to who on the secure network is allowed to access and GET/SET (read/change/create/delete) the objects in this MIB module.

Implementations SHOULD provide the security features described by the SNMPv3 framework (see [RFC3410]), and implementations claiming compliance to the SNMPv3 standard MUST include full support for authentication and privacy via the User-based Security Model (USM) [RFC3414] with the AES cipher algorithm [RFC3826]. Implementations MAY also provide support for the Transport Security Model (TSM) [RFC5591] in combination with a secure transport such as SSH [RFC5592] or TLS/DTLS [RFC6353].

Further, deployment of SNMP versions prior to SNMPv3 is NOT RECOMMENDED. Instead, it is RECOMMENDED to deploy SNMPv3 and to enable cryptographic security. It is then a customer/operator responsibility to ensure that the SNMP entity giving access to an instance of this MIB module is properly configured to give access to the objects only to those principals (users) that have legitimate rights to indeed GET or SET (change/create/delete) them.

7. Acknowledgements

The authors would like to thank Chris Hopps, Lou Berger and Tero Kivinen for their help and feedback on the MIB model.

8. References

8.1. Normative References

- [I-D.ietf-ipsecme-iptfs]
Hopps, C., "IP-TFS: Aggregation and Fragmentation Mode for ESP and its Use for IP Traffic Flow Security", Work in Progress, Internet-Draft, draft-ietf-ipsecme-iptfs-12, 8 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-ipsecme-iptfs-12.txt>>.

- [I-D.ietf-ipsecme-yang-iptfs]
Fedyk, D. and C. Hopps, "A YANG Data Model for IP Traffic Flow Security", Work in Progress, Internet-Draft, draft-ietf-ipsecme-yang-iptfs-03, 11 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-ipsecme-yang-iptfs-03.txt>>.
- [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>>.
- [RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <<https://www.rfc-editor.org/info/rfc2578>>.
- [RFC2579] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Textual Conventions for SMIv2", STD 58, RFC 2579, DOI 10.17487/RFC2579, April 1999, <<https://www.rfc-editor.org/info/rfc2579>>.
- [RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, DOI 10.17487/RFC3410, December 2002, <<https://www.rfc-editor.org/info/rfc3410>>.
- [RFC3414] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, DOI 10.17487/RFC3414, December 2002, <<https://www.rfc-editor.org/info/rfc3414>>.
- [RFC3826] Blumenthal, U., Maino, F., and K. McCloghrie, "The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model", RFC 3826, DOI 10.17487/RFC3826, June 2004, <<https://www.rfc-editor.org/info/rfc3826>>.
- [RFC5591] Harrington, D. and W. Hardaker, "Transport Security Model for the Simple Network Management Protocol (SNMP)", STD 78, RFC 5591, DOI 10.17487/RFC5591, June 2009, <<https://www.rfc-editor.org/info/rfc5591>>.

- [RFC5592] Harrington, D., Salowey, J., and W. Hardaker, "Secure Shell Transport Model for the Simple Network Management Protocol (SNMP)", RFC 5592, DOI 10.17487/RFC5592, June 2009, <<https://www.rfc-editor.org/info/rfc5592>>.
- [RFC6353] Hardaker, W., "Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP)", STD 78, RFC 6353, DOI 10.17487/RFC6353, July 2011, <<https://www.rfc-editor.org/info/rfc6353>>.
- [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>>.

8.2. Informative References

- [RFC2580] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Conformance Statements for SMIV2", STD 58, RFC 2580, DOI 10.17487/RFC2580, April 1999, <<https://www.rfc-editor.org/info/rfc2580>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<https://www.rfc-editor.org/info/rfc5348>>.

Authors' Addresses

Don Fedyk
LabN Consulting, L.L.C.

Email: dfedyk@labn.net

Eric Kinzie
LabN Consulting, L.L.C.

Email: ekinzie@labn.net

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 22 May 2022

D. Fedyk
C. Hopps
LabN Consulting, L.L.C.
18 November 2021

A YANG Data Model for IP Traffic Flow Security
draft-ietf-ipsecme-yang-iptfs-05

Abstract

This document describes a yang module for the management of IP Traffic Flow Security additions to IKEv2 and IPsec.

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 22 May 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology & Concepts	3
2. Overview	3
3. YANG Management	5
3.1. YANG Tree	5
3.2. YANG Module	7
4. IANA Considerations	19
4.1. Updates to the IETF XML Registry	19
4.2. Updates to the YANG Module Names Registry	19
5. Security Considerations	20
6. Acknowledgements	20
7. References	20
7.1. Normative References	20
7.2. Informative References	21
Appendix A. Examples	22
A.1. Example XML Configuration	22
A.2. Example XML Operational Data	23
A.3. Example JSON Configuration	24
A.4. Example JSON Operational Data	26
A.5. Example JSON Operational Statistics	27
Authors' Addresses	28

1. Introduction

This document defines a YANG module [RFC7950] for the management of the IP Traffic Flow Security (IP-TFS) extensions as defined in [I-D.ietf-ipsecme-iptfs]. IP-TFS provides enhancements to an IPsec tunnel Security Association to provide improved traffic confidentiality. Traffic confidentiality reduces the ability of traffic analysis to determine identity and correlate observable traffic patterns. IP-TFS offers efficiency when aggregating traffic in fixed size IPsec tunnel packets.

The YANG data model in this document conforms to the Network Management Datastore Architecture (NMDA) defined in [RFC8342].

The published YANG modules for IPsec are defined in [RFC9061]. This document uses these models as a general IPsec model that is augmented for IP-TFS. The models in [RFC9061] provide for both an IKE and an IKELESS model.

1.1. Terminology & Concepts

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 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Overview

This document defines configuration and operational parameters of IP traffic flow security (IP-TFS). IP-TFS, defined in [I-D.ietf-ipsecme-iptfs], defines a security association for tunnel mode IPsec with characteristics that improve traffic confidentiality and reduce bandwidth efficiency loss. These documents assume familiarity with IP security concepts described in [RFC4301].

IP-TFS uses tunnel mode to improve confidentiality by hiding inner packet identifiable information, packet size and packet timing. IP-TFS provides a general capability allowing aggregation of multiple packets in uniform size outer tunnel ipsec packets. It maintains the outer packet size by utilizing combinations of aggregating, padding and fragmenting inner packets to fill out the IPsec outer tunnel packet. Zero byte padding is used to fill the packet when no data is available to send.

This document specifies an extensible configuration model for IP-TFS. This version utilizes the capabilities of IP-TFS to configure fixed size IP-TFS Packets that are transmitted at a constant rate. This model is structured to allow for different types of operation through future augmentation.

IP-TFS YANG augments IPsec YANG model from [RFC9061]. IP-TFS makes use of IPsec tunnel mode and adds a small number configuration items to tunnel mode IPsec. As defined in [I-D.ietf-ipsecme-iptfs], any SA configured to use IP-TFS supports only IP-TFS packets i.e. no mixed IPsec modes.

The behavior for IP-TFS is controlled by the source. The self-describing format of an IP-TFS packets allows a sending side to adjust the packet-size and timing independently from any receiver. Both directions are also independent, e.g. IP-TFS may be run only in one direction. This means that counters, which are created here for both directions may be 0 or not updated in the case of an SA that uses IP-TFS only in on direction.

Cases where IP-TFS statistics are active for one direction:

- * SA one direction - IP-TFS enabled
- * SA both directions - IP-TFS only enabled in one direction

Case where IP-TFS statistics are for both directions:

- * SA both directions - IP-TFS enable for both directions

The IP-TFS model support IP-TFS configuration and operational data.

This YANG module supports configuration of fixed size and fixed rate packets, and elements that may be augmented to support future configuration. The protocol specification [I-D.ietf-ipsecme-iptfs], goes beyond this simple fixed mode of operation by defining a general format for any type of scheme. In this document the outer IPsec packets can be sent with fixed or variable size (without padding). The configuration allows the fixed packet size to be determined by the path MTU. The fixed packet size can also be configured if a value lower than the path MTU is desired.

Other configuration items include:

- * Congestion Control. A congestion control setting to allow IP-TFS to reduce the packet rate when congestion is detected.
- * Fixed Rate configuration. The IP-TFS tunnel rate can be configured taking into account either layer 2 overhead or layer 3 overhead. Layer 3 overhead is the IP data rate and layer 2 overhead is the rate of bits on the link. The combination of packet size and rate determines the nominal maximum bandwidth and the transmission interval when fixed size packets are used.
- * User packet Fragmentation Control. While fragmentation is recommended for improved efficiency, a configuration is provided if users wish to observe the effect no-fragmentation on their data flows.

The YANG operational data allows the readout of the configured parameters as well as the per SA statistics and error counters for IP-TFS. Per SA IPsec packet statistics are provided as a feature and per SA IP-TFS specific statistics as another feature. Both sets of statistics augment the IPsec YANG models with counters that allow observation of IP-TFS packet efficiency.

RFC [RFC9061] has a set of IPsec YANG management objects. IP-TFS YANG augments the IKE and the IKELESS models. In these models the Security Policy database entry and Security Association entry for an IPsec Tunnel can be augmented with IP-TFS.

3. YANG Management

3.1. YANG Tree

The following is the YANG tree diagram ([RFC8340]) for the IP-TFS extensions.

```

module: ietf-ipsec-iptfs
  augment /nsfike:ipsec-ike/nsfike:conn-entry/nsfike:spd
    /nsfike:spd-entry/nsfike:ipsec-policy-config
    /nsfike:processing-info/nsfike:ipsec-sa-cfg:
    +--rw traffic-flow-security
      +--rw congestion-control?                boolean
      +--rw packet-size
        |   +--rw use-path-mtu-discovery?    boolean
        |   +--rw outer-packet-size?         uint16
      +--rw (tunnel-rate)?
        |   +--:(l2-fixed-rate)
        |   |   +--rw l2-fixed-rate?         yang:counter64
        |   +--:(l3-fixed-rate)
        |   |   +--rw l3-fixed-rate?         yang:counter64
      +--rw dont-fragment?                    boolean
      +--rw max-aggregation-time?             decimal64
      +--rw window-size?                     uint16
      +--rw send-immediately?                boolean
      +--rw lost-packet-timer-interval?      decimal64
  augment /nsfike:ipsec-ike/nsfike:conn-entry/nsfike:child-sa-info:
    +--ro traffic-flow-security
      +--ro congestion-control?              boolean
      +--ro packet-size
        |   +--ro use-path-mtu-discovery?    boolean
        |   +--ro outer-packet-size?         uint16
      +--ro (tunnel-rate)?
        |   +--:(l2-fixed-rate)
        |   |   +--ro l2-fixed-rate?         yang:counter64
        |   +--:(l3-fixed-rate)
        |   |   +--ro l3-fixed-rate?         yang:counter64
      +--ro dont-fragment?                  boolean
      +--ro max-aggregation-time?           decimal64
      +--ro window-size?                   uint16
      +--ro send-immediately?              boolean
      +--ro lost-packet-timer-interval?    decimal64
  augment /nsfikels:ipsec-ikeless/nsfikels:spd/nsfikels:spd-entry
    /nsfikels:ipsec-policy-config/nsfikels:processing-info
    /nsfikels:ipsec-sa-cfg:
    +--rw traffic-flow-security
      +--rw congestion-control?              boolean
      +--rw packet-size

```

```

    | +--rw use-path-mtu-discovery?    boolean
    | +--rw outer-packet-size?         uint16
+--rw (tunnel-rate)?
    | +--:(l2-fixed-rate)
    | | +--rw l2-fixed-rate?          yang:counter64
    | +--:(l3-fixed-rate)
    | | +--rw l3-fixed-rate?          yang:counter64
+--rw dont-fragment?                  boolean
+--rw max-aggregation-time?           decimal64
+--rw window-size?                    uint16
+--rw send-immediately?               boolean
+--rw lost-packet-timer-interval?     decimal64
augment /nsfikels:ipsec-ikeless/nsfikels:sad/nsfikels:sad-entry:
+--ro traffic-flow-security
+--ro congestion-control?             boolean
+--ro packet-size
    | +--ro use-path-mtu-discovery?    boolean
    | +--ro outer-packet-size?         uint16
+--ro (tunnel-rate)?
    | +--:(l2-fixed-rate)
    | | +--ro l2-fixed-rate?          yang:counter64
    | +--:(l3-fixed-rate)
    | | +--ro l3-fixed-rate?          yang:counter64
+--ro dont-fragment?                  boolean
+--ro max-aggregation-time?           decimal64
+--ro window-size?                    uint16
+--ro send-immediately?               boolean
+--ro lost-packet-timer-interval?     decimal64
augment /nsfike:ipsec-ike/nsfike:conn-entry/nsfike:child-sa-info:
+--ro ipsec-stats {ipsec-stats}?
    | +--ro tx-pkts?                  yang:counter64
    | +--ro tx-octets?                yang:counter64
    | +--ro tx-drop-pkts?             yang:counter64
    | +--ro rx-pkts?                  yang:counter64
    | +--ro rx-octets?                yang:counter64
    | +--ro rx-drop-pkts?             yang:counter64
+--ro iptfs-inner-pkt-stats {iptfs-stats}?
    | +--ro tx-pkts?                  yang:counter64
    | +--ro tx-octets?                yang:counter64
    | +--ro rx-pkts?                  yang:counter64
    | +--ro rx-octets?                yang:counter64
    | +--ro rx-incomplete-pkts?       yang:counter64
+--ro iptfs-outer-pkt-stats {iptfs-stats}?
    | +--ro tx-all-pad-pkts?          yang:counter64
    | +--ro tx-all-pad-octets?        yang:counter64
    | +--ro tx-extra-pad-pkts?         yang:counter64
    | +--ro tx-extra-pad-octets?       yang:counter64
    | +--ro rx-all-pad-pkts?          yang:counter64

```



```

    +--ro rx-all-pad-octets?      yang:counter64
    +--ro rx-extra-pad-pkts?      yang:counter64
    +--ro rx-extra-pad-octets?    yang:counter64
    +--ro rx-errored-pkts?        yang:counter64
    +--ro rx-missed-pkts?         yang:counter64
augment /nsfikels:ipsec-ikeless/nsfikels:sad/nsfikels:sad-entry:
+--rw ipsec-stats {ipsec-stats}?
|
|   +--ro tx-pkts?                yang:counter64
|   +--ro tx-octets?              yang:counter64
|   +--ro tx-drop-pkts?           yang:counter64
|   +--ro rx-pkts?                yang:counter64
|   +--ro rx-octets?              yang:counter64
|   +--ro rx-drop-pkts?           yang:counter64
+--ro iptfs-inner-pkt-stats {iptfs-stats}?
|
|   +--ro tx-pkts?                yang:counter64
|   +--ro tx-octets?              yang:counter64
|   +--ro rx-pkts?                yang:counter64
|   +--ro rx-octets?              yang:counter64
|   +--ro rx-incomplete-pkts?     yang:counter64
+--ro iptfs-outer-pkt-stats {iptfs-stats}?
|
|   +--ro tx-all-pad-pkts?        yang:counter64
|   +--ro tx-all-pad-octets?      yang:counter64
|   +--ro tx-extra-pad-pkts?       yang:counter64
|   +--ro tx-extra-pad-octets?     yang:counter64
|   +--ro rx-all-pad-pkts?        yang:counter64
|   +--ro rx-all-pad-octets?      yang:counter64
|   +--ro rx-extra-pad-pkts?       yang:counter64
|   +--ro rx-extra-pad-octets?     yang:counter64
|   +--ro rx-errored-pkts?         yang:counter64
|   +--ro rx-missed-pkts?          yang:counter64

```

3.2. YANG Module

The following is the YANG module for managing the IP-TFS extensions. The model contains references to [I-D.ietf-ipsecme-iptfs] and [RFC5348].

```

<CODE BEGINS> file "ietf-ipsec-iptfs@2021-11-18.yang"
module ietf-ipsec-iptfs {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-ipsec-iptfs";
  prefix iptfs;

  import ietf-i2nsf-ike {
    prefix nsfike;
  }
  import ietf-i2nsf-ikeless {
    prefix nsfikels;
  }

```

```
}
import ietf-yang-types {
  prefix yang;
}

organization
  "IETF IPSECME Working Group (IPSECME)";
contact
  "WG Web: <https://tools.ietf.org/wg/ipsecme/>
  WG List: <mailto:ipsecme@ietf.org>

  Author: Don Fedyk
         <mailto:dfedyk@labn.net>

  Author: Christian Hopps
         <mailto:chopps@chopps.org>";

// RFC Ed.: replace XXXX with actual RFC number and
// remove this note.

description
  "This module defines the configuration and operational state for
  managing the IP Traffic Flow Security functionality [RFC XXXX].

  Copyright (c) 2021 IETF Trust and the persons identified as
  authors of the code. All rights reserved.

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject to
  the license terms contained in, the Simplified BSD License set
  forth in Section 4.c of the IETF Trust's Legal Provisions
  Relating to IETF Documents
  (https://trustee.ietf.org/license-info).

  This version of this YANG module is part of RFC XXXX
  (https://tools.ietf.org/html/rfcXXXX); see the RFC itself for
  full legal notices.";

revision 2021-11-18 {
  description
    "Initial Revision";
  reference
    "RFC XXXX: IP Traffic Flow Security YANG Module";
}

feature ipsec-stats {
  description
    "This feature indicates the device supports
```

```
        per SA IPsec statistics";
    }

    feature iptfs-stats {
        description
            "This feature indicates the device supports
            per SA IP Traffic Flow Security statistics";
    }

    /*-----*/
    /* groupings */
    /*-----*/

    grouping ipsec-tx-stat-grouping {
        description
            "IPsec outbound statistics";
        leaf tx-pkts {
            type yang:counter64;
            config false;
            description
                "Outbound Packet count";
        }
        leaf tx-octets {
            type yang:counter64;
            config false;
            description
                "Outbound Packet bytes";
        }
        leaf tx-drop-pkts {
            type yang:counter64;
            config false;
            description
                "Outbound dropped packets count";
        }
    }

    grouping ipsec-rx-stat-grouping {
        description
            "IPsec inbound statistics";
        leaf rx-pkts {
            type yang:counter64;
            config false;
            description
                "Inbound Packet count";
        }
        leaf rx-octets {
            type yang:counter64;
            config false;
        }
    }
}
```

```
        description
            "Inbound Packet bytes";
    }
    leaf rx-drop-pkts {
        type yang:counter64;
        config false;
        description
            "Inbound dropped packets count";
    }
}

grouping iptfs-inner-tx-stat-grouping {
    description
        "IP-TFS outbound inner packet statistics";
    leaf tx-pkts {
        type yang:counter64;
        config false;
        description
            "Total number of IP-TFS inner packets sent. This
            count is whole packets only. A fragmented packet
            counts as one packet";
        reference
            "draft-ietf-ipsecme-iptfs";
    }
    leaf tx-octets {
        type yang:counter64;
        config false;
        description
            "Total number of IP-TFS inner octets sent. This is
            inner packet octets only. Does not count padding.";
        reference
            "draft-ietf-ipsecme-iptfs";
    }
}

grouping iptfs-outer-tx-stat-grouping {
    description
        "IP-TFS outbound inner packet statistics";
    leaf tx-all-pad-pkts {
        type yang:counter64;
        config false;
        description
            "Total number of transmitted IP-TFS packets that
            were all padding with no inner packet data.";
        reference
            "draft-ietf-ipsecme-iptfs section 2.2.3";
    }
    leaf tx-all-pad-octets {
```

```
    type yang:counter64;
    config false;
    description
      "Total number transmitted octets of padding added to
      IP-TFS packets with no inner packet data.";
    reference
      "draft-ietf-ipsecme-iptfs section 2.2.3";
  }
  leaf tx-extra-pad-pkts {
    type yang:counter64;
    config false;
    description
      "Total number of transmitted outer IP-TFS packets
      that included some padding.";
    reference
      "draft-ietf-ipsecme-iptfs section 2.2.3.1";
  }
  leaf tx-extra-pad-octets {
    type yang:counter64;
    config false;
    description
      "Total number of transmitted octets of padding added
      to outer IP-TFS packets with data.";
    reference
      "draft-ietf-ipsecme-iptfs section 2.2.3.1";
  }
}

grouping iptfs-inner-rx-stat-grouping {
  description
    "IP-TFS inner packet inbound statistics";
  leaf rx-pkts {
    type yang:counter64;
    config false;
    description
      "Total number of IP-TFS inner packets received.";
    reference
      "draft-ietf-ipsecme-iptfs section 2.2";
  }
  leaf rx-octets {
    type yang:counter64;
    config false;
    description
      "Total number of IP-TFS inner octets received. Does
      not include padding or overhead";
    reference
      "draft-ietf-ipsecme-iptfs section 2.2";
  }
}
```

```
    leaf rx-incomplete-pkts {
      type yang:counter64;
      config false;
      description
        "Total number of IP-TFS inner packets that were
        incomplete. Usually this is due to fragments not
        received. Also, this may be due to misordering or
        errors in received outer packets.";
      reference
        "draft-ietf-ipsecme-iptfs";
    }
  }

  grouping iptfs-outer-rx-stat-grouping {
    description
      "IP-TFS outer packet inbound statistics";
    leaf rx-all-pad-pkts {
      type yang:counter64;
      config false;
      description
        "Total number of received IP-TFS packets that were
        all padding with no inner packet data.";
      reference
        "draft-ietf-ipsecme-iptfs section 2.2.3";
    }
    leaf rx-all-pad-octets {
      type yang:counter64;
      config false;
      description
        "Total number received octets of padding added to
        IP-TFS packets with no inner packet data.";
      reference
        "draft-ietf-ipsecme-iptfs section 2.2.3";
    }
    leaf rx-extra-pad-pkts {
      type yang:counter64;
      config false;
      description
        "Total number of received outer IP-TFS packets that
        included some padding.";
      reference
        "draft-ietf-ipsecme-iptfs section 2.2.3.1";
    }
    leaf rx-extra-pad-octets {
      type yang:counter64;
      config false;
      description
        "Total number of received octets of padding added to
```

```
        outer IP-TFS packets with data.";
    reference
        "draft-ietf-ipsecme-iptfs section 2.2.3.1";
}
leaf rx-errored-pkts {
    type yang:counter64;
    config false;
    description
        "Total number of IP-TFS outer packets dropped due to
        errors.";
    reference
        "draft-ietf-ipsecme-iptfs";
}
leaf rx-missed-pkts {
    type yang:counter64;
    config false;
    description
        "Total number of IP-TFS outer packets missing
        indicated by missing sequence number.";
    reference
        "draft-ietf-ipsecme-iptfs";
}
}

grouping iptfs-config {
    description
        "This is the grouping for iptfs configuration";
    container traffic-flow-security {
        description
            "Configure the IPSec TFS in Security
            Association Database (SAD)";
        leaf congestion-control {
            type boolean;
            default "true";
            description
                "When set to true, the default, this enables the
                congestion control on-the-wire exchange of data that is
                required by congestion control algorithms as defined by
                RFC 5348. When set to false, IP-TFS sends fixed-sized
                packets over an IP-TFS tunnel at a constant rate.";
            reference
                "draft-ietf-ipsecme-iptfs section 2.5.2, RFC 5348";
        }
        container packet-size {
            description
                "Packet size is either auto-discovered or manually
                configured.";
            leaf use-path-mtu-discovery {
```

```
    type boolean;
    default "true";
    description
      "Utilize path mtu discovery to determine maximum
       IP-TFS packet size. If the packet size is explicitly
       configured, then it will only be adjusted downward if
       use-path-mtu-discovery is set.";
    reference
      "draft-ietf-ipsecme-iptfs section 4.2";
  }
  leaf outer-packet-size {
    type uint16;
    description
      "On transmission, the size of the outer encapsulating
       tunnel packet (i.e., the IP packet containing the ESP
       payload).";
    reference
      "draft-ietf-ipsecme-iptfs section 4.2";
  }
}
choice tunnel-rate {
  description
    "TFS bit rate may be specified at layer 2 wire
     rate or layer 3 packet rate";
  leaf l2-fixed-rate {
    type yang:counter64;
    description
      "On transmission, target bandwidth/bit rate in bps
       for iptfs tunnel. This fixed rate is the nominal
       timing for the fixed size packet. If congestion
       control is enabled the rate may be adjusted down (or
       up if unset).";
    reference
      "draft-ietf-ipsecme-iptfs section 4.1";
  }
  leaf l3-fixed-rate {
    type yang:counter64;
    description
      "On transmission, target bandwidth/bit rate in bps
       for iptfs tunnel. This fixed rate is the nominal
       timing for the fixed size packet. If congestion
       control is enabled the rate may be adjusted down (or
       up if unset).";
    reference
      "draft-ietf-ipsecme-iptfs section 4.1";
  }
}
leaf dont-fragment {
```



```
    type boolean;
    default "false";
    description
      "On transmission, disable packet fragmentation across
       consecutive iptfs tunnel packets; inner packets larger
       than what can be transmitted in outer packets will be
       dropped.";
    reference
      "draft-ietf-ipsecme-iptfs section 2.2.4 and 6.4.1";
  }
  leaf max-aggregation-time {
    type decimal64 {
      fraction-digits 6;
    }
    units "milliseconds";
    description
      "On transmission, maximum aggregation time is the
       maximum length of time a received inner packet can be
       held prior to transmission in the iptfs tunnel. Inner
       packets that would be held longer than this time, based
       on the current tunnel configuration will be dropped
       rather than be queued for transmission. Maximum
       aggregation time is configurable in milliseconds or
       fractional milliseconds down to 1 nanosecond.";
  }
  leaf window-size {
    type uint16 {
      range "0..65535";
    }
    description
      "On reception, the maximum number of out-of-order
       packets that will be reordered by an iptfs receiver
       while performing the reordering operation. The value 0
       disables any reordering.";
    reference
      "draft-ietf-ipsecme-iptfs section 2.2.3";
  }
  leaf send-immediately {
    type boolean;
    default false;
    description
      "On reception, send inner packets as soon as possible, do
       not wait for lost or misordered outer packets.
       Selecting this option reduces the inner (user) packet
       delay but can amplify out-of-order delivery of the
       inner packet stream in the presence of packet
       aggregation and any reordering.";
    reference
```

```
        "draft-ietf-ipsecme-iptfs section 2.5";
    }
    leaf lost-packet-timer-interval {
        type decimal64 {
            fraction-digits 6;
        }
        units "milliseconds";
        description
            "On reception, this interval defines the length of time
            an iptfs receiver will wait for a missing packet before
            considering it lost. If not using send-immediately,
            then each lost packet will delay inner (user) packets
            until this timer expires. Setting this value too low
            can impact reordering and reassembly. The value is
            configurable in milliseconds or fractional milliseconds
            down to 1 nanosecond.";
        reference
            "draft-ietf-ipsecme-iptfs section 2.2.3";
    }
}

/*
 * IP-TFS ike configuration
 */

augment "/nsfike:ipsec-ike/nsfike:conn-entry/nsfike:spd/"
    + "nsfike:spd-entry/"
    + "nsfike:ipsec-policy-config/"
    + "nsfike:processing-info/"
    + "nsfike:ipsec-sa-cfg" {
    description
        "IP-TFS configuration for this policy.";
    uses iptfs-config;
}

augment "/nsfike:ipsec-ike/nsfike:conn-entry/"
    + "nsfike:child-sa-info" {
    description
        "IP-TFS configured on this SA.";
    uses iptfs-config {
        refine "traffic-flow-security" {
            config false;
        }
    }
}

/*
```

```
* IP-TFS ikeless configuration
*/

augment "/nsfikels:ipsec-ikeless/nsfikels:spd/"
  + "nsfikels:spd-entry/"
  + "nsfikels:ipsec-policy-config/"
  + "nsfikels:processing-info/"
  + "nsfikels:ipsec-sa-cfg" {
  description
    "IP-TFS configuration for this policy.";
  uses iptfs-config;
}

augment "/nsfikels:ipsec-ikeless/nsfikels:sad/"
  + "nsfikels:sad-entry" {
  description
    "IP-TFS configured on this SA.";
  uses iptfs-config {
    refine "traffic-flow-security" {
      config false;
    }
  }
}

/*
* packet counters
*/

augment "/nsfike:ipsec-ike/nsfike:conn-entry/"
  + "nsfike:child-sa-info" {
  description
    "Per SA Counters";
  container ipsec-stats {
    if-feature "ipsec-stats";
    config false;
    description
      "IPsec per SA packet counters.";
    uses ipsec-tx-stat-grouping {
      //when "direction = 'outbound'";
    }
    uses ipsec-rx-stat-grouping {
      //when "direction = 'inbound'";
    }
  }
  container iptfs-inner-pkt-stats {
    if-feature "iptfs-stats";
    config false;
    description
```

```
        "IPTFS per SA inner packet counters.";
    uses iptfs-inner-tx-stat-grouping {
        //when "direction = 'outbound'";
    }
    uses iptfs-inner-rx-stat-grouping {
        //when "direction = 'inbound'";
    }
}
container iptfs-outer-pkt-stats {
    if-feature "iptfs-stats";
    config false;
    description
        "IPTFS per SA outer packets counters.";
    uses iptfs-outer-tx-stat-grouping {
        //when "direction = 'outbound'";
    }
    uses iptfs-outer-rx-stat-grouping {
        //when "direction = 'inbound'";
    }
}
}

/*
 * packet counters
 */

augment "/nsfikels:ipsec-ikeless/nsfikels:sad/"
    + "nsfikels:sad-entry" {
    description
        "Per SA Counters";
    container ipsec-stats {
        if-feature "ipsec-stats";
        description
            "IPsec per SA packet counters.";
        uses ipsec-tx-stat-grouping {
            //when "direction = 'outbound'";
        }
        uses ipsec-rx-stat-grouping {
            //when "direction = 'inbound'";
        }
    }
}
container iptfs-inner-pkt-stats {
    if-feature "iptfs-stats";
    config false;
    description
        "IPTFS per SA inner packet counters.";
    uses iptfs-inner-tx-stat-grouping {
        //when "direction = 'outbound'";
    }
}
```

```
    }
    uses iptfs-inner-rx-stat-grouping {
      //when "direction = 'inbound'";
    }
  }
  container iptfs-outer-pkt-stats {
    if-feature "iptfs-stats";
    config false;
    description
      "IPTFS per SA outer packets counters.";
    uses iptfs-outer-tx-stat-grouping {
      //when "direction = 'outbound'";
    }
    uses iptfs-outer-rx-stat-grouping {
      //when "direction = 'inbound'";
    }
  }
}
}
<CODE ENDS>
```

4. IANA Considerations

4.1. Updates to the IETF XML Registry

This document registers a URI in the "IETF XML Registry" [RFC3688]. Following the format in [RFC3688], the following registration has been made:

URI:
urn:ietf:params:xml:ns:yang:ietf-ipsec-iptfs

Registrant Contact:
The IESG.

XML:
N/A; the requested URI is an XML namespace.

4.2. Updates to the YANG Module Names Registry

This document registers one YANG module in the "YANG Module Names" registry [RFC6020]. Following the format in [RFC6020], the following registration has been made:

name:
ietf-ipsec-iptfs

```
namespace:
  urn:ietf:params:xml:ns:yang:ietf-ipsec-iptfs

prefix:
  iptfs

reference:
  RFC XXXX (RFC Ed.: replace XXXX with actual RFC number and remove
  this note.)
```

5. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

The YANG module defined in this document can enable, disable and modify the behavior of IP traffic flow security, for the implications regarding these types of changes consult the [I-D.ietf-ipsecme-iptfs] which defines the functionality.

IP-TFS hides the traffic flows through the network, anywhere that access YANG statistics is enabled needs to be protected from third party observation.

6. Acknowledgements

The authors would like to thank Eric Kinzie, Juergen Schoenwaelder, Lou Berger and Tero Kivinen for their feedback and review on the YANG model.

7. References

7.1. Normative References

- [I-D.ietf-ipsecme-iptfs]
Hopps, C., "IP-TFS: Aggregation and Fragmentation Mode for ESP and its Use for IP Traffic Flow Security", Work in Progress, Internet-Draft, draft-ietf-ipsecme-iptfs-12, 8 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-ipsecme-iptfs-12.txt>>.
- [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>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [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>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.
- [RFC9061] Marin-Lopez, R., Lopez-Millan, G., and F. Pereniguez-Garcia, "A YANG Data Model for IPsec Flow Protection Based on Software-Defined Networking (SDN)", RFC 9061, DOI 10.17487/RFC9061, July 2021, <<https://www.rfc-editor.org/info/rfc9061>>.

7.2. Informative References

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.

- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<https://www.rfc-editor.org/info/rfc5348>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

Appendix A. Examples

The following examples show configuration and operational data for the ikeless and ike cases using xml and json. Also, the operational statistics for the ikeless case is illustrated.

A.1. Example XML Configuration

This example illustrates configuration for IP-TFS in the ikeless case. Note that since this augments the ipsec ikeless schema only minimal a ikeless configuration to satisfy the schema has been populated.


```
<i:ipsec-ikeless
  xmlns:i="urn:ietf:params:xml:ns:yang:ietf-i2nsf-ikeless"
  xmlns:tfs="urn:ietf:params:xml:ns:yang:ietf-ipsec-iptfs">
  <i:spd>
    <i:spd-entry>
      <i:name>protect-policy-1</i:name>
      <i:direction>outbound</i:direction>
      <i:ipsec-policy-config>
        <i:traffic-selector>
          <i:local-prefix>192.0.2.0/16</i:local-prefix>
          <i:remote-prefix>198.51.100.0/16</i:remote-prefix>
        </i:traffic-selector>
        <i:processing-info>
          <i:action>protect</i:action>
          <i:ipsec-sa-cfg>
            <tfs:traffic-flow-security>
              <tfs:congestion-control>true</tfs:congestion-control>
              <tfs:packet-size>
                <tfs:use-path-mtu-discovery>
                  >true</tfs:use-path-mtu-discovery>
                </tfs:packet-size>
              <tfs:l2-fixed-rate>10000000000</tfs:l2-fixed-rate>
              <tfs:max-aggregation-time>
                >0.1</tfs:max-aggregation-time>
              <tfs>window-size>5</tfs>window-size>
              <tfs:send-immediately>false</tfs:send-immediately>
              <tfs:lost-packet-timer-interval>
                >0.2</tfs:lost-packet-timer-interval>
            </tfs:traffic-flow-security>
          </i:ipsec-sa-cfg>
        </i:processing-info>
      </i:ipsec-policy-config>
    </i:spd-entry>
  </i:spd>
</i:ipsec-ikeless>
```

Figure 1: Example IP-TFS XML configuration

A.2. Example XML Operational Data

This example illustrates operational data for IP-TFS in the `ikeless` case. Note that since this augments the `ipsec ikeless` schema only minimal `ikeless` configuration to satisfy the schema has been populated.

```

<i:ipsec-ikeless
  xmlns:i="urn:ietf:params:xml:ns:yang:ietf-i2nsf-ikeless"
  xmlns:tfs="urn:ietf:params:xml:ns:yang:ietf-ipsec-iptfs">
  <i:sad>
    <i:sad-entry>
      <i:name>sad-1</i:name>
      <i:ipsec-sa-config>
        <i:spi>1</i:spi>
        <i:traffic-selector>
          <i:local-prefix>2001:DB8::0/16</i:local-prefix>
          <i:remote-prefix>2001:DB8::1:0/16</i:remote-prefix>
        </i:traffic-selector>
      </i:ipsec-sa-config>
      <tfs:traffic-flow-security>
        <tfs:congestion-control>true</tfs:congestion-control>
        <tfs:packet-size>
          <tfs:use-path-mtu-discovery>
            >true</tfs:use-path-mtu-discovery>
          </tfs:packet-size>
        <tfs:l2-fixed-rate>1000000000</tfs:l2-fixed-rate>
        <tfs:max-aggregation-time>0.100</tfs:max-aggregation-time>
        <tfs>window-size>0</tfs>window-size>
        <tfs:send-immediately>true</tfs:send-immediately>
        <tfs:lost-packet-timer-interval>
          >0.200</tfs:lost-packet-timer-interval>
        </tfs:traffic-flow-security>
      </i:sad-entry>
    </i:sad>
  </i:ipsec-ikeless>

```

Figure 2: Example IP-TFS XML Operational data

A.3. Example JSON Configuration

This example illustrates config data for IP-TFS in the ike case. Note that since this augments the ipsec ike schema only minimal ike configuration to satisfy the schema has been populated.

```

{
  "ietf-i2nsf-ike:ipsec-ike": {
    "ietf-i2nsf-ike:conn-entry": [
      {
        "name": "my-peer-connection",
        "ike-sa-encr-alg": [
          {
            "id": 1,
            "algorithm-type": 12,
            "key-length": 128
          }
        ]
      }
    ]
  }
}

```

```

    }
  ],
  "local": {
    "local-pad-entry-name": "local-1"
  },
  "remote": {
    "remote-pad-entry-name": "remote-1"
  },
  "ietf-i2nsf-ike:spd": {
    "spd-entry": [
      {
        "name": "protect-policy-1",
        "ipsec-policy-config": {
          "traffic-selector": {
            "local-prefix": "192.0.2.0/16",
            "remote-prefix": "198.51.100.0/16"
          },
          "processing-info": {
            "action": "protect",
            "ipsec-sa-cfg": {
              "ietf-ipsec-iptfs:traffic-flow-security": {
                "congestion-control": "true",
                "l2-fixed-rate": 10000000000,
                "packet-size": {
                  "use-path-mtu-discovery": "true"
                },
                "max-aggregation-time": "0.1",
                "window-size": "1",
                "send-immediately": "false",
                "lost-packet-timer-interval": "0.2"
              }
            }
          }
        }
      }
    ]
  }
}

```

Figure 3: Example IP-TFS JSON configuration

A.4. Example JSON Operational Data

This example illustrates operational data for IP-TFS in the ike case. Note that since this augments the ipsec ike tree only minimal ike configuration to satisfy the schema has been populated.

```
{
  "ietf-i2nsf-ike:ipsec-ike": {
    "ietf-i2nsf-ike:conn-entry": [
      {
        "name": "my-peer-connection",
        "ike-sa-encr-alg": [
          {
            "id": 1,
            "algorithm-type": 12,
            "key-length": 128
          }
        ],
        "local": {
          "local-pad-entry-name": "local-1"
        },
        "remote": {
          "remote-pad-entry-name": "remote-1"
        },
        "ietf-i2nsf-ike:child-sa-info": {
          "ietf-ipsec-iptfs:traffic-flow-security": {
            "congestion-control": "true",
            "l2-fixed-rate": 10000000000,
            "packet-size": {
              "use-path-mtu-discovery": "true"
            },
            "max-aggregation-time": "0.1",
            "window-size": "5",
            "send-immediately": "false",
            "lost-packet-timer-interval": "0.2"
          }
        }
      }
    ]
  }
}
```

Figure 4: Example IP-TFS JSON Operational data

A.5. Example JSON Operational Statistics

This example shows the json formatted statistics for IP-TFS. Note a unidirectional IP-TFS transmit side is illustrated, with arbitrary numbers for transmit.

```
{
  "ietf-i2nsf-ikeless:ipsec-ikeless": {
    "sad": {
      "sad-entry": [
        {
          "name": "sad-1",
          "ipsec-sa-config": {
            "spi": 1,
            "traffic-selector": {
              "local-prefix": "192.0.2.1/16",
              "remote-prefix": "198.51.100.0/16"
            }
          },
          "ietf-ipsec-iptfs:traffic-flow-security": {
            "window-size": "5",
            "send-immediately": "false",
            "lost-packet-timer-interval": "0.2"
          },
          "ietf-ipsec-iptfs:ipsec-stats": {
            "tx-pkts": "300",
            "tx-octets": "80000",
            "tx-drop-pkts": "2",
            "rx-pkts": "0",
            "rx-octets": "0",
            "rx-drop-pkts": "0"
          },
          "ietf-ipsec-iptfs:iptfs-inner-pkt-stats": {
            "tx-pkts": "250",
            "tx-octets": "75000",
            "rx-pkts": "0",
            "rx-octets": "0",
            "rx-incomplete-pkts": "0"
          },
          "ietf-ipsec-iptfs:iptfs-outer-pkt-stats": {
            "tx-all-pad-pkts": "40",
            "tx-all-pad-octets": "40000",
            "tx-extra-pad-pkts": "200",
            "tx-extra-pad-octets": "30000",
            "rx-all-pad-pkts": "0",
            "rx-all-pad-octets": "0",
            "rx-extra-pad-pkts": "0",
            "rx-extra-pad-octets": "0",

```

```
        "rx-errored-pkts": "0",
        "rx-missed-pkts": "0"
      },
      "ipsec-sa-state": {
        "sa-lifetime-current": {
          "time": 80000,
          "bytes": 4000606,
          "packets": 1000,
          "idle": 5
        }
      }
    }
  ]
}
```

Figure 5: Example IP-TFS JSON Statistics

```
<tfs:traffic-flow-security> <tfs:reorder-window-
size>300</tfs:reorder-window-size>
```

Authors' Addresses

Don Fedyk
LabN Consulting, L.L.C.

Email: dfedyk@labn.net

Christian Hopps
LabN Consulting, L.L.C.

Email: chopps@chopps.org

IPSECME Working Group
Internet-Draft
Intended status: Standards Track
Expires: 14 August 2022

S. Kompoti
W. Pan
Huawei
P. Wouters
Aiven
M. Bharath
Mavenir
M. Chen
CMCC
M. Richardson, Ed.
Sandelman Software Works
10 February 2022

IKEv2 Optional SA&TS Payloads in Child Exchange
draft-kompoti-ipsecme-ikev2-sa-ts-payloads-opt-08

Abstract

This document describes a method for reducing the size of the Internet Key Exchange version 2 (IKEv2) CREATE_CHILD_SA exchanges used for rekeying of the IKE or Child SA by replacing the SA and TS payloads with a Notify Message payload. Reducing size and complexity of IKEv2 exchanges is especially useful for low power consumption battery powered devices.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at
<https://datatracker.ietf.org/doc/draft-kompoti-ipsecme-ikev2-sa-ts-payloads-opt/>.

Discussion of this document takes place on the ipsec Working Group mailing list (<mailto:ipsec@ietf.org>), which is archived at <https://mailarchive.ietf.org/arch/browse/ipsec/>.

Source for this draft and an issue tracker can be found at <https://github.com/mcr/ipsecme-ikev2-sa-ts-payloads.git>.

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 14 August 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
2. Conventions Used in This Document	4
2.1. Requirements Language	4
3. Negotiation of Support for OPTIMIZED REKEY	4
4. Optimized Rekey of the IKE SA	5
5. Optimized Rekey of Child SAs	5
6. Payload Formats	6
6.1. OPTIMIZED_REKEY_SUPPORTED Notify	6
6.2. OPTIMIZED_REKEY Notify	7
7. IANA Considerations	7
8. Operational Considerations	8
9. Security Considerations	8
10. Acknowledgments	8
11. Normative References	8
Authors' Addresses	8

1. Introduction

The Internet Key Exchange protocol version 2 (IKEv2) [RFC7296] is used to negotiate Security Association (SA) parameters for the IKE SA and the Child SAs. Cryptographic key material for these SAs have a limited lifetime before it needs to be refreshed, a process referred to as "rekeying". IKEv2 uses the CREATE_CHILD_SA exchange to rekey either the IKE SA or the Child SAs.

When rekeying, a full set of negotiation parameters are exchanged. However, most of these parameters will be the same as before, and some of these parameters MUST not change.

For example, the Traffic Selector (TS) negotiated for the new Child SA MUST cover the Traffic Selectors negotiated for the old Child SA. And in practically all cases, a new Child SA does not need to cover a wider set of Traffic. In the rare case where this would be needed, either a standard rekey could be used or a new Child SA could be negotiated followed by a deletion of the replaced Child SA.

Similarly, IKEv2 states that the cryptographic parameters negotiated for rekeying SHOULD NOT be different. This means that the security properties of the IKE or Child SA in practise do not change during a typical rekey.

This document specifies a method to omit these parameters and replace them with a single Notify Message declaring that all these parameters are identical to the originally negotiated parameters.

Large scale IKEv2 gateways such as Evolved Packet Data Gateway (ePDG) in 4G networks or Centralized Radio Access Network (cRAN/Cloud) gateways in 5G networks typically support more than 100,000 IKE/IPsec connections. At any point in time, there will be hundreds or thousands of IKE SAs and Child SAs that are being rekeyed. This takes a large amount of bandwidth and CPU power and any protocol simplification or bandwidth reducing would result in a significant resource saving.

For Internet of Things (IoT) devices which utilize low power consumption technology, reducing the size of the CREATE_CHILD_SA exchange for rekeying reduces its power consumption, as sending bytes over the air is usually the most power consuming operation of such a device. Reducing the CPU operations required to verify the rekey exchanges parameters will also save power and extend the lifetime for these devices.

When using identical parameters for the IKE SA or Child SA rekey, the SA and TS payloads can be omitted thanks to the optimization defined in this document. For an IKE SA rekey, instead of the (large) SA payload, only a Key Exchange (KE) payload and a new Notify Type payload with the new SPI are required. For a Child SA payload, instead of the SA or TS payloads, only an optional nonce payload (when using PFS) and a new Notify Type payload with the new SPI are needed. This makes the rekey exchange packets much smaller and the peers do not need to verify that the SA or TS parameters are compatible with the old SA parameters.

2. Conventions Used in This Document

2.1. 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. Negotiation of Support for OPTIMIZED REKEY

To indicate support for the optimized rekey negotiation, the initiator includes the OPTIMIZED_REKEY_SUPPORTED notify payload in the IKE_AUTH exchange request. During this initial key request, the entire SA and TA payloads are included as normal. A responder that supports the optimized rekey exchange includes the OPTIMIZED_REKEY_SUPPORTED notify payload in its response. Note that the notify indicates support for optimized rekey for both IKE and Child SAs.

A responder that does not support the optimized rekey exchange processes the SA and TA payloads as normal, and does not include the new Notify. As per regular IKEv2 processing, a responder that does not recognize this new Notify, MUST ignore the notify. Responders may have been administratively configured with the optimization turned off for local reasons. The absence of the Notify indicates to the initiator that the optimization is not available, and normal, full rekey should be done.

When a peer wishes to rekey an IKE SA or Child SA, it MAY use the optimized rekey method during the CREATE_CHILD_SA exchange. If both peers have exchanged OPTIMIZED_REKEY_SUPPORTED notifies, peers SHOULD use the optimized rekey method for rekeys. Non-optimized, regular rekey requests MUST always be accepted.

The IKE_AUTH message exchange in this case is shown below:

Initiator	Responder
<hr/>	
HDR, SK {IDi, [CERT,] [CERTREQ,] [IDr,] AUTH, SAi2, TSi, TSr, N(OPTIMIZED_REKEY_SUPPORTED)} -->	<-- HDR, SK {IDr, [CERT,] AUTH, SAr2, TSi, TSr, N(OPTIMIZED_REKEY_SUPPORTED)}

4. Optimized Rekey of the IKE SA

The initiator of an optimized rekey request sends a CREATE_CHILD_SA payload with the OPTIMIZED_REKEY notify payload containing the new Security Parameter Index (SPI) for the new IKE SA. It omits the SA payload.

The responder of an optimized rekey request replies with an included OPTIMIZED_REKEY notify with its new IKE SPI and also omits the SA payload.

Both parties send their nonce and KE payloads just as they would do for a regular IKE SA rekey.

Using the old SPI from the IKE header and the two new SPIs respectively from the initiator and responder's OPTIMIZED_REKEY payloads, both parties can perform the IKE SA rekey operation.

The CREATE_CHILD_SA message exchange in this case is shown below:

Initiator	Responder
<hr/>	
HDR, SK {N(OPTIMIZED_REKEY,newSPIi), Ni, KEi} -->	<-- HDR, SK {N(OPTIMIZED_REKEY,newSPIr), Nr, KEr}

5. Optimized Rekey of Child SAs

The initiator of an optimized rekey request sends a CREATE_CHILD_SA payload with the OPTIMIZED_REKEY notify payload containing the new Security Parameter Index (SPI) for the new Child SA. It omits the SA and TS payloads. If the current Child SA was negotiated with Perfect Forward Secrecy (PFS), a KEi payload MUST be included as well. If no PFS was negotiated for the current Child SA, a KEi payload MUST NOT be included.

The responder of an optimized rekey request performs the same process. It includes the OPTIMIZED_REKEY notify with its new IKE SPI and omits the SA and TS payloads. Depending on the PFS negotiation of the current Child SA, the responder includes a KEr payload.

Both parties send their nonce payloads just as they would do for a regular Child SA rekey.

Using the old SPI from the REKEY_SA payload and the two new SPIs respectively from the initiator and responder's OPTIMIZED_REKEY payloads, both parties can perform the Child SA rekey operation.

The CREATE_CHILD_SA message exchange in this case is shown below:

Initiator	Responder

HDR, SK {N(REKEY_SA,oldSPI), N(OPTIMIZED_REKEY,newSPIi), Ni, [KEi,]} -->	<-- HDR, SK {N(OPTIMIZED_REKEY,newSPIr), Nr, [KEr,]}

6. Payload Formats

6.1. OPTIMIZED_REKEY_SUPPORTED Notify

The OPTIMIZED_REKEY_SUPPORTED Notify Message type notification is used by the initiator and responder to indicate their support for the optimized rekey negotiation.

1										2										3											
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Next Payload										C	RESERVED										Payload Length										
Protocol ID(=0)										SPI Size (=0)										Notify Message Type											

* Protocol ID (1 octet) - MUST be 0.

* SPI Size (1 octet) - MUST be 0, meaning no SPI is present.

* Notify Message Type (2 octets) - MUST be set to the value TBD1.

This Notify Message type contains no data.

The Critical bit MUST be 0. A non-zero value MUST be ignored.

6.2. OPTIMIZED_REKEY Notify

The OPTIMIZED_REKEY Notify Message type is used to perform an optimized IKE SA or Child SA rekey.

0									1									2									3										
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1						
+-----+-----+-----+-----+																																					
Next Payload									C	RESERVED								Payload Length																			
+-----+-----+-----+-----+																																					
Protocol ID										SPI Size (=8)								Notify Message Type																			
+-----+-----+-----+-----+																																					
										Security Parameter Index (SPI)																											
+-----+-----+-----+-----+																																					

- * Protocol ID (1 octet) - For an IKE SA rekey, this field MUST contain (1). For Child SAs, this field MUST contain either (2) to indicate AH or (3) to indicate ESP.
- * SPI Size (1 octet) - MUST be 8 when rekeying an IKE SA. MUST be 4 when rekeying a Child SA.
- * Notify Message Type (2 octets) - MUST be set to the value TBD2.
- * SPI (4 octets or 8 octets) - Security Parameter Index. The new SPI.

The Critical bit MUST be 1. A value of 0 MUST be ignored.

7. IANA Considerations

This document defines two new Notify Message Types in the "IKEv2 Notify Message Types - Status Types" registry. IANA is requested to assign codepoints in this registry.

NOTIFY messages: status types	Value
OPTIMIZED_REKEY_SUPPORTED	TBD1
OPTIMIZED_REKEY	TBD2

8. Operational Considerations

Some implementations allow sending rekey messages with a different set of Traffic Selectors or cryptographic parameters in response to a configuration update. IKEv2 states this SHOULD NOT be done. Whether or not optimized rekeying is used, a configuration change that changes the Traffic Selectors or cryptographic parameters MUST NOT use the optimized rekey method. It SHOULD also not use a regular rekey method but instead start an entire new IKE and Child SA negotiation with the new parameters.

9. Security Considerations

The optimized rekey removes sending unnecessary new parameters that originally would have to be validated against the original parameters. In that sense, this optimization enhances the security of the rekey process by reducing the complexity and code required.

10. Acknowledgments

Special thanks to Valery Smyslov and Antony Antony.

11. 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/rfc/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, <<https://www.rfc-editor.org/rfc/rfc7296>>.
- [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/rfc/rfc8174>>.

Authors' Addresses

Sandeep Kampati
Huawei Technologies
Divyashree Techno Park, Whitefield
Bangalore 560066
Karnataka
India

Email: sandeepkampati@huawei.com

Wei Pan
Huawei Technologies
101 Software Avenue, Yuhuatai District
Nanjing
Jiangsu,
China

Email: william.panwei@huawei.com

Paul Wouters
Aiven

Email: paul.wouters@aiven.io

Meduri S S Bharath
Mavenir Systems Pvt Ltd
Manyata Tech Park
Bangalore
Karnataka
India

Email: bharath.meduri@mavenir.com

Meiling Chen
China Mobile
32 Xuanwumen West Street, West District
Beijing
100053
China

Email: chenmeiling@chinamobile.com

Michael Richardson (editor)
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

Network
Internet-Draft
Intended status: Standards Track
Expires: 22 September 2022

A. Antony
secunet
T. Brunner
codelabs
S. Klassert
secunet
P. Wouters
Aiven
21 March 2022

IKEv2 support for per-queue Child SAs
draft-pwouters-ipsecme-multi-sa-performance-03

Abstract

This document defines three Notify Message Type Payloads for the Internet Key Exchange Protocol Version 2 (IKEv2) indicating support for the negotiation of multiple identical Child SAs to optimize performance.

The CPU_QUEUES notification indicates support for multiple queues or CPUs. The CPU_QUEUE_INFO notification is used to confirm and optionally convey information about the specific queue. The TS_MAX_QUEUE notify conveys that the peer is unwilling to create more additional Child SAs for this particular Traffic Selector set.

Using multiple identical Child SAs has the benefit that each stream has its own Sequence Number Counter, ensuring that CPUs don't have to synchronize their crypto state or disable their packet replay protection.

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 22 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Language	3
2. Performance bottlenecks	3
3. Negotiation of CPU specific Child SAs	3
4. Implementation Considerations	5
5. Payload Format	6
5.1. CPU_QUEUES Notify Status Message Payload	6
5.2. CPU_QUEUE_INFO Notify Status Message Payload	7
5.3. TS_MAX_QUEUE Notify Error Message Payload	7
6. Operational Considerations	8
7. Security Considerations	9
8. Implementation Status	9
8.1. Linux XFRM	9
8.2. Libreswan	10
8.3. strongSwan	11
8.4. iproute2	11
9. IANA Considerations	11
10. References	12
10.1. Normative References	12
10.2. Informative References	12
Authors' Addresses	13

1. Introduction

IPsec implementations are currently limited to using one queue or CPU per Child SA. The result is that a machine with many queues/CPU is limited to only using one of these per Child SA. This severely limits the throughput that can be attained. An unencrypted link of 10Gbps or more is commonly reduced to 2-5Gbps when IPsec is used to encrypt the link using AES-GCM. By using the implementation specified in this document, aggregate throughput increased from 5Gbps using 1 CPU to 40-60 Gbps using 25-30 CPUs

While this could be (partially) mitigated by setting up multiple narrowed Child SAs, for example using Populate From Packet (PFP) as specified in [RFC4301], this IPsec feature is not widely implemented. Some route based IPsec implementations might be able to implement this with specific rules into separate network interfaces, but these methods might not be available for policy based IPsec implementations.

To make better use of multiple network queues and CPUs, it can be beneficial to negotiate and install multiple identical Child SAs. IKEv2 [RFC7296] already allows installing multiple identical Child SAs, it offers no method to negotiate the number of Child SAs or indicate the purpose for the multiple Child SAs that are requested.

When two IKEv2 peers want to negotiate multiple Child SAs, it is useful to be able to convey how many Child SAs are required for optimized traffic. This avoids triggering CREATE_CHILD_SA exchanges that will only be rejected by the peer.

1.1. 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.

2. Performance bottlenecks

Currently, most IPsec implementations are limited by using one CPU or network queue per Child SA. There are a number of practical reasons for this, but a key limitation is that sharing the crypto state, counters and sequence numbers between multiple CPUs is not feasible without a significant performance penalty. There is a need to negotiate and establish multiple Child SAs with identical TSi/TSr on a per-queue or per-CPU basis.

3. Negotiation of CPU specific Child SAs

When negotiating CPU specific Child SAs, the first SA negotiated either in an IKE_AUTH exchange or CREATE_CHILD_SA is called Fallback SA. This Child SA is similar to a regular Child SA in that it is not bound to a single CPU. This Fallback Child SA (or its rekeyed successors) MUST remain active for the lifetime of the IPsec session to ensure that there is always a Child SA that can be selected to send traffic over, in case a per-resource Child SA is not available. Additional Child SAs are installed bound to a specific CPU. These Child SAs are responsible for the bulk of the traffic.

The CPU_QUEUES notification payload is sent in the IKE_AUTH or CREATE_CHILD_SA Exchange indicating the negotiated Child SA is a Fallback SA.

The CPU_QUEUES notification value refers to the number of additional resource-specific Child SAs that may be installed for this particular TSi/TSr combination excluding the Fallback Child SA. Both peers send the preferred minimum number of additional Child SAs to install. Both peers pick the maximum of the two numbers (within reason). That is, if the initiator prefers 16 and the responder prefers 48, then the number negotiated is 48. The responder may at any time reject additional Child SAs by returning TS_MAX_QUEUE. It should not return NO_ADDITIONAL_SAS, as there might be another Child SAs with different Traffic Selectors that would still be allowed by the peer.

CPU-specific Child SAs are negotiated as regular Child SAs using the CREATE_CHILD_SA exchange and are identified by a CPU_QUEUE_INFO notification. Upon installation, each Child SA is associated with an additional local selector, such as CPU or queue. These additional Child SAs MUST be negotiated with identical Child SA properties that were negotiated for the Fallback SA. This includes cryptographic algorithms, Traffic Selectors, Mode (e.g. transport mode), compression usage, etc. However, the Child SAs do have their own individual keying material that is derived according to the regular IKEv2 process. The CPU_QUEUE_INFO can be empty or contain some identifying data that could be useful for debugging purposes.

Additional Child SAs can be started on-demand or can be started all at once. Peers may also delete specific per-resource Child SAs if they deem the associated resource to be idle. The Fallback SA MUST NOT be deleted while any per-resource Child SAs are still present.

During the CREATE_CHILD_SA rekey for the Child SA, the CPU_QUEUE_INFO notification MAY be included, but regardless of whether or not it is included, the rekeyed Child SA MUST be bound to the same resource(s) as the Child SA that is being rekeyed.

As with regular Child SA rekeying, the new Child SA may not be different from the rekeyed Child SA with respect to cryptographic algorithms and MUST cover the original Traffic Selector ranges.

If a CREATE_CHILD_SA exchange request containing both a CPU_QUEUE_INFO and a CPU_QUEUES notification is received, the responder MUST ignore the CPU_QUEUE_INFO payload. If a CREATE_CHILD_SA exchange reply is received with both CPU_QUEUE_INFO and CPU_QUEUES notifications, the initiator MUST ignore the notification that it did not send in the request.

The CPU_QUEUES notification, even when it is sent in the IKE_AUTH exchange, is not an attribute of the IKE peer. It is an attribute of the Child SA, similar to the USE_TRANSPORT notification. That is, an IKE peer can have multiple Child SAs covering different traffic selectors and selectively decide whether or not to enable additional per-resource Child SAs for each of these Child SAs covering different Traffic Selectors.

4. Implementation Considerations

There are various considerations that an implementation can use to determine the best way to install multiple Child SAs. Below are examples of such strategies.

A simple distribution could be to install one additional Child SA on each CPU. The Fallback Child SA ensures that any CPU generating traffic to be encrypted has an available (if not optimal) Child SA to use. Any subsequent Child SAs with identical TSi/TSr Traffic Selectors are installed in such a way to only be used by a single CPU or network queue.

Performing per-CPU Child SA negotiations can result in both peers initiating additional Child SAs at once. This is especially likely if per-CPU Child SAs are triggered by individual SADB_ACQUIRE [RFC2367] messages. Responders should install the additional Child SA on a CPU with the least amount of additional Child SAs for this TSi/TSr pair. It should count outstanding SADB_ACQUIRES as an assigned additional Child SA. It is still possible that when the peers only have one slot left to assign, that both peers send a CREATE_CHILD_SA request at the same time.

As an optimization, additional Child SAs that see little traffic MAY be deleted. The Fallback Child SA MUST NOT be deleted when idle, as it is likely to be idle if enough per-CPU Child SAs are installed. However, if one of those per-CPU child SAs is deleted because it was idle, and subsequently that CPU starts to generate traffic again, that traffic does not have a per-CPU Child SA and will be encrypted using the Fallback Child SA. Meanwhile, the IKE daemon might be negotiating to bring up a new per-CPU Child SA.

When the number of queues or CPUs are different between the peers, the peer with the least amount of queues or CPUs MAY decide to not install a second outbound Child SA for the same resource as it will never use it to send traffic. However, it MUST install all inbound Child SAs as it has committed to receiving traffic on these negotiated Child SAs.

If per-CPU SADB_ACQUIRE messages are implemented (see Section 6), the Traffic Selector (TSi) entry containing the information of the trigger packet should still be included in the TS set. This information MAY be used by the peer to select the most optimal target CPU to install the additional Child SA on. For example, if the trigger packet was for a TCP destination to port 25 (SMTP), it might be able to install the Child SA on the CPU that is also running the mail server process. Trigger packet Traffic Selectors are documented in [RFC7296] Section 2.9.

As per RFC 7296, rekeying a Child SA SHOULD use the same (or wider) Traffic Selectors to ensure that the new Child SA covers everything that the rekeyed Child SA covers. This includes Traffic Selectors negotiated via Configuration Payloads (CP) such as INTERNAL_IP4_ADDRESS which may use the original wide TS set or use the narrowed TS set.

5. Payload Format

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

5.1. CPU_QUEUES Notify Status Message Payload

1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
! Next Payload										!C! RESERVED										Payload Length									
! Protocol ID										! SPI Size										Notify Message Type									
! Minimum number of IPsec SAs																													

- * Protocol ID (1 octet) - MUST be 0. MUST be ignored if not 0.
- * SPI Size (1 octet) - MUST be 0. MUST be ignored if not 0.
- * Notify Status Message Type (2 octets) - set to [TBD1]
- * Minimum number of per-CPU IPsec SAs (4 octets). MUST be greater than 0. If 0 is received, it MUST be interpreted as 1.

Note: The Fallback Child SA that is not bound to a single CPU is not counted as part of these numbers.

5.2. CPU_QUEUE_INFO Notify Status Message Payload

1										2										3											
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
! Next Payload										!C!	RESERVED					!	Payload Length										!				
! Protocol ID										!	SPI Size					!	Notify Message Type										!				
!										~										Optional queue identifier										!	
!										~																				!	
!										~																				!	

- * Protocol ID (1 octet) - MUST be 0. MUST be ignored if not 0.
- * SPI Size (1 octet) - MUST be 0. MUST be ignored if not 0.
- * Notify Status Message Type (2 octets) - set to [TBD2]
- * Optional Payload Data. This value MAY be set to convey the local identity of the queue. The value SHOULD be a unique identifier and the peer SHOULD only use it for debugging purposes.

5.3. TS_MAX_QUEUE Notify Error Message Payload

1										2										3												
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	
! Next Payload										!	C! RESERVED										!	Payload Length										!
! Protocol ID										!	SPI Size										!	Notify Message Type										!

- * Protocol ID (1 octet) - MUST be 0. MUST be ignored if not 0.
- * SPI Size (1 octet) - MUST be 0. MUST be ignored if not 0.
- * Notify Error Message Type (2 octets) - set to [TBD3]
- * Optional Payload Data. Must be 0.

6. Operational Considerations

Implementations supporting per-CPU SAs SHOULD extend their local SPD selector, and the mechanism of on-demand negotiation that is triggered by traffic to include a CPU (or queue) identifier in their SADB_ACQUIRE message from the SPD to the IKE daemon. If the IKEv2 extension defined in this document is negotiated with the peer, a node which does not support receiving per-CPU SADB_ACQUIRE messages MAY initiate all its Child SAs immediately upon receiving the (only) SADB_ACQUIRE it will receive from the IPsec stack. Such implementations also need to be careful when receiving a Delete Notify request for a per-CPU Child SA, as it has no method to detect when it should bring up such a per-CPU Child SA again later. And bringing the deleted per-CPU Child SA up again immediately after receiving the Delete Notify might cause an infinite loop between the peers. Another issue of not bringing up all its per-CPU Child SAs is that if the peer acts similarly, the two peers might end up with only the Fallback SA without ever activating any per-CPU Child SAs. It is there for RECOMMENDED to implement per-CPU SADB_ACQUIRE messages.

The minimum number of Child SAs negotiated should not be treated as the maximum number of allowed Child SAs. Peers SHOULD be lenient with this number to account for corner cases. For example, during Child SA rekeying, there might be a large number of additional Child SAs created before the old Child SAs are torn down. Similarly, when using on-demand Child SAs, both ends could trigger multiple Child SA requests as the initial packet causing the Child SA negotiation might have been transported to the peer via the Fallback SA where its reply packet might also trigger an on-demand Child SA negotiation to start. A peer may want to allow up to double the negotiated minimum number of Child SAs, and rely on idleness of Child SAs to tear down any unused Child SAs gradually to reach an optimal number of Child SAs. Adding too many SAs may slow down per-packet SAD lookup.

Implementations might support dynamically moving a per-CPU Child SAs from one CPU to another CPU. If this method is supported, implementations must be careful to move both the inbound and outbound SAs. If the IPsec endpoint is a gateway, it can move the inbound SA and outbound SA independently from each other. It is likely that for a gateway, IPsec traffic would be asymmetric. If the IPsec endpoint is the same host responsible for generating the traffic, the inbound and outbound SAs SHOULD remain as a pair on the same CPU. If a host previously skipped installing an outbound SA because it would be an unused duplicate outbound SA, it will have to create and add the previously skipped outbound SA to the SAD with the new CPU ID. The inbound SA may not have CPU ID in the SAD. Adding the outbound SA to the SAD requires access to the key material, whereas for updating the CPU selector on an existing outbound SAs. access to key material

might not be needed. To support this, the IKE software might have to hold on to the key material longer than it normally would, as it might actively attempt to destroy key material from memory that it no longer needs access to.

7. Security Considerations

[TO DO]

8. Implementation Status

[Note to RFC Editor: Please remove this section and the reference to [RFC6982] before publication.]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

Authors are requested to add a note to the RFC Editor at the top of this section, advising the Editor to remove the entire section before publication, as well as the reference to [RFC7942].

8.1. Linux XFRM

Organization: Linux kernel XFRM

Name: XFRM-PCPU-v2

<https://git.kernel.org/pub/scm/linux/kernel/git/klasert/linux-stk.git/log/?h=xfrm-pcpu-v2>

Description: An initial Kernel IPsec implementation of the per-CPU method.

Level of maturity: Alpha

Coverage: Implements Fallback Child SA and per-CPU Child SAs. It only supports the NETLINK API. The PFKEYv2 API is not supported.

Licensing: GPLv2

Implementation experience: The Linux XFRM implementation added two additional attributes to support per-CPU SAs. There is a new attribute `XFRMA_SA_PCPU`, u32, for the SAD entry. This attribute should present on the outgoing SA, per-CPU Child SAs, starting from 0. This attribute MUST NOT be present on the Fallback XFRM SA. It is used by the kernel only for the outgoing traffic, (clear to encrypted). The incoming SAs, both the Fallback and the per-CPU SA, do not need `XFRMA_SA_PCPU` attribute. XFRM stack can not use CPU id on the incoming SA. The kernel internally sets the value to `0xFFFFFFFF` for the incoming SA and the Fallback SA. However, one may add `XFRMA_SA_PCPU` to the incoming per-CPU SA to steer the ESP flow, to a specific Q or CPU e.g. `ethtool` ntuple configuration. The SPD entry has new flag `XFRM_POLICY_CPU_ACQUIRE`. It should be set only on the "out" policy. The flag should be disabled when the policy is a trap policy, without SPD entries. After a successful negotiation of `CPU_QUEUES`, while adding the Fallback Child SA, the SPD entry can be updated with the `XFRM_POLICY_CPU_ACQUIRE` flag. When `XFRM_POLICY_CPU_ACQUIRE` is set, the `XFRM_MSG_ACQUIRE` generated will include the `XFRMA_SA_PCPU` attribute.

Contact: Steffen Klassert steffen.klassert@secunet.com

8.2. Libreswan

Organization: The Libreswan Project

Name: `pcpu-3` https://libreswan.org/wiki/XFRM_pCPU

Description: An initial IKE implementation of the per-CPU method.

Level of maturity: Alpha

Coverage: implements Fallback Child SA and per-CPU additional Child SAs

Licensing: GPLv2

Implementation experience: TBD

Contact: Libreswan Development: swan-dev@libreswan.org

8.3. strongSwan

Organization: The StrongSwan Project

Name: StrongSwan <https://github.com/strongswan/strongswan/tree/per-cpu-sas-poc/>

Description: An initial IKE implementation of the per-CPU method.

Level of maturity: Alpha

Coverage: implements Fallback Child SA and per-CPU additional Child SAs

Licensing: GPLv2

Implementation experience: StrongSwan use private space values for notifications CPU_QUEUES (40970) and QUEUE_INFO (40971).

Contact: Tobias Brunner tobias@strongswan.org

8.4. iproute2

Organization: The iproute2 Project

Name: iproute2 <https://github.com/antonyantony/iproute2/tree/pcpu-v1>

Description: Implemented the per-CPU attributes for the "ip xfrm" command.

Level of maturity: Alpha

Licensing: GPLv2

Implementation experience: TBD

Contact: Antony Antony antony.antony@secunet.com

9. IANA Considerations

This document defines two new IKEv2 Notify Message Type payloads for the IANA "IKEv2 Notify Message Types - Status Types" registry.

Value	Notify Type Messages - Status Types	Reference
[TBD1]	CPU_QUEUES	[this document]
[TBD2]	CPU_QUEUE_INFO	[this document]

Figure 1

This document defines one new IKEv2 Notify Message Type payloads for the IANA "IKEv2 Notify Message Types - Error Types" registry.

Value	Notify Type Messages - Status Types	Reference
[TBD3]	TS_MAX_QUEUE	[this document]

Figure 2

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>>.
- [RFC2367] McDonald, D., Metz, C., and B. Phan, "PF_KEY Key Management API, Version 2", RFC 2367, DOI 10.17487/RFC2367, July 1998, <<https://www.rfc-editor.org/info/rfc2367>>.
- [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>>.
- [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>>.

10.2. Informative References

- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.
- [RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, DOI 10.17487/RFC6982, July 2013, <<https://www.rfc-editor.org/info/rfc6982>>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <<https://www.rfc-editor.org/info/rfc7942>>.

Authors' Addresses

Antony Antony
secunet Security Networks AG
Email: antony.antony@secunet.com

Tobias Brunner
codelabs GmbH
Email: tobias@codelabs.ch

Steffen Klassert
secunet Security Networks AG
Email: steffen.klassert@secunet.com

Paul Wouters
Aiven
Email: paul.wouters@aiven.io

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 1, 2022

CJ. Tjhai
Post-Quantum
T. Heider
genua GmbH
V. Smyslov
ELVIS-PLUS
January 28, 2022

Beyond 64KB Limit of IKEv2 Payloads
draft-tjhai-ikev2-beyond-64k-limit-02

Abstract

The maximum Internet Key Exchange Version 2 (IKEv2) payload size is limited to 64KB. This makes IKEv2 not usable for conservative post-quantum cryptosystem whose public-key is larger than 64KB. This document discusses the considerations and defines a mechanism to exchange large post-quantum public keys and signatures in IKEv2.

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 August 1, 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	4
2. Proposed Solution Overview	4
3. Protocol Details	6
4. Operational Considerations	8
5. Denial of Service Considerations	9
6. Security Considerations	9
7. IANA Considerations	10
8. References	10
8.1. Normative References	10
8.2. Informative References	11
Appendix A. Alternative Approaches	11
A.1. Hash and URL	11
A.1.1. Key Exchange Payload	12
A.1.2. Certificate Payload	13
A.2. Incremental Transfer and Confirmation	13
Authors' Addresses	14

1. Introduction

Digital communications are secured by public-key cryptography algorithms that rely on computational hardness assumptions such as the difficulty in factoring large integers or that of finding the discrete logarithm on an elliptic curve group or finite-field. Recent advances in quantum computing, however, have caused some concerns on the security of these assumptions. It is conjectured that these hard computational problems can be solved in polynomial time when sufficiently large quantum computers become available. The concerns have prompted the National Institute of Standards and Technology (NIST) to initiate a process to standardize one or more public-key algorithms that are quantum-resistant. This family of algorithms is known as post-quantum or quantum-resistant cryptographic algorithms.

It would be ideal if these cryptographic algorithms can be drop-in replacements to the classical algorithms we currently use. Unfortunately, almost all of the post-quantum cryptography algorithms have either public-key, ciphertext or signature size that is many times larger than their classical counterparts. One of the issues that this will cause, in particular for UDP-based protocols such as IPsec, is fragmentation of packets at IP layer. In the context of IPsec/IKEv2 post-quantum key exchange, the fragmentation issue can be

addressed by sending the post-quantum exchange data in IKE_INTERMEDIATE [I-D.ietf-ipsecme-ikev2-intermediate], which is the intermediary state between IKE_SA_INIT and IKE_AUTH. This is the approach taken in [I-D.ietf-ipsecme-ikev2-multiple-ke] whereby a classical and one or more post-quantum key exchanges are combined in order to establish security associations that are quantum-resistant.

Because all public-key cryptography algorithms rely on computational hardness assumptions, the confidence of a cryptographic algorithm is an important consideration. An algorithm that has been well-studied and field-tested is generally better trusted than newer ones. Unfortunately, the confidence of post-quantum cryptographic algorithms is relatively low. All of the algorithms submitted to NIST post-quantum standardization are based on new computational hardness assumptions and despite being conjectured to be resistant to quantum computer attacks, they have not been well cryptanalyzed compared to the classical counterparts. An exception to this is the Goppa-code based McEliece cryptosystem [McEliece] which has withstood years of cryptanalysis since 1978 and still remains unbroken. It is not surprising that a more efficient and CCA secure version of McEliece cryptosystem, Classic McEliece [CM], is selected as one of the finalists in NIST post-quantum cryptography standardization (at the time of writing this document) [NIST]. Furthermore, this cryptosystem has also been recommended for long-term confidentiality protection of data, see [BSI].

While there is interest in using McEliece cryptosystem, in particular for information that needs to remain secure for a long time, there is a challenge in integrating it with IKEv2 [RFC7296]. One characteristic of McElieces cryptosystem is the very asymmetric size of its ciphertext and public-key. While its ciphertext is the smallest compared to all other post-quantum key-establishment algorithms submitted to NIST, the size of its public-key, however, is the largest. The smallest public-key size of Classic McEliece is 255KB. This presents a problem if one were to use Classic McEliece for key-establishment with IKEv2, as the maximum payload size supported by IKEv2 is limited to 64KB. This document describes a mechanism to support IKEv2 key-exchange with key size larger than 64KB, building on the works in [I-D.ietf-ipsecme-ikev2-multiple-ke] and [I-D.ietf-ipsecme-ikev2-intermediate].

In addition, some post-quantum digital signature algorithms that are finalists or alternate candidates of NIST post-quantum cryptography standardization (at the time of writing this document) [NIST], also have either public key size or signature size greater than 64 KB. This makes impossible to use them in IKEv2 as drop-in replacement for classic signature algorithms.

This document is focused on providing a solution for using large post-quantum algorithms related data (public keys and signatures) in IKEv2. It is not a goal of this document to provide a generic solution to transport large data blocks of arbitrary type in IKEv2.

1.1. Terminology

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] and [RFC8174].

This document assumes familiarity with IKEv2 concept described in [RFC7296].

2. Proposed Solution Overview

While the Length field in IKEv2 header has a size of 32 bits, so that the maximum size of an IKEv2 message can theoretically reach 4 GB, the size of any individual payload inside a message is limited to 64 KB due to the fact that the Payload Length field in generic payload header consumes 16 bits only. This makes impossible to transfer blocks of data greater than 64 KB, such as public keys of some post-quantum key exchange methods or some post-quantum signatures. In IKEv2 three types of payloads may contain large amounts of data related to post-quantum algorithms:

- o Key Exchange (KE) payload in case of large public key of a post-quantum key exchange method
- o Authentication (AUTH) payload in case of large signature of a post-quantum signature algorithm
- o Certificate (CERT) payloads in case of large public key of a post-quantum signature algorithm

This specification proposes the following solution to the problem: when block of data of a particular type (public key, signature) exceeds 64 KB in size, it is split into a series of chunks smaller than 64 KB. Each chunk then is placed in its own payload, so that the large block of data is eventually transferred in a series of adjacent payloads of the same type. All these payloads MUST have the same values in their headers (except for Next Payload and Payload Length fields) and MUST be transferred adjacent to each other, so that no other payload should appear between them.

This approach works well for KE and AUTH payloads, since only one such large block is transferred in a message and there is no

ambiguity when it is split over multiple payloads. However, when multiple certificates containing large public keys are transferred and each of them is further splitted into several CERT payloads, there must be a way to find boundaries between these certificates on a receiving side. To solve this problem an empty CERT payload MUST be inserted between other non-empty CERT payloads to mark boundaries between individual certificates. Note that large certificates can also be transferred using "Hash and URL" format (see Section 3.6 of [RFC7296]).

The resulting message would exceed 64 KB in size, so that it would not fit into a single UDP datagram. Even if TCP transport [I-D.ietf-ipsecme-rfc8229bis] is used, the size of any individual IKE message in a TCP stream is still limited to 64 KB. For this reason, IKE Fragmentation [RFC7383] MUST be used regardless of the transport protocol if peers are going to transfer large blocks of data. In the case of TCP, the size of fragments is not related to path MTU and can reach 64 KB.

Since IKE Fragmentation is mandatory with this extension and it only can be used on encrypted IKE messages, large blocks of data cannot be transferred in the IKE_SA_INIT exchange.

While mandatory IKE Fragmentation makes it possible to transfer large blocks of data using UDP transport, in practice it may be problematic for the following reason. When fragmenting large messages the number of fragments would be high and all of them are sent at once. If any of these fragment were lost, all the fragments should be re-sent. In congested network environments this would have a negative effect, worsening the congestion. Moreover, the number of IKE message fragments is limited to 2^{16} . With typical size of IKE message fragment equal to PMTU or less, this would limit the size of a single large block of data to ~30-100 MB. While this is enough for current applications of this specification, it may be a limitation in the future.

TCP transport has built-in acknowledgement and congestion control mechanisms and does not suffer from these problems. In addition, since the size of IKE message fragments in case of TCP may be up to 64 KB, the size of a single large block of data can in theory reach 4 GB. However, [I-D.ietf-ipsecme-rfc8229bis] implies that if TCP is used as transport for IKE, it is also used for ESP. Encapsulation ESP in TCP has a lot of negative effects on performance and on ESP functionality (see Section 10 of [I-D.ietf-ipsecme-rfc8229bis]).

This specifications proposes a mixed transport mode as a solution to the problem. In this mode, IKE uses TCP as its transport, while ESP packets are still sent over IP or are encapsulated in UDP. The use

of mixed transport mode is optional and is negotiated in the IKE_SA_INIT exchange.

3. Protocol Details

The initiator starts creating an IKEv2 SA by sending the IKE_SA_INIT request message. If the initiator is going to transfer large blocks of data (e.g. large public keys), then it should make some preparations:

- o IKEV2_FRAGMENTATION_SUPPORTED notification MUST be included to negotiate support for IKE Fragmentation
- o INTERMEDIATE_EXCHANGE_SUPPORTED notification MUST be included if the initiator proposes key exchange methods with public keys greater than 64 KB
- o If the initiator is going to use mixed transport mode then it starts the IKE_SA_INIT request using UDP port 4500 and includes a new status type notification IKE_OVER_TCP (<TBA by IANA>), which has protocol 0, SPI size 0 and contains no data; if the initiator starts the IKE_SA_INIT over TCP, then the mixed transport mode cannot be used and this notification SHOULD NOT be included, it MUST be ignored by the responder if it is still included in the message

Note that UDP port 4500 (and not port 500) is used for the IKE_SA_INIT messages, which is allowed by [RFC7296]. Using port 4500 allows return routability check for UDP messages to be carried out and ensures ESP packets can get through if they are UDP encapsulated.

The responder supporting this specification MUST agree on using IKE Fragmentation by sending back IKEV2_FRAGMENTATION_SUPPORTED notification. If it selects proposal with key exchange method having public key greater than 64 KB, then it MUST agree on using the IKE_INTERMEDIATE exchange by sending back INTERMEDIATE_EXCHANGE_SUPPORTED notification.

If the initiator proposed using mixed transport mode by initiating the IKE_SA_INIT exchange over UDP port 4500 and including IKE_OVER_TCP notification and the responder supports this mode and is willing to use it, then it sends this notification back in the IKE_SA_INIT response. In this case the initiator MUST switch to TCP using destination port 4500 in the next exchange (IKE_INTERMEDIATE or IKE_AUTH) and the responder MUST be prepared to receive the next exchange request message on TCP port 4500. Once switched all subsequent IKE exchanges MUST use TCP transport as described in [I-D.ietf-ipsecme-rfc8229bis], but ESP packets MUST NOT be sent using

TCP, instead they are sent either over IP or using UDP encapsulation, depending on the presence of NAT, which is determined in the IKE_SA_INIT exchange. Note, that if NAT is detected and UDP encapsulation of ESP is used, then NAT keepalive messages MUST be sent by the peer that is behind NAT over UDP using ports from the IKE_SA_INIT exchange, as defined in [RFC3948].

If the responder does not support mixed transport mode, then it ignores the IKE_OVER_TCP notification and all subsequent IKE exchanges will use UDP transport. Note, that in case the initiator started the IKE_SA_INIT over TCP then the IKE_OVER_TCP notification would not be included in the request message and there would be no option for mixed transport mode.

Initiator	Responder

HDR, SAi1, KEi1, Ni, N(NAT_DETECTION_SOURCE_IP), N(NAT_DETECTION_DESTINATION_IP), N(IKEV2_FRAGMENTATION_SUPPORTED), [N(INTERMEDIATE_EXCHANGE_SUPPORTED),] [N(IKE_OVER_TCP)] --->	HDR, SAR1, KER1, Nr, N(NAT_DETECTION_SOURCE_IP), N(NAT_DETECTION_DESTINATION_IP), N(IKEV2_FRAGMENTATION_SUPPORTED), [N(INTERMEDIATE_EXCHANGE_SUPPORTED),] <--- [N(IKE_OVER_TCP)]

Once the IKE_SA_INIT exchange is completed, the peers continue with the following exchanges: one or more IKE_INTERMEDIATE exchanges in case multiple key exchanges are negotiated or the IKE_AUTH exchange, as shown below. Note that all messages containing large blocks of data are sent fragmented using IKE Fragmentation mechanism, but they are not shown here for the sake of simplicity.

Initiator	Responder

HDR, SK{KEi2.1, KEi2.2, KEi2.3, ...} --->	<--- HDR, SK{KEr2.1, KEr2.2, ...}
HDR, SK{KEi3.1, KEi3.2, ...} --->	<--- HDR, SK{KEr3.1, KEr3.2, ...}
...	
HDR, SK{IDi, [IDr,] [CERTi1, CERTi2, ...] [CERTREQ,] [IDr,] AUTHi1, AUTHi2, ... SAi2, TSi, TSr} --->	<--- HDR, SK{IDr, [CERTr1, CERTr2, ...] AUTHr1, AUTHr2, ... SAr2, TSi, TSr}

Since the Payload Length field in the generic IKE payload header has a size of 16 bits, it is impossible to set a proper value for it in the Encrypted Payload header when it contains inner payloads with total length greater than 64 KB. However, using IKE Fragmentation is mandatory when transferring large blocks of data (even in case of TCP transport) and with IKE Fragmentation, the Payload Length field in the Encrypted payload is never transmitted. Instead, the IKE message fragments that appear on the wire are limited to 64 KB in size, so there is no problem with setting a proper value in the Length field of Encrypted Fragment payloads. However, when IKE_INTERMEDIATE exchanges are being authenticated, the content of the Encrypted Payload before encryption and fragmentation is fed to the prf. In this case if the size of the Encrypted payload content exceeds 64 KB then the Payload Length field in the Encrypted Payload header MUST be set to zero when feeding into the prf. On receipt it MUST be checked that the total size of unencrypted payloads the Encrypted Payload contains matches the size of the Encrypted payload calculated from the size of the received message.

4. Operational Considerations

The IKE fragmentation does not require additional infrastructure, however, there is non-zero probability of lost packets when sending a large number of fragments over a UDP connection. Given a set of fragments, when transmitted, each one of them is not individually acknowledged and if any one of them is lost, the entire set will have to be retransmitted. As a consequence, given the size of the payload and also the potential of multiple retransmissions, there may be a noticeable delay in establishing an security association (SA), in particular in lossy network conditions. Therefore, implementations

MAY use a longer timeout value for the purpose of dead-peer detection, but a balance needs to be struck as too large of a value will open up security vulnerabilities as discussed in the following section. In the unlikely event where there is a frequent retransmission due to loss of fragments, implementations MAY send the IKE messages over a TCP connection as specified in [I-D.ietf-ipsecme-rfc8229bis]. If TCP is used as IKE transport, then using mixed transport mode is RECOMMENDED to allow better ESP performance.

5. Denial of Service Considerations

Malicious peers may send a large number of fragments, but incomplete, to the legitimate peer causing memory exhaustion. It is RECOMMENDED that the strategies and recommendations described in [RFC8019] be implemented to counter possible DoS attacks.

An alternative arrangement, if peers do not support [RFC8019], is to allow the transfer of large block of data only after peers are authenticated. In other words, key-establishment using large public-key should not be done to establish an IKE SA, but it should only be used to establish a Child SA or rekeying an IKE SA. In order to protect IKE messages from quantum threats, multiple key-exchanges using a combination of classical and post-quantum ciphers, as described in [I-D.ietf-ipsecme-ikev2-multiple-ke] can be used. Nonetheless, this approach has a limitation whereby if a digital signature scheme with large public-key or signature payload is used, it is still susceptible to DoS attacks.

*** More to be populated in the next version ***

6. Security Considerations

If TCP encapsulation is used, refer to the security considerations in [I-D.ietf-ipsecme-rfc8229bis].

Downloading or transferring large amounts of data is an expensive operation, bandwidth and memory wise. Consequently, implementations should consider using a longer rekeying interval or SHOULD consider relaxing forward secrecy requirements but using CCA-secure key-establishment algorithms only. With chosen-ciphertext attack (CCA)-secure schemes, there is no loss in security if the public-key is reused.

7. IANA Considerations

This document defines a new Notify Message Type in the "Notify Message Types - Status Types" registry:

<TBA> IKE_OVER_TCP

8. References

8.1. Normative References

- [I-D.ietf-ipsecme-ikev2-intermediate]
Smyslov, V., "Intermediate Exchange in the IKEv2 Protocol", draft-ietf-ipsecme-ikev2-intermediate-07 (work in progress), August 2021.
- [I-D.ietf-ipsecme-ikev2-multiple-ke]
Tjhai, C., Tomlinson, M., Bartlett, G., Fluhrer, S., Geest, D. V., Garcia-Morchon, O., and V. Smyslov, "Multiple Key Exchanges in IKEv2", draft-ietf-ipsecme-ikev2-multiple-ke-04 (work in progress), September 2021.
- [I-D.ietf-ipsecme-rfc8229bis]
Smyslov, V. and T. Pauly, "TCP Encapsulation of IKE and IPsec Packets", draft-ietf-ipsecme-rfc8229bis-01 (work in progress), October 2021.
- [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>>.
- [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>>.
- [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>>.

- [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>>.

8.2. Informative References

- [BSI] Federal Office for Information Security, "Cryptographic Mechanisms: Recommendations and Key Lengths", 2020, <<https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf>>.
- [CM] Classic McEliece submission team, "Classic McEliece: NIST post-quantum cryptography standardization finalist", 2020, <<https://classic.mceliece.org/>>.
- [FIPS-202] National Institute of Standards and Technology, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", 2015, <<https://doi.org/10.6028/NIST.FIPS.202>>.
- [McEliece] McEliece, R., "A Public-key Cryptosystem based on Algebraic Coding Theory", DSN Progress Report 42-44, 1978.
- [NIST] National Institute of Standards and Technology, "Post-Quantum Cryptography Standardization", <<https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization>>.
- [RFC8019] Nir, Y. and V. Smyslov, "Protecting Internet Key Exchange Protocol Version 2 (IKEv2) Implementations from Distributed Denial-of-Service Attacks", RFC 8019, DOI 10.17487/RFC8019, November 2016, <<https://www.rfc-editor.org/info/rfc8019>>.

Appendix A. Alternative Approaches

A.1. Hash and URL

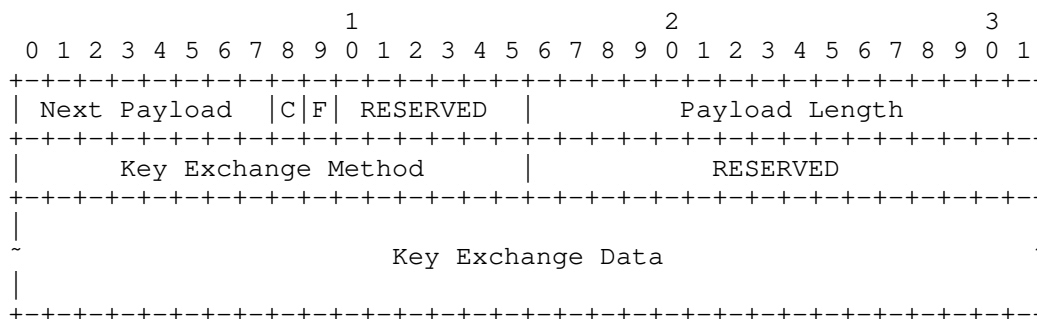
[RFC7296] defines a mechanism whereby an authentication payload such as a certificate can be encoded using a hash value and a URL. A peer utilizes HTTP_CERT_LOOKUP_SUPPORTED Notify payload to indicate that X.509 certificates are not transported in-band, instead the other peer shall fetch the certificates from the given URL and verify its integrity from the hash value. In this way, the peer needs to send 20 octets plus a variable length URL only over the wire, instead of a

few kilobytes of payload, which is useful in the event IKEv2 message fragmentation is not available.

Large public keys can be transported by reusing the same technique and this can be done in two ways, as described below.

A.1.1. Key Exchange Payload

The Key Exchange Data field of IKEv2 Key Exchange Payload contains a single format, which is a blob that is only meaningful to the specified key exchange method. In order to support hash and URL data, an encoding format needs to be specified on the header.



The reserved bit-field F above specifies the encoding format. If it is 0, the Key Exchange Data is a blob as specified in RFC7296. On the other hand if it is 1, the Key Exchange Data is in the form of hash and URL format, whereby the hash value is the SHA3-256 digest [FIPS-202] of the replaced value truncated to 20 octets and the URL value is a variable length URL (in either http or https schema) that resolves to the DER-encoded of the replaced value itself.

Because the hash and URL value is transported in a Key Exchange Payload, it is possible to support the use-case of a single post-quantum key-establishment with large public-key. This payload will be sent as part of IKE_SA_INIT exchange and it will not require IKE_INTERMEDIATE exchanges.

While using hash and URL method to transport large key-establishment data requires minimal modification to IKEv2 protocol, there are disadvantages from deployment point of view that may make this method impractical. Firstly, an IKE peer that originates a hash and URL value will also need to implement additional infrastructure so that it can serve HTTP requests in order to allow the actual key-establishment data to be fetched. While this may not be an issue for Internet facing peers, in the context of road-warrior or remote-access cases, the hash and URL value is initiated by an IKE peer that

is usually a device sitting behind a network address translation (NAT) device and as such, it may not be able to run a publicly reachable HTTP server infrastructure on the same device. An possible solution for these cases is to publish the key-establishment data to a separate server, which is not practical as one cannot expect an IKE initiator to always have deployed an HTTP server. Lastly, IKE peers are predominantly deployed at the network edge where strict firewall rules are generally enforced. The need to open up another port to serve HTTP requests may cause either technical or policy complication that render this approach unacceptable.

The hash and URL approach is vulnerable to (distributed) denial of service attacks as an unauthenticated rogue peer may trick a legitimate peer to fetch a large amount of random meaningless data from a remote server. Implementations SHOULD NOT blindly download all of the data in the given URL. Because a legitimate key-establishment payload should be DER-encoded, they SHOULD download the first few octets to determine the length of the ASN.1 structure representing these octets, then only continue to download the remaining decoded number of octets if the length is expected for the chosen key-establishment algorithm. It should be noted that the content of the data to be downloaded may be under attacker's control and therefore even if the length is as expected, the content may be meaningless bit that is of no use for key-establishment.

A.1.2. Certificate Payload

An alternative is to re-purpose Certificate Payload to carry the hash and URL value of the post-quantum key-establishment data. At the time of writing, the IANA registry defines two hash and URL encoding values, namely X.509 certificate and X.509 certificate bundle. In order to use this payload, a new encoding value for key establishment data will be required.

Because a Certificate Payload is part of IKE_AUTH message, unlike the previous approach, the hash and URL value of the key-establishment data shall be transported via IKE_INTERMEDIATE message. As such, it will not be able to support a single post-quantum key-establishment with a large public-key case. Furthermore, it is semantically incorrect to re-purpose Certificate Payload, which is intended to carry authentication data, to transport key-establishment data.

A.2. Incremental Transfer and Confirmation

As stated in Section 4 of [RFC7383], if any single fragment is lost, the receiving peer will not be able to reassemble the original large key-establishment payload. The above bulk transfer is susceptible to this issue. There is another way to transfer these payload chunks

that is less susceptible to this, but at the cost of higher latency. Instead of transferring in a bulk, each Key Exchange payload chunk must be acknowledged prior to sending the subsequent chunk. As before, the large key-establishment payload is split over several Key Exchange payload chunks where each of them share the same Key Exchange Method value. Each chunk is then sent to the peer using the IKE_INTERMEDIATE message, and each one must be acknowledged by the receiving peer before the subsequent chunk can be sent.

Initiator	Responder

HDR, SAi1, KEi1, Ni, N(IKEV2_FRAGMENTATION_SUPPORTED)*, N(INTERMEDIATE_EXCHANGE_SUPPORTED) --->	
	HDR, SAr1, KEr1, Nr, N(IKEV2_FRAGMENTATION_SUPPORTED)*, <--- N(INTERMEDIATE_EXCHANGE_SUPPORTED)
HDR, SK{KEi2.1, ...} --->	
	<--- HDR, SK{}
HDR, SK{KEi2.2, ...} --->	
	<--- HDR, SK{}
HDR, SK{KEi2.3, ...} --->	
	<--- HDR, SK{KEr2, ...}
HDR, SK{}	---
*: optional	

In order to support key-encapsulation mechanism, the receiving peer has to wait until the entire chunks are received before it can respond with its own Key Exchange payload, which may not be large.

Authors' Addresses

CJ Tjhai
Post-Quantum
UK

Email: cjt@post-quantum.com

Tobias Heider
genua GmbH
DE

Email: me@tobhe.de

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

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