Stream: Internet Engineering Task Force (IETF)
RFC: 9370
Updates: 7296
Category: Standards Track
Published: May 2023
ISSN: 2070-1721
Authors: CJ. Tjhai, *Post-Quantum*; 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*

RFC 9370
Multiple Key Exchanges in the Internet Key Exchange Protocol Version 2 (IKEv2)

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.

This document utilizes the IKE_INTERMEDIATE exchange, where multiple key exchanges are performed when an IKE SA is being established. It also introduces a new IKEv2 exchange, IKE_FOLLOWUP_KE, which is used for the same purpose when the IKE SA is being rekeyed or is creating additional Child SAs.

This document updates RFC 7296 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

Copyright Notice

Table of Contents

1. Introduction

1.1. Problem Description

The Internet Key Exchange Protocol version 2 (IKEv2), as specified in [RFC7296], uses the Diffie-Hellman (DH) or the Elliptic Curve Diffie-Hellman (ECDH) algorithm, which shall be referred to as "(EC)DH" collectively, to establish a shared secret between an initiator and a responder. The security of the (EC)DH algorithms relies on the difficulty to solve a discrete logarithm problem in multiplicative (and, respectively, elliptic curve) groups when the order of the group parameter is large enough. While solving such a problem remains infeasible 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 to quantum-computer attacks. This family of cryptosystems is known as "post-quantum cryptography" (or "PQC"). It is sometimes also referred to as "quantum-safe cryptography" (or "QSC") or "quantum-resistant cryptography" (or "QRC").

It is essential to have the ability to perform one or more post-quantum key exchanges in conjunction with an (EC)DH key exchange so that the resulting shared key is resistant to quantum-computer attacks. Since there is currently no post-quantum key exchange that is as well-studied as (EC)DH, performing multiple key exchanges with different post-quantum algorithms along with the well-established classical key-exchange algorithms addresses this concern, since the overall security is at least as strong as each individual primitive.

1.2. Proposed Extension

This document describes a method to perform multiple successive key exchanges in IKEv2. This method allows integration of PQC in IKEv2, while maintaining backward compatibility, to derive a set of IKE keys that is resistant to quantum-computer attacks. This extension allows the negotiation of one or more PQC algorithms to exchange data, in addition to the existing (EC)DH key exchange data. It is believed that the feature of using more than one post-quantum algorithm 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 PQC algorithms.

IKE peers perform multiple successive key exchanges to establish an IKE SA. Each exchange produces some shared secret, and these secrets are combined in a way such that:

(a) the final shared secret is computed from all of the component key exchange secrets;

(b) unless both peers support and agree to use the additional key exchanges introduced in this specification, the final shared secret equivalent to the shared secret specified in [RFC7296] is obtained; and

(c) if any part of the component key exchange method is a post-quantum algorithm, the final shared secret is post-quantum secure.

Some post-quantum key exchange payloads may have sizes larger than the standard maximum transmission unit (MTU) size. Therefore, there could be issues with fragmentation at the IP layer. In order to allow the use of those larger payload sizes, this mechanism relies on the IKE_INTERMEDIATE exchange as specified in [RFC9242]. With this mechanism, the key exchange is initiated using a smaller, possibly classical primitive, such as (EC)DH. Then, before the IKE_AUTH exchange, one or more IKE_INTERMEDIATE exchanges are carried out, 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, as described in Section 2.2.2; thus, the final IKE SA keys depend on all the key exchanges. Hence, the keys are secure if any of the key exchanges are secure.

While this extension is primarily aimed at IKE SAs due to the potential fragmentation issue discussed above, it also applies to CREATE_CHILD_SA exchanges as illustrated in Section 2.2.4 for creating/rekeying of Child SAs and rekeying of IKE SAs.

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 specified in [RFC8784].

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

This specification does not attempt to address key exchanges with KE payloads longer than 64 KB; the current IKE payload format does not allow such a possibility. At the time of writing, it appears likely that there are a number of key exchanges available that would not have such a requirement. [BEYOND-64K] discusses approaches that could be taken to exchange huge payloads if such a requirement were needed.

1.3. Document Organization

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

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. Multiple Key Exchanges

2.1. Design Overview

Most post-quantum key agreement algorithms are relatively new. Thus, they are not fully trusted. There are also many proposed algorithms that have different trade-offs and that rely on different hard problems. The concern is that some of these hard problems may turn out to be easier to solve than anticipated; 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 combining it with 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, this specification utilizes the IKE_INTERMEDIATE exchange defined in [RFC9242]. The initial IKE_SA_INIT messages do not have any inherent fragmentation support within IKE. However, IKE_SA_INIT messages can include a relatively short KE payload. The additional key exchanges are performed using IKE_INTERMEDIATE messages that follow the IKE_SA_INIT exchange. This is to allow the standard IKE fragmentation mechanisms (which cannot be used in IKE_SA_INIT) to be available for the potentially large Key Exchange payloads with post-quantum algorithm data.

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 that require multiple round trips to complete, a separate document should define how such methods are split into several IKE_INTERMEDIATE exchanges.

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

It is assumed that new Transform Type 4 identifiers will be assigned later for various post-quantum key exchanges [IKEV2TYPE4ID]. This specification does not make a distinction between classical (EC)DH and post-quantum key exchanges, nor between post-quantum algorithms that are true key exchanges and post-quantum algorithms that act as key transport mechanisms: all are treated equivalently by the protocol. This document renames a field in the Key Exchange Payload from "Diffie-Hellman Group Num" to "Key Exchange Method". This document also renames Transform Type 4 from "Diffie-Hellman Group (D-H)" to "Key Exchange Method (KE)". The corresponding renaming to the IANA registry is described in Section 3.

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. This approach allows any combination of the defined key exchange methods to take place. This also allows IKE peers to perform a single post-quantum key exchange in the IKE_SA_INIT without additional key exchanges, provided that the 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 that 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. Following that, 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} -->
```

2.2. Protocol Details

In the simplest case, the initiator starts 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 (because either the key exchange that it selects is small enough not to fragment or the initiator is confident that fragmentation will be handled either by IP fragmentation or by transport via TCP).

In this case, the initiator performs the IKE_SA_INIT for a single key exchange using a Transform Type 4 (possibly with a post-quantum algorithm) 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 wants to negotiate multiple key exchanges, then the initiator uses the protocol behavior listed below.

2.2.1. 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 are defined: Additional Key Exchange 1 (ADDKE1) with IANA-assigned value 6, Additional Key Exchange 2 (ADDKE2) (7), Additional Key Exchange 3 (ADDKE3) (8), Additional Key Exchange 4 (ADDKE4) (9), Additional Key Exchange 5 (ADDKE5) (10), Additional Key Exchange 6 (ADDKE6) (11), and Additional Key Exchange 7 (ADDKE7) (12). They are collectively called "Additional Key Exchange (ADDKE) Transform Types" in this document and have slightly different semantics than the existing IKEv2 Transform Types. They are interpreted as an indication of additional key exchange methods that peers agree 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 the IDs for Transform Type 4, so they all share a single IANA registry for Transform IDs.

The 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. This is so that the key exchange negotiated using Additional Key Exchange i always precedes that of Additional Key Exchange i + 1. Each additional key exchange method **MUST** be fully completed before the next one is started.

With these semantics, note that ADDKE Transform Types are not associated with any particular type of key exchange and do not have any Transform IDs that are specific per Transform Type IANA registry. Instead, they all share a single registry for Transform IDs, namely "Transform Type 4 - Key Exchange Method Transform IDs". All 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 (EC)DH key exchange methods, but that is not a requirement. Additional key exchange methods are proposed using ADDKE Transform Types. All of these transform types are optional; the initiator is free to select any of them for proposing additional key exchange methods. Consequently, if none of the ADDKE Transform Types are included in the proposal, then this proposal indicates the performing of standard IKEv2, as defined in [RFC7296]. On the other hand, if the initiator includes any ADDKE Transform Type in the proposal, the responder **MUST** select one of the algorithms proposed using this type. Note that this is not a new requirement; this behavior is already specified in Section 2.7 of [RFC7296]. A Transform ID NONE **MAY** be added to those transform types that contain key exchange methods which the initiator believes are optional according to its local policy.

The responder performs the negotiation using the standard IKEv2 procedure described in Section 3.3 of [RFC7296]. However, for the ADDKE Transform Types, the responder's choice **MUST NOT** contain duplicated algorithms (those with an identical Transform ID and attributes), except for the 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 the following two transforms define the same algorithm: "ID=alg1, ATTR1=attr1, ATTR2=attr2" and "ID=alg1, ATTR2=attr2, ATTR1=attr1". If the responder is unable to select algorithms that are not duplicated for each proposed key exchange (either because the proposal contains too few choices or due to the local policy restrictions on using the proposed algorithms), then the responder **MUST** reject the message with an error notification of type NO_PROPOSAL_CHOSEN. If the responder's message contains one or more duplicated choices, the initiator should log the error and **MUST** treat the exchange as failed. The initiator **MUST NOT** initiate any IKE_INTERMEDIATE (or IKE_FOLLOWUP_KE) exchanges so that no new SA is created. If this happens in the CREATE_CHILD_SA exchange, then the initiator **MAY** delete the IKE SA over which the invalid message was received by sending a Delete payload.

If the responder selects NONE for some ADDKE Transform Types (provided they are proposed by the initiator), then any corresponding additional key exchanges **MUST NOT** take place. Therefore, if the initiator includes NONE in all of the ADDKE Transform Types and the responder selects this value for all of them, then no IKE_INTERMEDIATE exchanges performing additional key exchanges will take place between the peers. Note that the IKE_INTERMEDIATE exchanges may still take place for other purposes.

The initiator **MAY** propose ADDKE Transform Types that are not consecutive, for example, proposing ADDKE2 and ADDKE5 Transform Types only. The responder **MUST** treat all of the omitted ADDKE transforms as if they were proposed with Transform ID NONE.

Below is an example of the SA payload in the initiator's IKE_SA_INIT request message. Here, the abbreviation "KE" is used for the Key Exchange transform, which this document renames from the Diffie-Hellman Group transform. Additionally, the notations PQ_KEM_1, PQ_KEM_2, and PQ_KEM_3 are used to represent Transform IDs that have yet to be defined of some 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 ADDKE2 ( ID = PQ_KEM_1 )
               |
               +-- Transform ADDKE2 ( ID = PQ_KEM_2 )
               |
               +-- Transform ADDKE3 ( ID = PQ_KEM_1 )
               |
               +-- Transform ADDKE3 ( ID = PQ_KEM_2 )
               |
               +-- Transform ADDKE5 ( ID = PQ_KEM_3 )
               |
               +-- Transform ADDKE5 ( ID = NONE )
```

In this example, the initiator proposes performing the initial key exchange using a 4096-bit MODP Group followed by two mandatory additional key exchanges (i.e., ADDKE2 and ADDKE3 Transform Types) using PQ_KEM_1 and PQ_KEM_2 methods in any order followed by an additional key exchange (i.e., ADDKE5 Transform Type) using the 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 that it does not want to additionally perform PQ_KEM_3.

```
      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 ADDKE2 ( ID = PQ_KEM_2 )
               |
               +-- Transform ADDKE3 ( ID = PQ_KEM_1 )
               |
               +-- Transform ADDKE5 ( ID = NONE )
```

If the initiator includes any ADDKE Transform Types into the SA payload in the IKE_SA_INIT exchange request message, then it **MUST** also negotiate the use of the IKE_INTERMEDIATE exchange, as described in [RFC9242] by including an INTERMEDIATE_EXCHANGE_SUPPORTED notification in the same message. If the responder agrees to use additional key exchanges while establishing an initial IKE SA, it **MUST** also return this notification in the IKE_SA_INIT response message, 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 ADDKE Transform Types 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 if no proposal has been chosen (i.e., the responder will send a NO_PROPOSAL_CHOSEN notification).

```
Initiator                           Responder
---------------------------------------------------------------------
HDR, SAi1(.. ADDKE*...), KEi, Ni,
N(INTERMEDIATE_EXCHANGE_SUPPORTED)    --->
                                    HDR, SAr1(.. ADDKE*...), KEr, Nr,
                                    [CERTREQ],
                            <---    N(INTERMEDIATE_EXCHANGE_SUPPORTED)
```

It is possible for an attacker to manage to send a response to the initiator's IKE_SA_INIT request before the legitimate responder does. If the initiator continues to create the IKE SA using this response, the attempt will fail. Implementers may wish to consider strategies as described in Section 2.4 of [RFC7296] to handle such an attack.

2.2.2. 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 an IKE_INTERMEDIATE exchange, as described in [RFC9242].

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

The initiator sends key exchange data in the KEi(n) payload. This message is protected with the current SK_ei/SK_ai keys. The notation "KEi(n)" denotes the n-th IKE_INTERMEDIATE KE payload from the initiator; the integer "n" is sequential starting from 1.

On receiving this, the responder sends back key exchange payload KEr(n); "KEr(n)" denotes the n-th IKE_INTERMEDIATE KE payload from the responder. Similar to how the request is protected, this message 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:

```
SKEYSEED(n) = prf(SK_d(n-1), SK(n) | Ni | Nr)
```

From this exchange, SK(n) is the resulting shared secret. Ni and Nr are nonces from the IKE_SA_INIT exchange. SK_d(n-1) is the last generated SK_d (derived from IKE_SA_INIT for the first use of IKE_INTERMEDIATE and, otherwise, from the previous IKE_INTERMEDIATE exchange). The other keying materials, SK_d, SK_ai, SK_ar, SK_ei, SK_er, SK_pi, and SK_pr, are generated from the SKEYSEED(n) as follows:

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

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

2.2.3. 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, as described in [RFC7296], with the modification of AUTH payload calculation described in [RFC9242].

2.2.4. CREATE_CHILD_SA Exchange

The CREATE_CHILD_SA exchange is used in IKEv2 for the purposes of creating additional Child SAs, rekeying these Child SAs, and rekeying IKE SA itself. When creating or rekeying Child SAs, the peers may optionally perform a key exchange to add a fresh entropy into the session keys. In the case of an 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 a CREATE_CHILD_SA exchange is negotiated in a similar fashion to the initial IKE exchange, see Section 2.2.1. If the initiator includes any ADDKE Transform Types in the SA payload (along with Transform Type 4), and if 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 follow the CREATE_CHILD_SA exchange. The IKE_FOLLOWUP_KE exchange is introduced especially for transferring data of additional key exchanges following the one performed in the CREATE_CHILD_SA. Its Exchange Type value is 44.

The key exchange negotiated via Transform Type 4 always takes place in the CREATE_CHILD_SA exchange, as per the IKEv2 specification [RFC7296]. Additional key exchanges are performed in an order of the values of their Transform Types so that the key exchange negotiated using Additional Key Exchange i always precedes the key exchange negotiated using Additional Key Exchange i + 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 that require multiple round trips, a separate document should define how such methods are split into several IKE_FOLLOWUP_KE exchanges.

After an IKE SA is created, the window size may be greater than one; thus, multiple concurrent exchanges may be in progress. Therefore, it is essential to link the IKE_FOLLOWUP_KE exchanges together with the corresponding CREATE_CHILD_SA exchange. Once an IKE SA is created, all IKE exchanges are independent and IKEv2 doesn't have a built-in mechanism to link an exchange with another one. A new status type notification called "ADDITIONAL_KEY_EXCHANGE" is introduced for this purpose. Its Notify Message Type value is 16441, and the 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_KE response message in case the next IKE_FOLLOWUP_KE exchange is expected, filling it with some data that would allow linking the current exchange to the next one. The initiator **MUST** send back this notification intact in the request message of the next IKE_FOLLOWUP_KE 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_KE), SK {KEi(1),
 N(ADDITIONAL_KEY_EXCHANGE)(link1)} -->
                               <--  HDR(IKE_FOLLOWUP_KE), SK {KEr(1),
                                   N(ADDITIONAL_KEY_EXCHANGE)(link2)}

HDR(IKE_FOLLOWUP_KE), SK {KEi(2),
 N(ADDITIONAL_KEY_EXCHANGE)(link2)} -->
                               <--  HDR(IKE_FOLLOWUP_KE), SK {KEr(2),
                                   N(ADDITIONAL_KEY_EXCHANGE)(link3)}

HDR(IKE_FOLLOWUP_KE), SK {KEi(3),
 N(ADDITIONAL_KEY_EXCHANGE)(link3)} -->
                               <--  HDR(IKE_FOLLOWUP_KE), 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.

Due to some unexpected events (e.g., a reboot), it is possible that the initiator may lose its state, forget that it is in the process of performing additional key exchanges, and never start the remaining IKE_FOLLOWUP_KE exchanges. The responder **MUST** handle this situation gracefully and delete the associated state if it does not receive the next expected IKE_FOLLOWUP_KE request after some reasonable period of time. Due to various factors such as computational resource and key exchange algorithm used, note that it is not possible to give normative guidance on how long this timeout period should be. In general, 5-20 seconds of waiting time should be appropriate in most cases.

It may also take too long for the initiator to prepare and to send the next IKE_FOLLOWUP_KE request, or, due to the network conditions, the request could be lost and retransmitted. In this case, the message may reach the responder when it has already deleted the associated state, following the advice above. If the responder receives an IKE_FOLLOWUP_KE message for which it does not have a key exchange state, it **MUST** send back a new error type notification called "STATE_NOT_FOUND". This is an error notification that is not fatal to the IKE SA. Its Notify Message Type value is 47, its Protocol ID and SPI Size are both set to 0, and the data is empty. If the initiator receives this notification in response to an IKE_FOLLOWUP_KE exchange performing an additional key exchange, it **MUST** cancel this exchange and **MUST** treat the whole series of exchanges started from the CREATE_CHILD_SA exchange as having failed. In most cases, the receipt of this notification is caused by the premature deletion of the corresponding state on the responder (the time period between IKE_FOLLOWUP_KE exchanges appeared to be 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, which may eventually be 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 the IKE SA by sending a DELETE payload.

It is possible that the peers start rekeying the IKE SA or the Child SA 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, in the case of simultaneous rekeying, if two identical new IKE SAs (or two pairs of Child SAs) are created, then one of them should be deleted. Which one to delete is determined by comparing the values of four nonces that are used in the colliding CREATE_CHILD_SA exchanges. The IKE SA (or pair of Child SAs) created by the exchange in which the smallest nonce is 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. Instead, they would be created only after the series of IKE_FOLLOWUP_KE exchanges is finished. For this reason, if additional key exchanges are negotiated in the CREATE_CHILD_SA exchange in which the smallest nonce is used, then, because there is nothing to delete yet, the initiator of this exchange just stops the rekeying process, and it **MUST NOT** initiate the IKE_FOLLOWUP_KE 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 the rekey of an SA that is already being rekeyed by this peer (i.e., the CREATE_CHILD_SA exchange initiated by this peer has already been completed, and the series of IKE_FOLLOWUP_KE exchanges is in progress). In this case, a TEMPORARY_FAILURE notification **MUST** be sent in response to such a request.

If multiple key exchanges are negotiated in the CREATE_CHILD_SA exchange, then the resulting keys are computed as follows.

In the case of an IKE SA rekey:

```
        SKEYSEED = prf(SK_d, SK(0) | Ni | Nr | SK(1) | ... SK(n))
```

In the case of a Child SA creation or rekey:

```
KEYMAT = prf+ (SK_d, SK(0) | Ni | Nr | SK(1) |  ... SK(n))
```

In both cases, SK_d is from the existing IKE SA; SK(0), Ni, and 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.

2.2.5. Interaction with IKEv2 Extensions

It is believed that this specification requires no modification to the IKEv2 extensions defined so far. In particular, the IKE SA resumption mechanism defined in [RFC5723] can be used to resume IKE SAs created using this specification.

2.2.5.1. Interaction with Childless IKE SA

It is possible to establish IKE SAs with post-quantum algorithms by only using IKE_FOLLOWUP_KE exchanges and without the use of IKE_INTERMEDIATE exchanges. In this case, the IKE SA that is created from the IKE_SA_INIT exchange, can be immediately rekeyed with CREATE_CHILD_SA with additional key exchanges, where IKE_FOLLOWUP_KE messages are used for these additional key exchanges. If the classical key exchange method is used in the IKE_SA_INIT message, the very first Child SA created in IKE_AUTH will offer no resistance against the quantum threats. Consequently, if the peers' local policy requires all Child SAs to be post-quantum secure, then the peers can avoid creating the very first Child SA by adopting [RFC6023]. In this case, the initiator sends two types of proposals in the IKE_SA_INIT request: one with and another one without ADDKE Transform Types. The responder chooses the latter proposal type and includes a CHILDLESS_IKEV2_SUPPORTED notification in the IKE_SA_INIT response. Assuming that the initiator supports childless IKE SA extension, both peers perform the modified IKE_AUTH exchange described in [RFC6023], and no Child SA is created in this exchange. The peers should then immediately rekey the IKE SA and subsequently create the Child SAs, all with additional key exchanges using a CREATE_CHILD_SA exchange.

It is also possible for the initiator to send proposals without any ADDKE Transform Types in the IKE_SA_INIT message. In this instance, the responder will have no information about whether or not the initiator supports the extension in this specification. This may not be efficient, as the responder will have to wait for the subsequent CREATE_CHILD_SA request to determine whether or not the initiator's request is appropriate for its local policy.

The support for childless IKE SA is not negotiated, but it is the responder that indicates the support for this mode. As such, the responder cannot enforce that the initiator use this mode. Therefore, it is entirely possible that the initiator does not support this extension and sends IKE_AUTH request as per [RFC7296] instead of [RFC6023]. In this case, the responder may respond with an error that is not fatal, such as the NO_PROPOSAL_CHOSEN notify message type.

Note that if the initial IKE SA is used to transfer sensitive information, then this information will not be protected using the additional key exchanges, which may use post-quantum algorithms. In this arrangement, the peers will have to use post-quantum algorithm in Transform Type 4 in order to mitigate the risk of quantum attack.

3. IANA Considerations

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

```
44          IKE_FOLLOWUP_KE
```

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

This document renames the IKEv2 registry originally titled "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
6	Additional Key Exchange 1 (ADDKE1)	(optional in IKE, AH, ESP)
7	Additional Key Exchange 2 (ADDKE2)	(optional in IKE, AH, ESP)
8	Additional Key Exchange 3 (ADDKE3)	(optional in IKE, AH, ESP)
9	Additional Key Exchange 4 (ADDKE4)	(optional in IKE, AH, ESP)
10	Additional Key Exchange 5 (ADDKE5)	(optional in IKE, AH, ESP)
11	Additional Key Exchange 6 (ADDKE6)	(optional in IKE, AH, ESP)
12	Additional Key Exchange 7 (ADDKE7)	(optional in IKE, AH, ESP)

Table 1: "Transform Type Values" Registry

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

```
16441       ADDITIONAL_KEY_EXCHANGE
```

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

```
47          STATE_NOT_FOUND
```

IANA has added the following instructions for designated experts for the "Transform Type 4 - Key Exchange Method Transform IDs" subregistry:

- While adding new Key Exchange (KE) methods, the following considerations must be applied. A KE method must take exactly one round-trip (one IKEv2 exchange), and at the end of this exchange, both peers must be able to derive the shared secret. In addition, any public value that peers exchanged during a KE method must fit into a single IKEv2 payload. If these restrictions are not met for a KE method, then there must be documentation on how this KE method is used in IKEv2.

IANA has also completed the following changes. It is assumed that [RFC9370] refers to this specification.

- Added a reference to [RFC9370] in what was the "Transform Type 4 - Diffie-Hellman Group Transform IDs" registry.
- Replaced the Note on what was the "Transform Type 4 - Diffie-Hellman Group Transform IDs" registry with the following notes:

  This registry was originally named "Transform Type 4 - Diffie-Hellman Group Transform IDs" and was referenced using that name in a number of RFCs published prior to [RFC9370], which gave it the current title.

  This registry is used by the "Key Exchange Method (KE)" transform type and by all "Additional Key Exchange (ADDKE)" transform types.

  To find out requirement levels for Key Exchange Methods for IKEv2, see [RFC8247].
- Appended [RFC9370] to the Reference column of Transform Type 4 in the "Transform Type Values" registry.
- Added these notes to the "Transform Type Values" registry:

  "Key Exchange Method (KE)" transform type was originally named "Diffie-Hellman Group (D-H)" and was referenced by that name in a number of RFCs published prior to [RFC9370], which gave it the current title.

  All "Additional Key Exchange (ADDKE)" entries use the same "Transform Type 4 - Key Exchange Method Transform IDs" registry as the "Key Exchange Method (KE)" entry.

4. Security Considerations

The extension in this document is intended to mitigate two possible threats in IKEv2: the compromise of (EC)DH key exchange using Shor's algorithm while remaining backward compatible and the potential compromise of existing or future PQC key exchange algorithms. To address the former threat, this extension allows the establishment of a shared secret by using multiple key exchanges: typically, one classical (EC)DH and the other one post-quantum algorithm. In order to address the latter threat, multiple key exchanges using a post-quantum algorithm can be performed to form the shared key.

Unlike key exchange methods (Transform Type 4), the Encryption Algorithm (Transform Type 1), the Pseudorandom Function (Transform Type 2), and the Integrity Algorithm (Transform Type 3) are not susceptible to Shor's algorithm. However, they are susceptible to Grover's attack [GROVER], which allows a quantum computer to perform a brute force key search, using quadratically fewer steps than the classical counterpart. Simply increasing the key length can mitigate this attack. It was previously believed that one needed to double the key length of these algorithms. However, there are a number of factors that suggest that it is quite unlikely to achieve the quadratic speedup using Grover's algorithm. According to NIST [NISTPQCFAQ], current applications can continue using an AES algorithm with the minimum key length of 128 bits. Nevertheless, if the data needs to remain secure for many years to come, one may want to consider using a longer key size for the algorithms in Transform Types 1-3.

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 post-quantum algorithm ensure that the final SKEYSEED is not compromised. This assumes that the algorithm defined in the Transform Type 2 is quantum resistant.

The ordering of the additional key exchanges should not matter in general, as only the final shared secret is of interest. Nonetheless, because the strength of the running shared secret increases with every additional key exchange, an implementer may want to first perform the most secure method (in some metrics) followed by less secure methods.

The main focus of this document is to prevent a passive attacker from performing a "harvest-and-decrypt" attack: in other words, attackers that record messages exchanged today and proceed to decrypt them once they have access to cryptographically relevant quantum computers. 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 that the authentication step remains classical. In particular, the authenticity of the SAs established under IKEv2 is protected by using a pre-shared key or digital signature algorithms. While the pre-shared key option, provided the key is long enough, is post-quantum secure, the other algorithms are not. Moreover, in implementations where scalability is a requirement, the pre-shared key method may not be suitable. Post-quantum authenticity may be provided by using a post-quantum digital signature.
- Secondly, 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 are archived with decryption by quantum computers 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 an on-path attack. Consequently, there is less urgency for post-quantum authenticity compared to post-quantum confidentiality.

Performing multiple key exchanges while establishing an IKE SA increases the responder's susceptibility to DoS attacks because of an increased amount of resources needed 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 an initial IKE SA without performing additional key exchanges if the initiator includes such an option in its proposals. Then, peers immediately rekey the initial IKE SA with the CREATE_CHILD_SA exchange, and additional key exchanges are performed via the IKE_FOLLOWUP_KE exchanges. In this case, at the point when resource-intensive operations are required, the peers have already authenticated each other. However, in the context of hybrid post-quantum key exchanges, this scenario would leave the 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 a full protection. This is similar to the scenario described in [RFC8784]. Depending on the arrangement and peers' policy, this scenario may or may not be appropriate. For example, in the G-IKEv2 protocol [G-IKEV2], the cryptographic materials are sent from the group controller to the group members when the initial IKE SA is created.

5. References

5.1. 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>.

[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>.

[RFC9242] Smyslov, V., "Intermediate Exchange in the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 9242, DOI 10.17487/RFC9242, May 2022, <https://www.rfc-editor.org/info/rfc9242>.

5.2. Informative References

[BEYOND-64K] Tjhai, CJ., Heider, T., and V. Smyslov, "Beyond 64KB Limit of IKEv2 Payloads", Work in Progress, Internet-Draft, draft-tjhai-ikev2-beyond-64k-limit-03, 28 July 2022, <https://datatracker.ietf.org/doc/html/draft-tjhai-ikev2-beyond-64k-limit-03>.

[G-IKEV2] Smyslov, V. and B. Weis, "Group Key Management using IKEv2", Work in Progress, Internet-Draft, draft-ietf-ipsecme-g-ikev2-09, 19 April 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-ipsecme-g-ikev2-09>.

[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), pp. 212-219, DOI 10.48550/arXiv.quant-ph/9605043, May 1996, <https://doi.org/10.48550/arXiv.quant-ph/9605043>.

[IKEV2TYPE4ID] IANA, "Internet Key Exchange Version 2 (IKEv2) Parameters: Transform Type 4 - Diffie-Hellman Group Transform IDs", <https://www.iana.org/assignments/ikev2-parameters/>.

[NISTPQCFAQ] NIST, "Post-Quantum Cryptography Standard", January 2023, <https://csrc.nist.gov/Projects/post-quantum-cryptography/faqs>.

[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>.

[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>.

[RFC8247] Nir, Y., Kivinen, T., Wouters, P., and D. Migault, "Algorithm Implementation Requirements and Usage Guidance for the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 8247, DOI 10.17487/RFC8247, September 2017, <https://www.rfc-editor.org/info/rfc8247>.

[RFC8784] Fluhrer, 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 not normative, 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. IKE_INTERMEDIATE Exchanges Carrying Additional Key Exchange Payloads

The exchanges below show that the initiator proposes the use of additional key exchanges to establish an IKE SA. The initiator proposes three sets of additional key exchanges, all of which are optional. Therefore, the responder can choose NONE for some or all of the additional exchanges if the proposed key exchange methods are not supported or for whatever reasons the responder decides not to perform the additional key exchange.

```
Initiator                          Responder
---------------------------------------------------------------------
HDR(IKE_SA_INIT), SAi1(.. ADDKE*...), --->
KEi(Curve25519), 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 ADDKE1 (ID = PQ_KEM_1)
    Transform ADDKE1 (ID = PQ_KEM_2)
    Transform ADDKE1 (ID = NONE)
    Transform ADDKE2 (ID = PQ_KEM_3)
    Transform ADDKE2 (ID = PQ_KEM_4)
    Transform ADDKE2 (ID = NONE)
    Transform ADDKE3 (ID = PQ_KEM_5)
    Transform ADDKE3 (ID = PQ_KEM_6)
    Transform ADDKE3 (ID = NONE)
                   <--- HDR(IKE_SA_INIT), SAr1(.. ADDKE*...),
                        KEr(Curve25519), 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 ADDKE1 (ID = PQ_KEM_2)
                          Transform ADDKE2 (ID = NONE)
                          Transform ADDKE3 (ID = PQ_KEM_5)

HDR(IKE_INTERMEDIATE), SK {KEi(1)(PQ_KEM_2)} -->
                   <--- HDR(IKE_INTERMEDIATE), SK {KEr(1)(PQ_KEM_2)}
HDR(IKE_INTERMEDIATE), SK {KEi(2)(PQ_KEM_5)} -->
                   <--- HDR(IKE_INTERMEDIATE), SK {KEr(2)(PQ_KEM_5)}

HDR(IKE_AUTH), SK{ IDi, AUTH, SAi2, TSi, TSr } --->
                      <--- HDR(IKE_AUTH), SK{ IDr, AUTH, SAr2,
                           TSi, TSr }
```

In this particular example, the responder chooses to perform two additional key exchanges. It selects PQ_KEM_2, NONE, and PQ_KEM_5 for the first, second, and third additional key exchanges, respectively. As per [RFC7296], a set of keying materials is derived, in particular SK_d, SK_a[i/r], and SK_e[i/r]. Both peers then perform an IKE_INTERMEDIATE exchange, carrying

PQ_KEM_2 payload, which is protected with SK_e[i/r] and SK_a[i/r] keys. After the completion of this IKE_INTERMEDIATE exchange, the SKEYSEED is updated using SK(1), which is the PQ_KEM_2 shared secret, as follows.

```
SKEYSEED(1) = prf(SK_d, SK(1) | Ni | Nr)
```

The updated SKEYSEED value is then used to derive the following keying materials.

```
{SK_d(1) | SK_ai(1) | SK_ar(1) | SK_ei(1) | SK_er(1) | SK_pi(1) |
 SK_pr(1)} = prf+ (SKEYSEED(1), Ni | Nr | SPIi | SPIr)
```

As per [RFC9242], both peers compute IntAuth_i1 and IntAuth_r1 using the SK_pi(1) and SK_pr(1) keys, respectively. These values are required in the IKE_AUTH phase of the exchange.

In the next IKE_INTERMEDIATE exchange, the peers use SK_e[i/r](1) and SK_a[i/r](1) keys to protect the PQ_KEM_5 payload. After completing this exchange, keying materials are updated as follows:

```
SKEYSEED(2) = prf(SK_d(1), SK(2) | Ni | Nr)
{SK_d(2) | SK_ai(2) | SK_ar(2) | SK_ei(2) | SK_er(2) | SK_pi(2) |
    SK_pr(2)} = prf+ (SKEYSEED(2), Ni | Nr | SPIi | SPIr)
```

In this update, SK(2) is the shared secret from the third additional key exchange, i.e., PQ_KEM_5. Then, both peers compute the values of IntAuth_[i/r]2 using the SK_p[i/r](2) keys.

After the completion of the second IKE_INTERMEDIATE exchange, both peers continue to the IKE_AUTH exchange phase. As defined in [RFC9242], the values IntAuth_[i/r]2 are used to compute IntAuth, which, in turn, is used to calculate InitiatorSignedOctets and ResponderSignedOctets blobs (see Section 3.3.2 of [RFC9242]).

A.2. 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. Consequently, the IKE_INTERMEDIATE exchange does not take place, and the exchange proceeds to the IKE_AUTH phase. The resulting keying materials are the same as those derived with [RFC7296].

```
Initiator                     Responder
---------------------------------------------------------------------
HDR(IKE_SA_INIT), SAi1(.. ADDKE*...), --->
KEi(Curve25519), 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 ADDKE1 (ID = PQ_KEM_1)
    Transform ADDKE1 (ID = PQ_KEM_2)
    Transform ADDKE1 (ID = NONE)
    Transform ADDKE2 (ID = PQ_KEM_3)
    Transform ADDKE2 (ID = PQ_KEM_4)
    Transform ADDKE2 (ID = NONE)
                   <--- HDR(IKE_SA_INIT), SAr1(.. ADDKE*...),
                        KEr(Curve25519), 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 ADDKE1 (ID = NONE)
                            Transform ADDKE2 (ID = NONE)

HDR(IKE_AUTH), SK{ IDi, AUTH, SAi2, TSi, TSr } --->
                   <--- HDR(IKE_AUTH), SK{ IDr, AUTH, SAr2,
                        TSi, TSr }
```

A.3. 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, the responder includes a CHILDLESS_IKEV2_SUPPORTED notification in their IKE_SA_INIT response message. The initiator understands and supports this notification, exchanges a modified IKE_AUTH message with the responder, and rekeys the IKE SA immediately with additional key exchanges. Any Child SA will have to be created via a subsequent CREATED_CHILD_SA exchange.

```
Initiator                     Responder
---------------------------------------------------------------------
HDR(IKE_SA_INIT), SAi1, --->
KEi(Curve25519), Ni, N(IKEV2_FRAG_SUPPORTED)
                   <--- HDR(IKE_SA_INIT), SAr1,
                        KEr(Curve25519), 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(.. ADDKE*...), Ni, KEi(Curve25519) } --->
  Proposal #1
    Transform ECR (ID = ENCR_AES_GCM_16,
                   256-bit key)
    Transform PRF (ID = PRF_HMAC_SHA2_512)
    Transform KE (ID = Curve25519)
    Transform ADDKE1 (ID = PQ_KEM_1)
    Transform ADDKE1 (ID = PQ_KEM_2)
    Transform ADDKE2 (ID = PQ_KEM_5)
    Transform ADDKE2 (ID = PQ_KEM_6)
    Transform ADDKE2 (ID = NONE)
                   <--- HDR(CREATE_CHILD_SA), SK{ SAr(.. ADDKE*...),
                        Nr, KEr(Curve25519),
                        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 ADDKE1 (ID = PQ_KEM_2)
                            Transform ADDKE2 (ID = PQ_KEM_5)

HDR(IKE_FOLLOWUP_KE), SK{ KEi(1)(PQ_KEM_2), --->
N(ADDITIONAL_KEY_EXCHANGE)(link1) }
                  <--- HDR(IKE_FOLLOWUP_KE), SK{ KEr(1)(PQ_KEM_2),
                        N(ADDITIONAL_KEY_EXCHANGE)(link2) }

HDR(IKE_FOLLOWUP_KE), SK{ KEi(2)(PQ_KEM_5), --->
N(ADDITIONAL_KEY_EXCHANGE)(link2) }
                  <--- HDR(IKE_FOLLOWUP_KE), SK{ KEr(2)(PQ_KEM_5) }
```

A.4. No 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 that either PQ_KEM_1 or PQ_KEM_2 must be used to establish an IKE SA, but ADDKE2 Transform Type is optional. Therefore, the responder can either select PQ_KEM_3 or PQ_KEM_4 or omit this key exchange by selecting NONE. Although the responder supports the optional PQ_KEM_3 and PQ_KEM_4 methods, it does not support either the PQ_KEM_1 or the PQ_KEM_2 mandatory method; therefore, it responds with a NO_PROPOSAL_CHOSEN notification.

```
Initiator                         Responder
---------------------------------------------------------------------
HDR(IKE_SA_INIT), SAi1(.. ADDKE*...), --->
KEi(Curve25519), 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 ADDKE1 (ID = PQ_KEM_1)
    Transform ADDKE1 (ID = PQ_KEM_2)
    Transform ADDKE2 (ID = PQ_KEM_3)
    Transform ADDKE2 (ID = PQ_KEM_4)
    Transform ADDKE2 (ID = NONE)
                         <--- HDR(IKE_SA_INIT), N(NO_PROPOSAL_CHOSEN)
```

Appendix B. Design Criteria

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

1) Need for PQC in IPsec

   Quantum computers, which might become feasible in the near future, pose a threat to our classical public key cryptography. PQC, a family of public key cryptography that is believed to be resistant to these computers, needs to be integrated into the IPsec protocol suite to restore confidentiality and authenticity.

2) Hybrid

   There is currently no post-quantum key exchange that is trusted at the level that (EC)DH is trusted for defending against conventional (non-quantum) adversaries. A hybrid post-quantum algorithm to be introduced, along with the well-established primitives, addresses this concern, since the overall security is at least as strong as each individual primitive.

3) Focus on post-quantum confidentiality

   A passive attacker can store all monitored encrypted IPsec communication today and decrypt it once a quantum computer is available in the future. This attack can have serious consequences that will not be visible for years to come. On the other hand, 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, this specification focuses on confidentiality due to the urgency of this problem and presents a defense against the serious attack described above, but it does not address authentication because it is less urgent at this stage.

4) Limit the 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 the initiator and the responder need to agree on a hybrid group or if multiple public keys need to be exchanged.

5) Not post-quantum specific

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 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) Backward compatibility and interoperability

This is a fundamental requirement to ensure that hybrid post-quantum IKEv2 and standard IKEv2 implementations as per [RFC7296] are interoperable.

11) Compliance with USA Federal Information Processing Standards (FIPS)

IPsec is widely used in Federal Information Systems, and FIPS certification is an important requirement. However, at the time of writing, none of the algorithms that is believed to be post-quantum is yet FIPS compliant. Nonetheless, it is possible to combine this post-quantum algorithm with a FIPS-compliant key establishment method so that the overall design remains FIPS compliant [NISTPQCFAQ].

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.

Appendix C. Alternative Design

This section gives an overview on a number of alternative approaches that have been considered but later discarded. These approaches are as follows.

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

  A method to combine the various key exchanges into a single large KE payload was considered. This effort is documented in a previous version of this document (draft-tjhai-ipsecme-hybrid-qske-ikev2-01). This method allows 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 the KE payload only

  With the objective of not introducing unnecessary notify payloads, a method to communicate the hybrid post-quantum proposal in the KE payload during the first pass of the protocol exchange was considered. Unfortunately, this design is susceptible to the following downgrade attack. Consider the scenario where there is an on-path attacker sitting between an initiator and a responder. Through the SAi payload, the initiator proposes using a hybrid post-quantum group and, as a fallback, a Diffie-Hellman group; and through the KEi payload, the initiator proposes a list of hybrid post-quantum proposals and policies. The on-path attacker intercepts this traffic and replies with N(INVALID_KE_PAYLOAD), suggesting a downgrade to the fallback Diffie-Hellman group instead. The initiator then resends the same SAi payload and the KEi payload containing the public value of the fallback Diffie-Hellman group. Note that the attacker may forward the second IKE_SA_INIT message only to the responder. 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 does not offer the corresponding public value in the KEi 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 a KE payload alone is not sufficient, and the initiator **MUST** always indicate its preferred post-quantum proposals and policies in a notify payload in the subsequent IKE_SA_INIT messages following an N(INVALID_KE_PAYLOAD) response.

- New payload types to negotiate hybrid proposals 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 backward compatible by not setting its critical flag as per Section 2.5 of [RFC7296], it is believed that it may not be that simple in practice. Since the original release of IKEv2 in RFC 4306, no new payload type has ever been proposed; therefore, this creates a potential risk of having a backward-compatibility issue from nonconformant IKEv2 implementations. Since there appears to be no other compelling advantages apart from a semantic one, the existing Transform Type and notify payloads are used instead.

- 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. However, following the same reasoning as above may not be a good idea from a backward-compatibility perspective. Using a single KE payload would require encoding or formatting to be defined so that both peers are able to compose and extract the individual public values. However, it is believed 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 for handling multiple KE payloads.

- Fragmentation

  The 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 have been discarded are outlined as follows.

  The first approach is to treat the entire IKE_SA_INIT message as a stream of bytes, which is then split into a number of fragments, each of which is wrapped onto a payload that will fit into the size of the network MTU. The payload that wraps each fragment has a new payload type, and it is 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. This approach has been discarded due to its blanket approach to fragmentation. In cases where only a few payloads need to be fragmented, this approach appears to be overly complicated.

  Another idea that has been discarded is fragmenting an individual payload without introducing a new payload type. The idea is 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.

```
                     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|F| RESERVED  |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Diffie-Hellman Group Number  |     Fragment Identifier       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Fragment Index        |        Total Fragments        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Total KE Payload Data Length                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                       Fragmented KE Payload                   ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: Example of How to Fragment a KE Payload

When the flag F is set, 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 2-octet RESERVED field following Diffie-Hellman Group Number was to be used as a fragment identifier to help the 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.

This approach has been discarded because it is believed that the working group may not want to use the RESERVED field to change the format of a packet, and that implementers may not like the added complexity 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. This issue can be solved using IKE_INTERMEDIATE [RFC9242] to transport the large post-quantum key exchange payloads and using the generic IKEv2 fragmentation protocol [RFC7383].

- 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 an 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 detail does not fit the principles of the document where it should deal with generic hybrid key exchange protocol and not a specific ciphersuite. Furthermore, there are enough Diffie-Hellman group identifiers should this be required in the future.

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.

Authors' Addresses

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

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

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

Scott Fluhrer
Cisco Systems
Email: sfluhrer@cisco.com

Daniel Van Geest
ISARA Corporation
Email: daniel.vangeest.ietf@gmail.com

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

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