Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: July 4, 2016

S. Fluhrer
D. McGrew
P. Kampanakis
Cisco Systems
January 2016

Postquantum Preshared Keys for IKEv2
draft-fluhrer-qr-ikev2-01

Abstract

This document describes an extension of IKEv2 to allow it to be resistant to a Quantum Computer, by using preshared keys

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 4, 2016.

Copyright Notice

Table of Contents

1. Introduction

It is an open question whether or not it is feasible to build a quantum computer, but if it is, many of the cryptographic algorithms and protocols currently in use would be insecure. A quantum computer would be able to solve DH and ECDH problems, and this would imply that the security of existing IKEv2 systems would be compromised. IKEv1 when used with preshared keys does not share this vulnerability, because those keys are one of the inputs to the key derivation function. If the preshared key have sufficient entropy and the PRF and encryption and authentication transforms are postquantum secure, then the resulting system is believed to be quantum resistant, that is, believed to be invulnerable to an attacker with a Quantum Computer.

This document describes a way to extend IKEv2 to have a similar property; assuming that the two end systems share a long secret key, then the resulting exchange is quantum resistant. By bringing postquantum security to IKEv2, this note removes the need to use an obsolete version of the Internet Key Exchange in order to achieve that security goal.

The general idea is that we add an additional secret that is shared between the initiator and the responder; this secret is in addition to the authentication method that is already provided within IKEv2. We stir in this secret when generating the IKE keys (along with the

parameters that IKEv2 normally uses); this secret adds quantum resistance to the exchange.

It was considered important to minimize the changes to IKEv2. The existing mechanisms to do authentication and key exchange remain in

place (that is, we continue to do (EC)DH, and potentially a PKI authentication if configured). This does not replace the authentication checks that the protocol does; instead, it is done as a parallel check.

1.1. Changes

Changes in this draft from the previous versions

draft-00

- We switched from using vendor ID's to transmit the additional data to notifications

- We added a mandatory cookie exchange to allow the server to communicate to the client before the initial exchange

- We added algorithm agility by having the server tell the client what algorithm to use in the cookie exchange

- We have the server specify the PPK Indicator Input, which allows the server to make a trade-off between the efficiency for the search of the clients PPK, and the anonymity of the client.

- We now use the negotiated PRF (rather than a fixed HMAC-SHA256) to transform the nonces during the KDF

1.2. Requirements Language

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

2. Assumptions

We assume that each IKE peer (both the initiator and the responder)

has an optional Postquantum Preshared Key (PPK) (potentially on a per-peer basis), and also has a configurable flag that determines whether this postquantum preshared key is mandatory. This preshared key is independent of the preshared key (if any) that the IKEv2 protocol uses to perform authentication.

In addition, we assume that the initiator knows which PPK to use with the peer it is initiating to (for instance, if it knows the peer, then it can determine which PPK will be used).

3. Exchanges

If the initiator has a configured postquantum preshared key (whether or not it is optional), then it will include a notify payload in its initial exchange as follows:

```
Initiator                          Responder
-----------------------------------------------------------------
HDR, SAi1, KEi, Ni, N(PPK_REQUEST)  --->
```

N(PPK_REQUEST) is a status notification payload with the type [TBA]; it has a protocol ID of 0, and no SPI and no notification data associated with it.

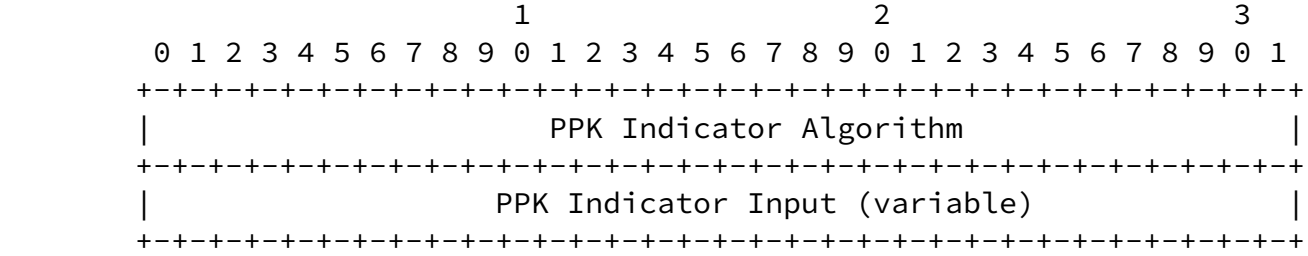
When the responder recieves the initial exchange with the notify payload, then (if it is configured to support PPK), it responds with:

```
Initiator                          Responder
-----------------------------------------------------------------
                            <--- HDR, N(COOKIE), N(PPK_ENCODE)
```

If it is not configured to support PPK, the responder continues with the standard IKEv2 protocol.

In other words, it asks for the responder to generate and send a cookie in its responses (as listed in section 2.6 of RFC7296), and in addition, include a notify that gives details of how the initiator should indicate what the PPK is. This notification payload has the type [TBA}; it has a protocol ID of 0, and no SPI; the notification

data is of the format:



The PPK Indicator Algorithm is a 4 byte word that states which PPK indicator to use. That is, it gives the encoding format for the PPK that should be used is given to the responder. At present, the only assigned encoding is 0x00000001, which indicates that AES256_SHA256 will be used (as explained below).

PPK Indicator Input is a data input to the PPK indicator Algorithm; its length will depend on the PPK indicator; for the indicator AES256_SHA256, this PPK Indicator Input is 16 bytes.

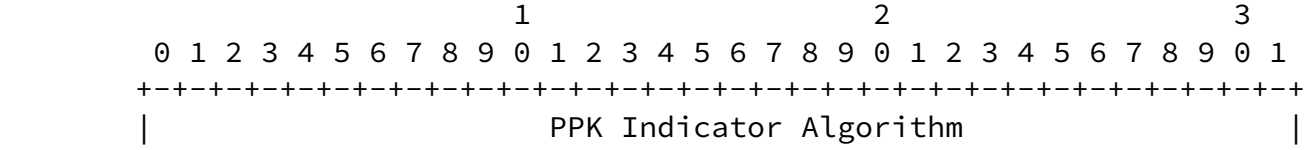
The contents of this PPK Indicator Input is selected by responder policy; below we give trade-offs of the various possibilities

When the initiator receives this notification, it responds as follows:

Initiator	Responder
HDR, N(COOKIE), SAi1, KEi, Ni, N(PPK_REQUEST)	--->

This is the standard IKEv2 cookie response, with a PPK_REQUEST notification added

N(PPK_REQUEST) is a status notification payload with the type [TBA]; it has a protocol ID of 0, and no SPI; however this time, the notification data as as follows:



```
|                    PPK Indicator Input (variable)             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       PPK Indicator (variable)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The PPK Indicator Algorithm and PPK Indicator Input are precisely the same as was given in the PPK_ENCODE format (as is repeated in case the responder ran this cookie protocol in a stateless manner). The PPK Indicator is the encoded version of the PPK that the initiator has. The idea behind this is to allow the responder to select which PPK it should use when it derives the IKEv2 keys.

For the AES256_SHA256 PPK indicator, the PPK Indicator is 16 bytes. To compute it, we use HMAC_SHA256(PPK, "A") as the 256 bit AES key to encrypt the 16 bytes on PPK Indicator Input (in ECB mode), where "A" is a string consisting of a single 0x41 octet.

When the responder receives this notification payload, it verifies that the PPK Indicator Algorithm is as it has specified, and it MAY verify that the PPK Indicator Input is as it has specified. If everything is on the level, it scans through its list of configured postquantum preshared keys, and determines which one it is (possibly (assuming AES256_SHA256_PPK) by computing AES256(HMAC_SHA256(PPK, "A"), PPK_Indicator_Input) and comparing that value to the 16 bytes within the payload. Alternatively, it may have preselected a PPK Indicator Input, and has precomputed (again assuming

AES256_SHA256_PPK) AES256(HMAC_SHA256(PPK, "A"), PPK_Indicator_Input) for each PPK it knows about (in which case, this is a simple search).

If the responder finds a value that matches the payload for a particular PPK, that indicates that the intiator and responder share a PPK and can make use of this extension. Upon finding such a preshared key, the responder includes a notification payload with the response:

```
Initiator                          Responder
-------------------------------------------------------------------
                     <--- HDR, SAr1, Ker, Nr, [CERTREQ], N(PPK_ACK)
```

N(PPK_ACK) is a status notification payload with the type [TBA]; it has a protocol ID of 0, and no SPI and no notification data

associated with it. This notification serves as a postquantum preshared key confirmation.

If the responder does not find such a PPK, then it MAY continue with the protocol without including a notification ID (if it is configured to not have mandatory preshared keys), or it MAY abort the exchange (if it configured to make preshared keys mandatory).

When the initiator receives the response, it MUST check for the presence of the notification. If it receives one, it marks the SA as using the configured preshared key; if it does not receive one, it MAY either abort the exchange (if the preshared key was configured as mandatory), or it MAY continue without using the preshared key (if the preshared key was configured as optional).

3.1. Computing SKEYSEED

When it comes time to generate the keying material during the initial Exchange, the implementation (both the initiator and the responder) checks to see if there was an agreed-upon preshared key. If there was, then both sides use this alternative formula:

```
SKEYSEED = prf(prf(PPK, Ni) | prf(PPK, Nr), g^ir)
(SK_d | SK_ai | SK_ar | SK_ei | SK_er | SK_pi | SK_pr) =
      prf+(SKEYSEED, prf(PPK, Ni) | prf(PPK, Nr) |
                          SPIi | SPIr)
```

where PPK is the postquantum preshared key, Ni, Nr are the nonces exchanged in the IKEv2 exchange, and prf is the pseudorandom function that was negotiated for this SA.

We reuse the negotiated PRF to transform the received nonces. We use this PRF, rather than negotiating a separate one, because this PRF is

agreed by both sides to have sufficient security properties (otherwise, they would have negotiated something else), and so that we don't need to specify a separate negotiation procedure.

3.2. Verifying preshared key

Once both the initiator and the responder have exchanged identities, they both double-check with their policy database to verify that they

were configured to use those preshared keys when negotiating with the peer. If they are not, they MUST abort the exchange.

3.3. Child SAs

When you create a child SA, the initiator and the responder will transform the nonces using the same PPK as they used during the original IKE SA negotiation. That is, they will use one of the alternative derivations (depending on whether an optional Diffie-Hellman was included):

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

or

```
KEYMAT = prf+(SK_d, g^ir (new) |
                      prf(PPK, Ni) | prf(PPK, Nr))
```

When you rekey an IKE SA (generating a fresh SKEYSEED), the initiator and the responder will transform the nonces using the same PPK as they used during the original IKE SA negotiation. That is, they will use the alternate derivation:

```
SKEYSEED = prf( SK_d (old), g^ir (new) |
                      prf(PPK, Ni) | prf(PPK, Nr))
(SK_d | SK_ai | SK_ar | SK_ei | SK_er | SK_pi | SK_pr) =
      prf+(SKEYSEED, prf(PPK, Ni) | prf(PPK, Nr) |
                        SPIi | SPIr)
```

4. Security Considerations

Quantum computers are able to perform Grover's algorithm; that effectively halves the size of a symmetric key. Because of this, the user SHOULD ensure that the postquantum preshared key used has at least 256 bits of entropy, in order to provide a 128 bit security level.

In addition, the policy SHOULD be set to negotiate only quantum-resistant symmetric algorithms (AES-256, SHA-256 or better).

The PPK Indicator Input within the PPK_ENCODE notification are there

to prevent anyone from deducing whether two different exchanges use the same PPK values. To prevent such a leakage, servers are encouraged to vary them as much as possible (however, they may want to repeat values to speed up the search for the PPK). Repeating these values places the anonymity at risk; however it has no other security implication.

5. References

5.1. Normative References

[AES] National Institute of Technology, "Specification for the Advanced Encryption Standard (AES)", 2001, <FIPS 197>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <http://www.rfc-editor.org/info/rfc2104>.

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

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

5.2. Informational References

[SPDP] McGrew, D., "A Secure Peer Discovery Protocol (SPDP)", 2001, <http://www.mindspring.com/~dmcgrew/spdp.txt>.

Appendix A. Discussion and Rationale

The idea behind this is that while a Quantum Computer can easily reconstruct the shared secret of an (EC)DH exchange, they cannot as easily recover a secret from a symmetric exchange this makes the SKEYSEED depend on both the symmetric PPK, and also the Diffie-Hellman exchange. If we assume that the attacker knows everything except the PPK during the key exchange, and there are 2**n plausible PPK's, then a Quantum Computer (using Grover's algorithm) would take O(2**(n/2)) time to recover the PPK. So, even if the (EC)DH can be trivially solved, the attacker still can't recover any key material unless they can find the PPK, and that's too difficult if the PPK has enough entropy (say, 256 bits).

Another goal of this protocol is to minimize the number of changes within the IKEv2 protocol, and in particular, within the cryptography of IKEv2. By limiting our changes to notifications, and translating the nonces, it is hoped that this would be implementable, even on systems that perform much of the IKEv2 processing is in hardware.

A third goal was to be friendly to incremental deployment in operational networks, for which we might not want to have a global shared key, and also if we're rolling this out incrementally. This is why we specifically try to allow the PPK to be dependent on the peer, and why we allow the PPK to be configured as optional.

A fourth goal was to avoid violating any of the security goals of IKEv2. One such goal is anonymity; that someone listening into the exchanges cannot easily determine who is negotiating with whom.

The third and fourth goals are in partial conflict. In order to achieve postquantum security, we need to stir in the PPK when the keys are computed, however the keys are computed before we know who we're talking to (and so which PPK we should use). And, we can't just tell the other side which PPK to use, as we might use different PPK's for different peers, and so that would violate the anonymity goal. If we just (for example) included a hash of the PPK, someone listening in could easily tell when we're using the same PPK for different exchanges, and thus deduce that the systems are related. The compromise we selected was to allow the responder to make the trade-off between anonymity and efficiency (by including the PPK Indicator Input, which varies how the PPK is encoded, and allowing the responder to specify it).

A responder who values anonymitity may select a random PPK Indicator Input each time; in this case, the responder needs to do a linear scan over all PPK's it has been configured with

A responder who can't afford a linear scan could precompute a small (possibly rolling) set of the PPK Indicator Inputs; in this case, it would precompute how each PPK would be indicated. If it reissues the same PPK Indicator Input to two different exchanges, someone would be able to verify whether the same PPK was used; this is some loss of anonymity; but is considerably more efficient.

An alternative approach to solve this problem would be to do a normal (non-QR) IKEv2 exchange, and when the two sides obtain identities, see if they need to be QR, and if so, create an immediate IKEv2 child SA (using the PPK). One issue with this is that someone with a quantum computer could deduce the identities used.

A slightly different approach to try to make this even more friendly to IKEv2-based cryptographic hardware might be to use invertible cryptography when we present the nonces to the kdf. The idea here is in case we have IKEv2 hardware that insists on selecting its own nonces (and so we won't be able to give a difference nonce to the KDF); instead, we encrypt the nonce that we send (and decrypt the nonce that we get). Of course, this means that the responder will need to figure out which PPK we're using up front (based on the notifications); we're not sure if this idea would be a net improvement (especially since the transform we're proposing now is cryptographically secure and simple).

The reasoning behind the cryptography used: the values we use in the AES256_SHA256 PPK Indicator Algorithm are cryptographically independent of the values used during the SKEYSEED generation (because, even if we use HMAC_256 as our PRF, HMAC_SHA256(PPK, A) is independent of HMAC_SHA256(PPK, B) if A and B are different strings (and as any real nonce must be longer than a single byte, there is never a collision between that and "A". This independent stems from the assumption that HMAC_SHA256 is a secure MAC.

The method of encoding the PPK within the notification (using AES-256) was chosen as it met two goals:

- Anonymity; given A, AES256_K1(A), B, AES256_K2(B), it's fairly obvious that gives someone (even if they have a quantum computer) no clue about whether K1==K2 (unless either A==B or AES256_K1(A)== AES256_K2(B); both highly unlikely events if A and B are chosen randomly).

- Performance during the linear search; a responder could preexpand the AES keys, and so comparing a potential PPK against a notification from the initiator would amount to performing a single AES block encryption and then doing a 16 byte comparison.

The first goal is considered important; one of the goals of IKEv2 is to provide anonymity. The second is considered important because the linear scan directly affects scalability. While this draft allows the server to gain performance at the cost of anonymity, it was

considered useful if we make the fully-anonymous method as attractive as possible. This use of AES makes this linear scan as cheap as possible (while preserving security).

We allow the responder to specify the PPK Indicator Algorithm; this was in response to requests for algorithm agility. At present, it appears unlikely that there would be a need for an additional encoding (as the current one is extremely conservative cryptographically); however the option is there.

The current draft forces a cookie exchange, and hence adds a round trip over the normal IKEv2 operation. This was done to allow the server to specify the PPK Indicator algorithm. While as additional round trip may seem costly, it does not invalidate this proposal, The reason for this proposal is to give an alternative to IKEv1 with preshared keys. While this additional round trip may seem costly, it is important to note that, even with the additional round trip, this proposal is still cheaper than IKEv1. Thus the mechanisms specified in this note meet the goal of providing a better alternative than relying on an obsolete version of the protocol for post quantum security.

One issue that is currently open: what should happen if the initiator guesses at the PPK Indicator Algorithm, selects a random PPK Indicator Input, and includes that in the initial message? After all, if the server follows the recommendation that the cookie exchange is stateless, and if the server chooses the PPK Indicator Input In randomly, it has no way to know that the client isn't running this protocol as specified. If the responder supports that PPK Indicator Algorithm, it could very well respond without forcing a cookie exchange (which would eliminate a message exchange round). It's not clear is whether we should endorse this mode of operation, and explicitly state that if the server recieves such an initial request, and it doesn't recognize the PPK Indicator Input, it should act like it recieved an iniital PPK_REQUEST.

Authors' Addresses

Scott Fluhrer
Cisco Systems

Email: sfluhrer@cisco.com

David McGrew
Cisco Systems

Email: mcgrew@cisco.com

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com