Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: March 13, 2016

S. Fluhrer
D. McGrew
P. Kampanakis
Cisco Systems

September 10, 2015

An Extension for Postquantum Security using Preshared Keys for IKEv2
draft-fluhrer-qr-ikev2-00

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 March 13, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction

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, then the resulting system is believed to be quantum resistant.

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, that is, believed to be invulnerable to an attacker with a Quantum Computer. 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 is 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. 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 vendor ID payload in its initial exchange as follows:

```
Initiator                       Responder
-----------------------------------------------------------------
HDR, SAi1, KEi, Ni, VID  --->
```

The contents of this vendor ID payload MUST consist of:

- 16 byte fixed vendor id
- 16 random bytes
- 16 bytes of AES256(HMAC_SHA256(ppk, "A"), random_bytes)

The 16 byte fixed vendor id consists of:

0x26, 0x9c, 0x82, 0x00, 0x36, 0x8a, 0xf5, 0x3b,
0x85, 0xd9, 0xde, 0x63, 0x6b, 0x3b, 0x29, 0xa4

this is the MD5 of "Quantum Resistant Secret Hash".

That is, we use HMAC_SHA256(ppk, "A") as the 256 bit AES key to encrypt the 16 random bytes (in ECB mode), where "A" is a string consisting of a single 0x41 octet.

When the responder receives this vendor ID, it scans through its list of configured postquantum preshared keys, and determines which one it is (by computing AES256(HMAC(ppk, "A"), Nonce) and comparing that value to the 16 bytes within the payload.

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 vendor ID payload with the response:

```
Initiator                        Responder
-------------------------------------------------------------------
                          <--- HDR, SAr1, Ker, Nr, [CERTREQ], VID
```

The contents of this vendor ID payload MUST consist of:

o 16 byte fixed vendor id

The 16 byte fixed vendor id consists of:

0x26, 0x9c, 0x82, 0x00, 0x36, 0x8a, 0xf5, 0x3b,
0x85, 0xd9, 0xde, 0x63, 0x6b, 0x3b, 0x29, 0xa4

this is the MD5 of "Quantum Resistant Secret Hash".

The random value and its encryption are not included in the VID this time. This VID serves as a postquantum preshared key confirmation.

If the responder does not find such a preshared key, then it MAY continue with the protocol without including a vendor 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 KUST check for the presence of the vendor ID. 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(HMAC_SHA256(ppk, Ni) | HMAC_SHA256(ppk, Nr), g^ir)
(SK_d | SK_ai | SK_ar | SK_ei | SK_er | SK_pi | SK_pr) =
      prf+(SKEYSEED, HMAC_SHA256(ppk, Ni) | HMAC_SHA256(ppk, Nr) |
                          SPIi | SPIr)
```

where ppk is the postquantum preshared key, Ni, Nr are the nonces exchanged in the IKEv2 exchange (and not the nonces used in the vendor id;apos;s), HMAC_SHA256(a, b) uses 'a' as the key, and 'b' as the text, and g^ir is the Diffie-Hellman shared secret.

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, HMAC_SHA256(ppk, Ni) | HMAC_SHA256(ppk, Nr))
```

or

```
 KEYMAT = prf+(SK_d, g^ir (new) |
                       HMAC_SHA256(ppk, Ni) | HMAC_SHA256(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) |
                       HMAC_SHA256(ppk, Ni) | HMAC_SHA256(ppk, Nr))
 (SK_d | SK_ai | SK_ar | SK_ei | SK_er | SK_pi | SK_pr) =
       prf+(SKEYSEED, HMAC_SHA256(ppk, Ni) | HMAC_SHA256(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 random values within the vendor ID are there to prevent anyone from deducing whether two different exchanges use the same ppk values. To prevent such a leakage, every exchange SHOULD use a fresh 16 byte random value. Violating this places the anonymity at risk; however it has no other security implication.

5. 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>.

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 vendor id's, 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 include enough information that someone who knows the ppk can recognize it, however someone who doesn't know the ppk learns nothing. However, one issue with this is that the responder needs to do a linear scan over all ppk's it has been configured with; this is not ideal, but it's the best compromise we can come up with. And, the current protocol (of having the initiator send an R, Enc(R) pair in the vendor id) doesn't allow anyone who doesn't know the vendor id have no information whether two exchanges use the same ppk or not.

An alternative approach to solve this problem without a linear scan 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

vendor id); 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 vendor id's are cryptographically independent of the values used during the SKEYSEED generation (because 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 \quot;A\quot;. This independent stems from the assumption that SHA-256 is a secure MAC. This was chosen over more ad hoc designs where the two uses of the ppk would appear to be independent (but that doesn't follow from any standard cryptographical assumption. The method of encoding the ppk within the vendor id (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 server could preexpand the AES keys, and so comparing a potential ppk against an vendor id 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 requires a linear scan over all ppk's known by the responder (it is unknown how to avoid this without leaking when the same ppk is being negotiated by two different exchanges), this use of AES makes this linear scan as cheap as possible. We don't know how to avoid the linear scan, so making the scan cheap (while not compromising on security) was considered important.

One thing that this draft does not address is algorithm agility; it specifies that we'll use HMAC-SHA256 and AES256, and does not allow any alternatives. This might change in a latter version of this draft.

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