Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: October 21, 2017

S. Fluhrer
D. McGrew
P. Kampanakis
Cisco Systems

April 19, 2017

Postquantum Preshared Keys for IKEv2
draft-fluhrer-qr-ikev2-04

Abstract

The possibility of quantum computers pose a serious challenge to cryptography algorithms widely today. IKEv2 is one example of a cryptosystem that could be broken; someone storing VPN communications today could decrypt them at a later time when a quantum computer is available. It is anticipated that IKEv2 will be extended to support quantum secure key exchange algorithms; however that is not likely to happen in the near term. To address this problem before then, 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

Copyright Notice

Table of Contents

1. Introduction

It is an open question whether or not it is feasible to build a quantum computer (and if so, when might one be implemented), 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 strong preshared keys is not vulnerable to quantum attacks, because those keys are one of the inputs to the key derivation function. If the preshared key has sufficient entropy and the PRF, 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 into the SK_d value, which is used to generate the key material (KEYMAT) keys and the SKEYSEED for the child SAs; this secret provides quantum resistance to the IPsec SAs (and any child IKE SAs). We also stir in the secret into the SK_pi, SK_pr values; this allows both sides to detect a secret mismatch cleanly.

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-03

- Modified how we stir the PPK into the IKEv2 secret state

- Modified how the use of PPKs is negotiated

draft-02

- Simplified the protocol by stirring in the preshared key into the child SAs; this avoids the problem of having the responder decide which preshared key to use (as it knows the initiator identity at that point); it does mean that someone with a Quantum Computer can recover the initial IKE negotation.

- Removed positive endorsements of various algorithms. Retained warnings about algorithms known to be weak against a Quantum Computer

draft-01

- Added explicit guidance as to what IKE and IPsec algorithms are Quantum Resistant

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 has a list of Postquantum Preshared Keys (PPK) along with their identifiers (PPK_id), and any potential IKE initiator has a selection of which PPK to use with with any specific responder. In addition, the implementation has a configurable flag that determines whether this postquantum preshared key is mandatory. This PPK is independent of the preshared key (if any) that the IKEv2 protocol uses to perform authentication.

3. Exchanges

If the initiator is configured to use a postquantum preshared key with the responder (whether or not the use of the PPK is optional), then it will include a notify payload in the initial exchange as follows:

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

N(PPK_SUPPORT) 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.

If the initiator needs to resend this initial message with a cookie (because the responder response included a cookie notification), then the resend would include the PPK_SUPPORT notification if the original message did.

When the responder receives this initial exchange with the notify, then it MUST check if has a PPK configured. If it does, it MUST reply with the IKE initial exchange including a notification in response.

```
Initiator                       Responder
------------------------------------------------------------------
                <--- HDR, SAr1, KEr, Nr, [CERTREQ], N(PPK_SUPPORT)
```

If the responder does not have a PPK configured, then it continues with the IKE protocol as normal, not including the notify.

When the initiator receives this reply, it checks whether the responder included the PPK_SUPPORT notify. If the responder did not, then the initiator MUST either proceed with the standard IKE negotiation (without using a PPK), or abort the exchange (for example, because the initiator has the PPK marked as mandatory). If the responder did include the PPK_SUPPORT notify, then it selects a PPK, along with its identifier PPK_id. Then, it computes this modification of the standard IKE key derivation:

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

That is, we use the standard IKE key derivation process except that the three subkeys SK_d, SK_pi, SK_pr are run through the prf again, this time using the PPK as the key.

The initiator then sends the initial encrypted message, including the PPK_id value as follows:

```
Initiator                       Responder
------------------------------------------------------------------
HDR, SK {IDi, [CERT,] [CERTREQ,]
    [IDr,] AUTH, SAi2,
    TSi, TSr, N(PPK_IDENTITY)(PPK_id)}  --->
```

N(PPK_IDENITY) is a status notification payload with the type [TBA]; it has a protocol ID of 0, and no SPI and has a notification data that consists of the identifier PPK_id.

When the responder receives this encrypted exchange, it first computes the values:

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

It then uses the SK_ei value to decrypt the message; and then finds the PPK_id value attached to the notify. It then scans through the payload for the PPK_id attached to the N(PPK_IDENTITY); if it has no such PPK, it fails the negotiation. If it does have a PPK with that identity, it further computes:

```
    SK_d = prf(PPK, SK_d')
    SK_pi = prf(PPK, SK_pi')
    SK_pr = prf(PPK, SK_pr')
```

And computes the enchange (validating the AUTH payload that the initiator included) as standard.

This table summarizes the above logic by the responder

Received PPK_SUPPORT	Have PPK	PPK Mandatory	Action
No	No	*	Standard IKE protocol
No	Yes	No	Standard IKE protocol
No	Yes	Yes	Abort negotiation
Yes	No	*	Standard IKE protocol
Yes	Yes	*	Include PPK_SUPPORT

When the initiator receives the response, then (if it is configured to use a PPK with the responder), then it checks for the presense of the notification. If it receives one, it marks the SA as using the configured PPK to generate SK_d, SK_pi, SK_pr (as shown above); if it does not receive one, it MUST either abort the exchange (if the PPK was configured as mandatory), or it MUST continue without using the PPK (if the PPK was configured as optional).

If the initial exchange had PPK_SUPPORT sent by both the initiator and the responder, and the initiator does not include a PPK_NOTIFY notification, then the responder SHOULD fail the exchange.

With this protocol, the computed SK_d is a function of the PPK, and assuming that the PPK has sufficient entropy (for example, at least 2**256 possible values), then even if an attacker were able to recover the rest of the inputs to the prf function, it would be infeasible to use Grover's algorithm with a Quantum Computer to recover the SK_d value. Similarly, every child SA key is a function of SK_d, hence all the keys for all the child SAs are also quantum resistant (assuming that the PPK was high entropy and secret, and that all the subkeys are sufficiently long). However, this quantum resistance does not extend to the initial SK_ei, SK_er keys; an implementation MAY rekey the initial IKE SA immediately after negotiating it; this would reduce the amount of data available to an attacker with a Quantum Computer.

4. PPK ID format

This standard requires that both the initiator and the responder have a secret PPK value, with the responder selecting the PPK based on the PPK_ID that the initiator sends. In this initial standard, both the initator and the responder are configured with fixed PPK and PPK_ID values, and do the look up based on that. It is anticipated that later standards will extend this technique to allow dynamically changing PPK values. To facilitate such an extension, we specify that the PPK_ID that the initiator sends will have its first octet be the PPK ID Type value, which is encoded as follows:

PPK ID Type	Value
PPK_ID_OPAQUE	0
PPK_ID_FIXED	1
RESERVED TO IANA	2-127
Reserved for private use	128-255

For PPK_ID_OPAQUE, the format of the PPK ID (and the PPK itself) is not specified by this document; it is assumed to be mutually intelligible by both by initiator and the responder. This PPK ID type is intended for those implementations that choose not to disclose the type of PPK to active attackers.

For PPK_ID_FIXED, the format of the PPK ID and the PPK are fixed octet strings; the remaining bytes of the PPK_ID are a configured value. We assume that there is a fixed mapping between PPK_ID and PPK, which is configured locally to both the initiator and the responder. The responder can use to do a look up the passed PPK_id value to determine the corresponding PPK value. Not all implementations are able to configure arbitrary octet strings; to improve the potential interoperability, it is recommended that, in the PPK_ID_FIXED case, both the PPK and the PPK_ID strings be limited to the base64 character set, namely the 64 characters 0-9, A-Z, a-z, + and /.

The PPK ID type values 2-127 are reserved for IANA; values 128-255 are for private use among mutually consenting parties.

5. PPK Distribution

PPK_id's of the type PPK_ID_FIXED (and the corresponding PPKs) are assumed to be configured within the IKE device in an out-of-band fashion. While the method of distribution is a local matter, one suggestion would be to reuse the format within [RFC6030], with the Key Id field being the PPK_ID (without the 0x01 prefix for a PPK_ID_FIXED), and with the PPK being the secret, and the algorithm as PIN ("Algorithm=urn:ietf:params:xml:ns:keyprov:pskc:pin").

6. Upgrade procedure

This algorithm was designed so that someone can introduce PPKs into an existing IKE network without causing network disruption.

In the initial phase of the network upgrade, the network administrator would visit each IKE node, and configure:

- The set of PPKs (and corresponding PPK_id's) that this node would need to know

- For each peer that this node would initiate to, which PPK that we would use

- That the use of PPK is currently optional

With this configuration, the node will continue to operate with nodes that have not yet been upgraded. This is due to the PPK_SUPPORT notify; if the initiator has not been upgraded, it will not send the PPK_SUPPORT notify (and so the responder will know that we will not use a PPK); if the responder has not been upgraded, it will not send the PPK_SUPPORT notify (and so the initiator will know not to use a PPK). And, if both peers have been upgraded, they will both realize it, and in that case, the link will be quantum secure

As an optional second step, after all nodes have been upgraded, then the administrator may then go back through the nodes, and mark the use of PPK as mandatory. This will not affect the strength against a passive attacker; it would mean that an attacker with a Quantum Computer (which is sufficiently fast to be able to break the (EC)DH in real time would not be able to perform a downgrade attack).

7. 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.

Although this protocol preserves all the security properties of IKE against adversaries with conventional computers, this protocol allows an adversary with a Quantum Computer to decrypt all traffic encrypted with the initial IKE SA. In particular, it allows the adversary to recover the identities of both sides. If there is IKE traffic other than the identities that need to be protected against such an adversary, one suggestion would be to form an initial IKE SA (which is used to exchange identities), perhaps by using the protocol documented in RFC6023. Then, you would immediately create a child IKE SA (which is used to exchange everything else). Because the child IKE SA keys are a function of SK_d, which is a function of the PPK (among other things), traffic protected by that SA is secure against Quantum capable adversaries.

In addition, the policy SHOULD be set to negotiate only quantum-resistant symmetric algorithms; while this RFC doesn't claim to give advise as to what algorithms are secure (as that may change based on future cryptographical results), here is a list of defined IKEv2 and IPsec algorithms that should NOT be used, as they are known not to be Quantum Resistant

Any IKE Encryption algorithm, PRF or Integrity algorithm with key size <256 bits

Any ESP Transform with key size <256 bits

PRF_AES128_XCBC and PRF_AES128_CBC; even though they are defined to be able to use an arbitrary key size, they convert it into a 128 bit key internally

8. References

8.1. Normative References

[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>.

8.2. Informational References

[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, <http://www.rfc-editor.org/info/rfc6023>.

[RFC6030] Hoyer, P., Pei, M., and S. Machani, "Portable Symmetric Key Container (PSKC)", RFC 6030, DOI 10.17487/RFC6030, October 2010, <http://www.rfc-editor.org/info/rfc6030>.

[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 SK_d, and hence the IPsec KEYMAT and any child SA's 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 (except for the SK_ei, SK_er, SK_ai, SK_ar values for the initial IKE exchange) unless they can find the PPK, and that's too difficult if the PPK has enough entropy (for example, 256 bits). Note that we do allow an attacker with a Quantum Computer to rederive the keying material for the initial IKE SA; this was a compromise to allow the responder to select the correct PPK quickly.

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.

Appendix B. Acknowledgement

We would like to thank Tero Kivine, Valery Smyslov, Paul Wouters and the rest of the ipsecme working group for their feedback and suggestions for the scheme

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

IPSecME Working Group Y. Nir
Internet-Draft Check Point
Intended status: Standards Track V. Smyslov
Expires: April 4, 2017 ELVIS-PLUS
October 1, 2016

Protecting Internet Key Exchange Protocol version 2 (IKEv2) Implementations from Distributed Denial of Service Attacks
draft-ietf-ipsecme-ddos-protection-10

Abstract

This document recommends implementation and configuration best practices for Internet Key Exchange Protocol version 2 (IKEv2) Responders, to allow them to resist Denial of Service and Distributed Denial of Service attacks. Additionally, the document introduces a new mechanism called "Client Puzzles" that help accomplish this task.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Denial of Service (DoS) attacks have always been considered a serious threat. These attacks are usually difficult to defend against since the amount of resources the victim has is always bounded (regardless of how high it is) and because some resources are required for distinguishing a legitimate session from an attack.

The Internet Key Exchange protocol version 2 (IKEv2) described in [RFC7296] includes defense against DoS attacks. In particular, there is a cookie mechanism that allows the IKE Responder to defend itself against DoS attacks from spoofed IP-addresses. However, botnets have become widespread, allowing attackers to perform Distributed Denial of Service (DDoS) attacks, which are more difficult to defend against. This document presents recommendations to help the Responder counter (D)DoS attacks. It also introduces a new mechanism -- "puzzles" -- that can help accomplish this task.

2. Conventions Used in This Document

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

3. The Vulnerability

The IKE_SA_INIT Exchange described in Section 1.2 of [RFC7296] involves the Initiator sending a single message. The Responder replies with a single message and also allocates memory for a structure called a half-open IKE Security Association (SA). This half-open SA is later authenticated in the IKE_AUTH Exchange. If that IKE_AUTH request never comes, the half-open SA is kept for an unspecified amount of time. Depending on the algorithms used and implementation, such a half-open SA will use from around 100 bytes to several thousands bytes of memory.

This creates an easy attack vector against an IKE Responder. Generating the IKE_SA_INIT request is cheap. Sending large amounts of IKE_SA_INIT requests can cause a Responder to use up all its resources. If the Responder tries to defend against this by throttling new requests, this will also prevent legitimate Initiators from setting up IKE SAs.

An obvious defense, which is described in Section 4.2, is limiting the number of half-open SAs opened by a single peer. However, since all that is required is a single packet, an attacker can use multiple spoofed source IP addresses.

If we break down what a Responder has to do during an initial exchange, there are three stages:

1. When the IKE_SA_INIT request arrives, the Responder:

   - Generates or re-uses a Diffie-Hellman (D-H) private part.

   - Generates a Responder Security Parameter Index (SPI).

   - Stores the private part and peer public part in a half-open SA database.

2. When the IKE_AUTH request arrives, the Responder:

   - Derives the keys from the half-open SA.

   - Decrypts the request.

3. If the IKE_AUTH request decrypts properly:

   - Validates the certificate chain (if present) in the IKE_AUTH request.

The fourth stage where the Responder creates the Child SA is not reached by attackers who cannot pass the authentication step.

Stage #1 is pretty light on CPU power, but requires some storage, and it's very light for the Initiator as well. Stage #2 includes private-key operations, so it is much heavier CPU-wise. Stage #3 may include public key operations if certificates are involved. These operations are often more computationly expensive than those performed at stage #2.

To attack such a Responder, an attacker can attempt either to exhaust memory or to exhaust CPU. Without any protection, the most efficient attack is to send multiple IKE_SA_INIT requests and exhaust memory. This is easy because IKE_SA_INIT requests are cheap.

There are obvious ways for the Responder to protect itself without changes to the protocol. It can reduce the time that an entry remains in the half-open SA database, and it can limit the amount of concurrent half-open SAs from a particular address or prefix. The attacker can overcome this by using spoofed source addresses.

The stateless cookie mechanism from Section 2.6 of [RFC7296] prevents an attack with spoofed source addresses. This doesn't completely solve the issue, but it makes the limiting of half-open SAs by address or prefix work. Puzzles, introduced in Section 4.4, accomplish the same thing only more of it. They make it harder for an attacker to reach the goal of getting a half-open SA. Puzzles do not have to be so hard that an attacker cannot afford to solve a single puzzle; it is enough that puzzles increase the cost of creating a half-open SAs, so the attacker is limited in the amount they can create.

Reducing the lifetime of an abandoned half-open SA also reduces the impact of such attacks. For example, if a half-open SA is kept for 1 minute and the capacity is 60 thousand half-open SAs, an attacker would need to create one thousand half-open SAs per second. If the retention time is reduced to 3 seconds, the attacker would need to create 20 thousand half-open SAs per second to get the same result. By introducing a puzzle, each half-open SA becomes more expensive for an attacker, making it more likely to prevent an exhaustion attack against Responder memory.

At this point, filling up the half-open SA database is no longer the most efficient DoS attack. The attacker has two alternative attacks to do better:

1. Go back to spoofed addresses and try to overwhelm the CPU that deals with generating cookies, or

2. Take the attack to the next level by also sending an IKE_AUTH request.

If an attacker is so powerfull that it is able to overwhelm the Responder's CPU that deals with generating cookies, then the attack cannot be dealt with at the IKE level and must be handled by means of the Intrusion Prevention System (IPS) technology.

On the other hand, the second alternative of sending an IKE_AUTH request is very cheap. It requires generating a proper IKE header with the correct IKE SPIs and a single Encrypted payload. The content of the payload is irrelevant and might be junk. The Responder has to perform the relatively expensive key derivation, only to find that the MAC on the Encrypted payload on the IKE_AUTH request fails the integrity check. If a Responder does not hold on to the calculated SKEYSEED and SK_* keys (which it should in case a valid IKE_AUTH comes in later) this attack might be repeated on the same half-open SA. Puzzles make attacks of such sort more costly for an attacker. See Section 7.2 for details.

Here too, the number of half-open SAs that the attacker can achieve is crucial, because each one allows the attacker to waste some CPU time. So making it hard to make many half-open SAs is important.

A strategy against DDoS has to rely on at least 4 components:

1. Hardening the half-open SA database by reducing retention time.

2. Hardening the half-open SA database by rate-limiting single IPs/ prefixes.

3. Guidance on what to do when an IKE_AUTH request fails to decrypt.

4. Increasing the cost of half-open SAs up to what is tolerable for legitimate clients.

Puzzles are used as a solution for strategy #4.

4. Defense Measures while the IKE SA is being created

4.1. Retention Periods for Half-Open SAs

As a UDP-based protocol, IKEv2 has to deal with packet loss through retransmissions. Section 2.4 of [RFC7296] recommends "that messages be retransmitted at least a dozen times over a period of at least several minutes before giving up". Many retransmission policies in practice wait one or two seconds before retransmitting for the first time.

Because of this, setting the timeout on a half-open SA too low will cause it to expire whenever even one IKE_AUTH request packet is lost. When not under attack, the half-open SA timeout SHOULD be set high enough that the Initiator will have enough time to send multiple retransmissions, minimizing the chance of transient network congestion causing an IKE failure.

When the system is under attack, as measured by the amount of half-open SAs, it makes sense to reduce this lifetime. The Responder should still allow enough time for the round-trip, enough time for the Initiator to derive the D-H shared value, and enough time to derive the IKE SA keys and the create the IKE_AUTH request. Two seconds is probably as low a value as can realistically be used.

It could make sense to assign a shorter value to half-open SAs originating from IP addresses or prefixes that are considered suspect because of multiple concurrent half-open SAs.

4.2. Rate Limiting

Even with DDoS, the attacker has only a limited amount of nodes participating in the attack. By limiting the amount of half-open SAs that are allowed to exist concurrently with each such node, the total amount of half-open SAs is capped, as is the total amount of key derivations that the Responder is forced to complete.

In IPv4 it makes sense to limit the number of half-open SAs based on IP address. Most IPv4 nodes are either directly attached to the Internet using a routable address or are hidden behind a NAT device with a single IPv4 external address. For IPv6, ISPs assign between a /48 and a /64, so it does not make sense for rate-limiting to work on single IPv6 IPs. Instead, ratelimits should be done based on either the /48 or /64 of the misbehaving IPv6 address observed.

The number of half-open SAs is easy to measure, but it is also worthwhile to measure the number of failed IKE_AUTH exchanges. If possible, both factors should be taken into account when deciding which IP address or prefix is considered suspicious.

There are two ways to rate-limit a peer address or prefix:

1. Hard Limit - where the number of half-open SAs is capped, and any further IKE_SA_INIT requests are rejected.

2. Soft Limit - where if a set number of half-open SAs exist for a particular address or prefix, any IKE_SA_INIT request will be required to solve a puzzle.

The advantage of the hard limit method is that it provides a hard cap on the amount of half-open SAs that the attacker is able to create. The disadvantage is that it allows the attacker to block IKE initiation from small parts of the Internet. For example, if an network service provider or some establishment offers Internet connectivity to its customers or employees through an IPv4 NAT device, a single malicious customer can create enough half-open SAs to fill the quota for the NAT device external IP address. Legitimate Initiators on the same network will not be able to initiate IKE.

The advantage of a soft limit is that legitimate clients can always connect. The disadvantage is that an adversary with sufficient CPU resources can still effectively DoS the Responder.

Regardless of the type of rate-limiting used, legitimate initiators that are not on the same network segments as the attackers will not be affected. This is very important as it reduces the adverse impact caused by the measures used to counteract the attack, and allows most initiators to keep working even if they do not support puzzles.

4.3. The Stateless Cookie

Section 2.6 of [RFC7296] offers a mechanism to mitigate DoS attacks: the stateless cookie. When the server is under load, the Responder responds to the IKE_SA_INIT request with a calculated "stateless cookie" - a value that can be re-calculated based on values in the

IKE_SA_INIT request without storing Responder-side state. The Initiator is expected to repeat the IKE_SA_INIT request, this time including the stateless cookie. This mechanism prevents DoS attacks from spoofed IP addresses, since an attacker needs to have a routable IP address to return the cookie.

Attackers that have multiple source IP addresses with return routability, such as in the case of botnets, can fill up a half-open SA table anyway. The cookie mechanism limits the amount of allocated state to the number of attackers, multiplied by the number of half-open SAs allowed per peer address, multiplied by the amount of state allocated for each half-open SA. With typical values this can easily reach hundreds of megabytes.

4.4. Puzzles

The puzzle introduced here extends the cookie mechanism of [RFC7296]. It is loosely based on the proof-of-work technique used in Bitcoins [bitcoins]. Puzzles set an upper bound, determined by the attacker's CPU, to the number of negotiations the attacker can initiate in a unit of time.

A puzzle is sent to the Initiator in two cases:

- The Responder is so overloaded that no half-open SAs may be created without solving a puzzle, or

- The Responder is not too loaded, but the rate-limiting method described in Section 4.2 prevents half-open SAs from being created with this particular peer address or prefix without first solving a puzzle.

When the Responder decides to send the challenge to solve a puzzle in response to a IKE_SA_INIT request, the message includes at least three components:

1. Cookie - this is calculated the same as in [RFC7296], i.e. the process of generating the cookie is not specified.

2. Algorithm, this is the identifier of a Pseudo-Random Function (PRF) algorithm, one of those proposed by the Initiator in the SA payload.

3. Zero Bit Count (ZBC). This is a number between 8 and 255 (or a special value - 0, see Section 7.1.1.1) that represents the length of the zero-bit run at the end of the output of the PRF function calculated over the cookie that the Initiator is to send. The values 1-8 are explicitly excluded, because they

create a puzzle that is too easy to solve. Since the mechanism is supposed to be stateless for the Responder, either the same ZBC is used for all Initiators, or the ZBC is somehow encoded in the cookie. If it is global then it means that this value is the same for all the Initiators who are receiving puzzles at any given point of time. The Responder, however, may change this value over time depending on its load.

Upon receiving this challenge, the Initiator attempts to calculate the PRF output using different keys. When enough keys are found such that the resulting PRF output calculated using each of them has a sufficient number of trailing zero bits, that result is sent to the Responder.

The reason for using several keys in the results, rather than just one key, is to reduce the variance in the time it takes the initiator to solve the puzzle. We have chosen the number of keys to be four (4) as a compromise between the conflicting goals of reducing variance and reducing the work the Responder needs to perform to verify the puzzle solution.

When receiving a request with a solved puzzle, the Responder verifies two things:

o That the cookie is indeed valid.

o That the results of PRF of the transmitted cookie calculated with the transmitted keys has a sufficient number of trailing zero bits.

Example 1: Suppose the calculated cookie is 739ae7492d8a810cf5e8dc0f9626c9dda773c5a3 (20 octets), the algorithm is PRF-HMAC-SHA256, and the required number of zero bits is 18. After successively trying a bunch of keys, the Initiator finds the following four 3-octet keys that work:

Key	Last 32 Hex PRF Digits	# 0-bits
061840	e4f957b859d7fb1343b7b94a816c0000	18
073324	0d4233d6278c96e3369227a075800000	23
0c8a2a	952a35d39d5ba06709da43af40700000	20
0d94c8	5a0452b21571e401a3d00803679c0000	18

Table 1: Four solutions for the 18-bit puzzle

Example 2: Same cookie, but modify the required number of zero bits to 22. The first 4-octet keys that work to satisfy that requirement are 005d9e57, 010d8959, 0110778d, and 01187e37. Finding these requires 18,382,392 invocations of the PRF.

# 0-bits	Time to Find 4 keys (seconds)
8	0.0025
10	0.0078
12	0.0530
14	0.2521
16	0.8504
17	1.5938
18	3.3842
19	3.8592
20	10.8876

Table 2: The time needed to solve a puzzle of various difficulty for the cookie = 739ae7492d8a810cf5e8dc0f9626c9dda773c5a3

The figures above were obtained on a 2.4 GHz single core i5. Run times can be halved or quartered with multi-core code, but would be longer on mobile phone processors, even if those are multi-core as well. With these figures 18 bits is believed to be a reasonable choice for puzzle level difficulty for all Initiators, and 20 bits is acceptable for specific hosts/prefixes.

Using puzzles mechanism in the IKE_SA_INIT exchange is described in Section 7.1.

4.5. Session Resumption

When the Responder is under attack, it SHOULD prefer previously authenticated peers who present a Session Resumption ticket [RFC5723]. However, the Responder SHOULD NOT serve resumed Initiators exclusively because dropping all IKE_SA_INIT requests would lock out legitimate Initiators that have no resumption ticket. When under attack the Responder SHOULD require Initiators presenting Session Resumption Tickets to pass a return routability check by including the COOKIE notification in the IKE_SESSION_RESUME response message, as described in Section 4.3.2. of [RFC5723]. Note that the Responder SHOULD cache tickets for a short time to reject reused tickets (Section 4.3.1), and therefore there should be no issue of half-open SAs resulting from replayed IKE_SESSION_RESUME messages.

Several kinds of DoS attacks are possible on servers supported IKE Session Resumption. See Section 9.3 of [RFC5723] for details.

4.6. Keeping computed Shared Keys

Once the IKE_SA_INIT exchange is finished, the Responder is waiting for the first message of the IKE_AUTH exchange from the Initiator. At this point the Initiator is not yet authenticated, and this fact allows an attacker to perform an attack, described in Section 3. Instead of sending properly formed and encrypted IKE_AUTH message the attacker can just send arbitrary data, forcing the Responder to perform costly CPU operations to compute SK_* keys.

If the received IKE_AUTH message failed to decrypt correctly (or failed to pass ICV check), then the Responder SHOULD still keep the computed SK_* keys, so that if it happened to be an attack, then an attacker cannot get advantage of repeating the attack multiple times on a single IKE SA. The responder can also use puzzles in the IKE_AUTH exchange as decribed in Section 7.2.

4.7. Preventing "Hash and URL" Certificate Encoding Attacks

In IKEv2 each side may use the "Hash and URL" Certificate Encoding to instruct the peer to retrieve certificates from the specified location (see Section 3.6 of [RFC7296] for details). Malicious initiators can use this feature to mount a DoS attack on the responder by providing an URL pointing to a large file possibly containing meaningless bits. While downloading the file the responder consumes CPU, memory and network bandwidth.

To prevent this kind of attack, the responder should not blindly download the whole file. Instead, it SHOULD first read the initial few bytes, decode the length of the ASN.1 structure from these bytes, and then download no more than the decoded number of bytes. Note, that it is always possible to determine the length of ASN.1 structures used in IKEv2, if they are DER-encoded, by analyzing the first few bytes. However, since the content of the file being downloaded can be under the attacker’s control, implementations should not blindly trust the decoded length and SHOULD check whether it makes sense before continuing to download the file. Implementations SHOULD also apply a configurable hard limit to the number of pulled bytes and SHOULD provide an ability for an administrator to either completely disable this feature or to limit its use to a configurable list of trusted URLs.

4.8. IKE Fragmentation

IKE Fragmentation described in [RFC7383] allows IKE peers to avoid IP fragmentation of large IKE messages. Attackers can mount several kinds of DoS attacks using IKE Fragmentation. See Section 5 of [RFC7383] for details on how to mitigate these attacks.

5. Defense Measures after an IKE SA is created

Once an IKE SA is created there usually are only a limited amount of IKE messages exchanged. This IKE traffic consists of exchanges aimed to create additional Child SAs, IKE rekeys, IKE deletions and IKE liveness tests. Some of these exchanges require relatively little resources (like liveness check), while others may be resource consuming (like creating or rekeying Child SA with D-H exchange).

Since any endpoint can initiate a new exchange, there is a possibility that a peer would initiate too many exchanges that could exhaust host resources. For example, the peer can perform endless continuous Child SA rekeying or create an overwhelming number of Child SAs with the same Traffic Selectors etc. Such behavior can be caused by broken implementations, misconfiguration, or as an intentional attack. The latter becomes more of a real threat if the peer uses NULL Authentication, as described in [RFC7619]. In this case the peer remains anonymous, allowing it to escape any responsibility for its behaviour. See Section 3 of [RFC7619] for details on how to mitigate attacks when using NULL Authentication.

The following recommendations apply especially for NULL Authenticated IKE sessions, but also apply to authenticated IKE sessions, with the difference that in the latter case, the identified peer can be locked out.

- If the IKEv2 window size is greater than one, peers are able to initiate multiple simultaneous exchanges that increase host resource consumption. Since there is no way in IKEv2 to decrease window size once it has been increased (see Section 2.3 of [RFC7296]), the window size cannot be dynamically adjusted depending on the load. It is NOT RECOMMENDED to allow an IKEv2 window size greater than one when NULL Authentication has been used.

- If a peer initiates an abusive amount of CREATE_CHILD_SA exchanges to rekey IKE SAs or Child SAs, the Responder SHOULD reply with TEMPORARY_FAILURE notifications indicating the peer must slow down their requests.

- If a peer creates many Child SA with the same or overlapping Traffic Selectors, implementations MAY respond with the NO_ADDITIONAL_SAS notification.

- If a peer initiates many exchanges of any kind, the Responder MAY introduce an artificial delay before responding to each request message. This delay would decrease the rate the Responder needs to process requests from any particular peer, and frees up resources on the Responder that can be used for answering legitimate clients. If the Responder receives retransmissions of the request message during the delay period, the retransmitted messages MUST be silently discarded. The delay must be short enough to avoid legitimate peers deleting the IKE SA due to a timeout. It is believed that a few seconds is enough. Note however, that even a few seconds may be too long when settings rely on an immediate response to the request message, e.g. for the purposes of quick detection of a dead peer.

- If these counter-measures are inefficient, implementations MAY delete the IKE SA with an offending peer by sending Delete Payload.

In IKE, a client can request various configuration attributes from server. Most often these attributes include internal IP addresses. Malicious clients can try to exhaust a server's IP address pool by continuously requesting a large number of internal addresses. Server implementations SHOULD limit the number of IP addresses allocated to any particular client. Note, this is not possible with clients using NULL Authentication, since their identity cannot be verified.

6. Plan for Defending a Responder

This section outlines a plan for defending a Responder from a DDoS attack based on the techniques described earlier. The numbers given here are not normative, and their purpose is to illustrate the configurable parameters needed for surviving DDoS attacks.

Implementations are deployed in different environments, so it is RECOMMENDED that the parameters be settable. For example, most commercial products are required to undergo benchmarking where the IKE SA establishment rate is measured. Benchmarking is indistinguishable from a DoS attack and the defenses described in this document may defeat the benchmark by causing exchanges to fail or take a long time to complete. Parameters SHOULD be tunable to allow for benchmarking (if only by turning DDoS protection off).

Since all countermeasures may cause delays and additional work for the Initiators, they SHOULD NOT be deployed unless an attack is likely to be in progress. To minimize the burden imposed on Initiators, the Responder should monitor incoming IKE requests, for two scenarios:

1. A general DDoS attack. Such an attack is indicated by a high number of concurrent half-open SAs, a high rate of failed IKE_AUTH exchanges, or a combination of both. For example, consider a Responder that has 10,000 distinct peers of which at peak 7,500 concurrently have VPN tunnels. At the start of peak time, 600 peers might establish tunnels within any given minute, and tunnel establishment (both IKE_SA_INIT and IKE_AUTH) takes anywhere from 0.5 to 2 seconds. For this Responder, we expect there to be less than 20 concurrent half-open SAs, so having 100 concurrent half-open SAs can be interpreted as an indication of an attack. Similarly, IKE_AUTH request decryption failures should never happen. Supposing that the tunnels are established using EAP (see Section 2.16 of [RFC7296]), users may be expected to enter a wrong password about 20% of the time. So we'd expect 125 wrong password failures a minute. If we get IKE_AUTH decryption failures from multiple sources more than once per second, or EAP failures more than 300 times per minute, this can also be an indication of a DDoS attack.

2. An attack from a particular IP address or prefix. Such an attack is indicated by an inordinate amount of half-open SAs from a specific IP address or prefix, or an inordinate amount of IKE_AUTH failures. A DDoS attack may be viewed as multiple such attacks. If these are mitigated successfully, there will not be a need to enact countermeasures on all Initiators. For example, measures might be 5 concurrent half-open SAs, 1 decrypt failure, or 10 EAP failures within a minute.

Note that using counter-measures against an attack from a particular IP address may be enough to avoid the overload on the half-open SA database. In this case the number of failed IKE_AUTH exchanges will never exceed the threshold of attack detection.

When there is no general DDoS attack, it is suggested that no cookie or puzzles be used. At this point the only defensive measure is to monitor the number of half-open SAs, and set a soft limit per peer IP or prefix. The soft limit can be set to 3-5. If the puzzles are used, the puzzle difficulty SHOULD be set to such a level (number of zero-bits) that all legitimate clients can handle it without degraded user experience.

As soon as any kind of attack is detected, either a lot of initiations from multiple sources or a lot of initiations from a few sources, it is best to begin by requiring stateless cookies from all

Initiators. This will mitigate attacks based on IP address spoofing, and help avoid the need to impose a greater burden in the form of puzzles on the general population of Initiators. This makes the per-node or per-prefix soft limit more effective.

When cookies are activated for all requests and the attacker is still managing to consume too many resources, the Responder MAY start to use puzzles for these requests or increase the difficulty of puzzles imposed on IKE_SA_INIT requests coming from suspicious nodes/prefixes. This should still be doable by all legitimate peers, but the use of puzzles at a higher difficulty may degrade the user experience, for example by taking up to 10 seconds to solve the puzzle.

If the load on the Responder is still too great, and there are many nodes causing multiple half-open SAs or IKE_AUTH failures, the Responder MAY impose hard limits on those nodes.

If it turns out that the attack is very widespread and the hard caps are not solving the issue, a puzzle MAY be imposed on all Initiators. Note that this is the last step, and the Responder should avoid this if possible.

7. Using Puzzles in the Protocol

This section describes how the puzzle mechanism is used in IKEv2. It is organized as follows. The Section 7.1 describes using puzzles in the IKE_SA_INIT exchange and the Section 7.2 describes using puzzles in the IKE_AUTH exchange. Both sections are divided into subsections describing how puzzles should be presented, solved and processed by the Initiator and the Responder.

7.1. Puzzles in IKE_SA_INIT Exchange

IKE Initiator indicates the desire to create a new IKE SA by sending an IKE_SA_INIT request message. The message may optionally contain a COOKIE notification if this is a repeated request performed after the Responder’s demand to return a cookie.

```
HDR, [N(COOKIE),] SA, KE, Ni, [V+][N+]   -->
```

According to the plan, described in Section 6, the IKE Responder monitors incoming requests to detect whether it is under attack. If the Responder learns that a (D)DoS attack is likely to be in progress, then its actions depend on the volume of the attack. If the volume is moderate, then the Responder requests the Initiator to return a cookie. If the volume is high to such an extent that puzzles need to be used for defense, then the Responder requests the Initiator to solve a puzzle.

The Responder MAY choose to process some fraction of IKE_SA_INIT requests without presenting a puzzle while being under attack to allow legacy clients, that don't support puzzles, to have a chance to be served. The decision whether to process any particular request must be probabilistic, with the probability depending on the Responder's load (i.e. on the volume of attack). The requests that don't contain the COOKIE notification MUST NOT participate in this lottery. In other words, the Responder must first perform a return routability check before allowing any legacy client to be served if it is under attack. See Section 7.1.4 for details.

7.1.1. Presenting a Puzzle

If the Responder makes a decision to use puzzles, then it includes two notifications in its response message - the COOKIE notification and the PUZZLE notification. Note that the PUZZLE notification MUST always be accompanied with the COOKIE notification, since the content of the COOKIE notification is used as an input data when solving puzzle. The format of the PUZZLE notification is described in Section 8.1.

```
                          <--   HDR, N(COOKIE), N(PUZZLE), [V+][N+]
```

The presence of these notifications in an IKE_SA_INIT response message indicates to the Initiator that it should solve the puzzle to have a better chance to be served.

7.1.1.1. Selecting the Puzzle Difficulty Level

The PUZZLE notification contains the difficulty level of the puzzle - the minimum number of trailing zero bits that the result of PRF must contain. In diverse environments it is nearly impossible for the Responder to set any specific difficulty level that will result in roughly the same amount of work for all Initiators, because computation power of different Initiators may vary by an order of magnitude, or even more. The Responder may set the difficulty level to 0, meaning that the Initiator is requested to spend as much power to solve a puzzle as it can afford. In this case no specific value of ZBC is required from the Initiator, however the larger the ZBC that Initiator is able to get, the better the chance is that it will be served by the Responder. In diverse environments it is RECOMMENDED that the Initiator set the difficulty level to 0, unless the attack volume is very high.

If the Responder sets a non-zero difficulty level, then the level SHOULD be determined by analyzing the volume of the attack. The Responder MAY set different difficulty levels to different requests depending on the IP address the request has come from.

7.1.1.2. Selecting the Puzzle Algorithm

The PUZZLE notification also contains an identifier of the algorithm, that is used by Initiator to compute puzzle.

Cryptographic algorithm agility is considered an important feature for modern protocols [RFC7696]. Algorithm agility ensures that a protocol doesn’t rely on a single built-in set of cryptographic algorithms, but has a means to replace one set with another and negotiate new algorithms with the peer. IKEv2 fully supports cryptographic algorithm agility for its core operations.

To support crypto agility in case of puzzles, the algorithm that is used to compute a puzzle needs to be negotiated during the IKE_SA_INIT exchange. The negotiation is performed as follows. The initial request message from the Initiator contains an SA payload containing a list of transforms of different types. Thereby the Initiator asserts that it supports all transforms from this list and can use any of them in the IKE SA being established. The Responder parses the received SA payload and finds a mutually supported of type PRF. The Responder selects the preferred PRF from the list of mutually supported ones and includes it into the PUZZLE notification. There is no requirement that the PRF selected for puzzles be the same as the PRF that is negotiated later for use in core IKE SA crypto operations. If there are no mutually supported PRFs, then IKE SA negotiation will fail anyway and there is no reason to return a puzzle. In this case the Responder returns a NO_PROPOSAL_CHOSEN notification. Note that PRF is a mandatory transform type for IKE SA (see Sections 3.3.2 and 3.3.3 of [RFC7296]) and at least one transform of this type is always present in the SA payload in an IKE_SA_INIT request message.

7.1.1.3. Generating a Cookie

If the Responder supports puzzles then a cookie should be computed in such a manner that the Responder is able to learn some important information from the sole cookie, when it is later returned back by Initiator. In particular - the Responder SHOULD be able to learn the following information:

- Whether the puzzle was given to the Initiator or only the cookie was requested.

- The difficulty level of the puzzle given to the Initiator.

- The number of consecutive puzzles given to the Initiator.

- The amount of time the Initiator spent to solve the puzzles. This can be calculated if the cookie is timestamped.

This information helps the Responder to make a decision whether to serve this request or demand more work from the Initiator.

One possible approach to get this information is to encode it in the cookie. The format of such encoding is an implementation detail of Responder, as the cookie would remain an opaque block of data to the Initiator. If this information is encoded in the cookie, then the Responder MUST make it integrity protected, so that any intended or accidental alteration of this information in the returned cookie is detectable. So, the cookie would be generated as:

```
Cookie = <VersionIDofSecret> | <AdditionalInfo> |
                  Hash(Ni | IPi | SPIi | <AdditionalInfo> | <secret>)
```

Note, that according to the Section 2.6 of [RFC7296], the size of the cookie cannot exceed 64 bytes.

Alternatively, the Responder may generate a cookie as suggested in Section 2.6 of [RFC7296], but associate the additional information, using local storage identified with the particular version of the secret. In this case the Responder should have different secrets for every combination of difficulty level and number of consecutive puzzles, and should change the secrets periodically, keeping a few previous versions, to be able to calculate how long ago a cookie was generated.

The Responder may also combine these approaches. This document doesn’t mandate how the Responder learns this information from a cookie.

When selecting cookie generation algorithm implementations MUST ensure that an attacker gains no or insignificant benefit from re-using puzzle solutions in several requests. See Section 10 for details.

7.1.2. Solving a Puzzle and Returning the Solution

If the Initiator receives a puzzle but it doesn’t support puzzles, then it will ignore the PUZZLE notification as an unrecognized status notification (in accordance to Section 3.10.1 of [RFC7296]). The Initiator MAY ignore the PUZZLE notification if it is not willing to spend resources to solve the puzzle of the requested difficulty, even if it supports puzzles. In both cases the Initiator acts as described in Section 2.6 of [RFC7296] - it restarts the request and includes the received COOKIE notification into it. The Responder should be able to distinguish the situation when it just requested a cookie from the situation where the puzzle was given to the Initiator, but the Initiator for some reason ignored it.

If the received message contains a PUZZLE notification and doesn't contain a COOKIE notification, then this message is malformed because it requests to solve the puzzle, but doesn't provide enough information to allow the puzzle to be solved. In this case the Initiator MUST ignore the received message and continue to wait until either a valid PUZZLE notification is received or the retransmission timer fires. If it fails to receive a valid message after several retransmissions of IKE_SA_INIT requests, then it means that something is wrong and the IKE SA cannot be established.

If the Initiator supports puzzles and is ready to solve them, then it tries to solve the given puzzle. After the puzzle is solved the Initiator restarts the request and returns back to the Responder the puzzle solution in a new payload called a Puzzle Solution payload (denoted as PS, see Section 8.2) along with the received COOKIE notification.

```
HDR, N(COOKIE), [PS,] SA, KE, Ni, [V+][N+]   -->
```

7.1.3. Computing a Puzzle

General principles of constructing puzzles in IKEv2 are described in Section 4.4. They can be summarized as follows: given unpredictable string S and pseudo-random function PRF find N different keys Ki (where i=[1..N]) for that PRF so that the result of PRF(Ki,S) has at least the specified number of trailing zero bits. This specification requires that the puzzle solution contains 4 different keys (i.e., N=4).

In the IKE_SA_INIT exchange it is the cookie that plays the role of unpredictable string S. In other words, in the IKE_SA_INIT the task for the IKE Initiator is to find the four different, equal-sized keys Ki for the agreed upon PRF such that each result of PRF(Ki,cookie) where i = [1..4] has a sufficient number of trailing zero bits. Only the content of the COOKIE notification is used in puzzle calculation, i.e., the header of the Notify payload is not included.

Note, that puzzles in the IKE_AUTH exchange are computed differently than in the IKE_SA_INIT_EXCHANGE. See Section 7.2.3 for details.

7.1.4. Analyzing Repeated Request

The received request must at least contain a COOKIE notification. Otherwise it is an initial request and in this case it MUST be processed according to Section 7.1. First, the cookie MUST be checked for validity. If the cookie is invalid, then the request is treated as initial and is processed according to Section 7.1. It is RECOMMENDED that a new cookie is requested in this case.

If the cookie is valid, then some important information is learned from it, or from local state based on identifier of the cookie's secret (see Section 7.1.1.3 for details). This information helps the Responder to sort out incoming requests, giving more priority to those which were created by spending more of the Initiator's resources.

First, the Responder determines if it requested only a cookie, or presented a puzzle to the Initiator. If no puzzle was given, this means that at the time the Responder requested a cookie it didn't detect the (D)DoS attack or the attack volume was low. In this case the received request message must not contain the PS payload, and this payload MUST be ignored if the message contains a PS payload for any reason. Since no puzzle was given, the Responder marks the request with the lowest priority since the Initiator spent little resources creating it.

If the Responder learns from the cookie that the puzzle was given to the Initiator, then it looks for the PS payload to determine whether its request to solve the puzzle was honored or not. If the incoming message doesn't contain a PS payload, this means that the Initiator either doesn't support puzzles or doesn't want to deal with them. In either case the request is marked with the lowest priority since the Initiator spent little resources creating it.

If a PS payload is found in the message, then the Responder MUST verify the puzzle solution that it contains. The solution is interpreted as four different keys. The result of using each of them in the PRF (as described in Section 7.1.3) must contain at least the requested number of trailing zero bits. The Responder MUST check all of the four returned keys.

If any checked result contains fewer bits than were requested, this means that the Initiator spent less resources than expected by the Responder. This request is marked with the lowest priority.

If the Initiator provided the solution to the puzzle satisfying the requested difficulty level, or if the Responder didn't indicate any particular difficulty level (by setting ZBC to zero) and the

Initiator was free to select any difficulty level it can afford, then the priority of the request is calculated based on the following considerations:

- The Responder MUST take the smallest number of trailing zero bits among the checked results and count it as the number of zero bits the Initiator solved for.

- The higher number of zero bits the Initiator provides, the higher priority its request should receive.

- The more consecutive puzzles the Initiator solved, the higher priority it should receive.

- The more time the Initiator spent solving the puzzles, the higher priority it should receive.

After the priority of the request is determined the final decision whether to serve it or not is made.

7.1.5. Deciding if to Serve the Request

The Responder decides what to do with the request based on the request's priority and the Responder's current load. There are three possible actions:

- Accept request.

- Reject request.

- Demand more work from the Initiator by giving it a new puzzle.

The Responder SHOULD accept an incoming request if its priority is high - this means that the Initiator spent quite a lot of resources. The Responder MAY also accept some low-priority requests where the Initiators don't support puzzles. The percentage of accepted legacy requests depends on the Responder's current load.

If the Initiator solved the puzzle, but didn't spend much resources for it (the selected puzzle difficulty level appeared to be low and the Initiator solved it quickly), then the Responder SHOULD give it another puzzle. The more puzzles the Initiator solves the higher its chances are to be served.

The details of how the Responder makes a decision for any particular request are implementation dependent. The Responder can collect all of the incoming requests for some short period of time, sort them out based on their priority, calculate the number of available memory slots for half-open IKE SAs and then serve that number of requests from the head of the sorted list. The remainder of requests can be either discarded or responded to with new puzzle requests.

Alternatively, the Responder may decide whether to accept every incoming request with some kind of lottery, taking into account its priority and the available resources.

7.2. Puzzles in an IKE_AUTH Exchange

Once the IKE_SA_INIT exchange is completed, the Responder has created a state and is waiting for the first message of the IKE_AUTH exchange from the Initiator. At this point the Initiator has already passed the return routability check and has proved that it has performed some work to complete IKE_SA_INIT exchange. However, the Initiator is not yet authenticated and this allows a malicious Initiator to perform an attack, described in Section 3. Unlike a DoS attack in the IKE_SA_INIT exchange, which is targeted on the Responder's memory resources, the goal of this attack is to exhaust a Responder's CPU power. The attack is performed by sending the first IKE_AUTH message containing arbitrary data. This costs nothing to the Initiator, but the Responder has to perform relatively costly operations when computing the D-H shared secret and deriving SK_* keys to be able to verify authenticity of the message. If the Responder doesn't keep the computed keys after an unsuccessful verification of the IKE_AUTH message, then the attack can be repeated several times on the same IKE SA.

The Responder can use puzzles to make this attack more costly for the Initiator. The idea is that the Responder includes a puzzle in the IKE_SA_INIT response message and the Initiator includes a puzzle solution in the first IKE_AUTH request message outside the Encrypted payload, so that the Responder is able to verify puzzle solution before computing the D-H shared secret.

The Responder constantly monitors the amount of the half-open IKE SA states that receive IKE_AUTH messages that cannot be decrypted due to integrity check failures. If the percentage of such states is high and it takes an essential fraction of Responder's computing power to calculate keys for them, then the Responder may assume that it is under attack and SHOULD use puzzles to make it harder for attackers.

7.2.1. Presenting Puzzle

The Responder requests the Initiator to solve a puzzle by including the PUZZLE notification in the IKE_SA_INIT response message. The Responder MUST NOT use puzzles in the IKE_AUTH exchange unless a puzzle has been previously presented and solved in the preceding IKE_SA_INIT exchange.

```
                          <--   HDR, SA, KE, Nr, N(PUZZLE), [V+][N+]
```

7.2.1.1. Selecting Puzzle Difficulty Level

The difficulty level of the puzzle in the IKE_AUTH exchange should be chosen so that the Initiator would spend more time to solve the puzzle than the Responder to compute the D-H shared secret and the keys needed to decrypt and verify the IKE_AUTH request message. On the other hand, the difficulty level should not be too high, otherwise legitimate clients will experience an additional delay while establishing the IKE SA.

Note, that since puzzles in the IKE_AUTH exchange are only allowed to be used if they were used in the preceding IKE_SA_INIT exchange, the Responder would be able to roughly estimate the computational power of the Initiator and select the difficulty level accordingly. Unlike puzzles in the IKE_SA_INIT, the requested difficulty level for IKE_AUTH puzzles MUST NOT be zero. In other words, the Responder must always set a specific difficulty level and must not let the Initiator to choose it on its own.

7.2.1.2. Selecting the Puzzle Algorithm

The algorithm for the puzzle is selected as described in Section 7.1.1.2. There is no requirement that the algorithm for the puzzle in the IKE_SA INIT exchange be the same as the algorithm for the puzzle in IKE_AUTH exchange; however, it is expected that in most cases they will be the same.

7.2.2. Solving Puzzle and Returning the Solution

If the IKE_SA_INIT regular response message (i.e. the message containing SA, KE, NONCE payloads) contains the PUZZLE notification and the Initiator supports puzzles, it MUST solve the puzzle. Note, that puzzle construction in the IKE_AUTH exchange differs from the puzzle construction in the IKE_SA_INIT exchange and is described in Section 7.2.3. Once the puzzle is solved the Initiator sends the IKE_AUTH request message containing the Puzzle Solution payload.

```
HDR, PS, SK {IDi, [CERT,] [CERTREQ,]
            [IDr,] AUTH, SA, TSi, TSr}   -->
```

The Puzzle Solution (PS) payload MUST be placed outside the Encrypted payload, so that the Responder is able to verify the puzzle before calculating the D-H shared secret and the SK_* keys.

If IKE Fragmentation [RFC7383] is used in IKE_AUTH exchange, then the PS payload MUST be present only in the first IKE Fragment message, in accordance with the Section 2.5.3 of [RFC7383]. Note, that calculation of the puzzle in the IKE_AUTH exchange doesn't depend on the content of the IKE_AUTH message (see Section 7.2.3). Thus the Initiator has to solve the puzzle only once and the solution is valid for both unfragmented and fragmented IKE messages.

7.2.3. Computing the Puzzle

A puzzle in the IKE_AUTH exchange is computed differently than in the IKE_SA_INIT exchange (see Section 7.1.3). The general principle is the same; the difference is in the construction of the string S. Unlike the IKE_SA_INIT exchange, where S is the cookie, in the IKE_AUTH exchange S is a concatenation of Nr and SPIr. In other words, the task for IKE Initiator is to find the four different keys Ki for the agreed upon PRF such that each result of PRF(Ki,Nr | SPIr) where i=[1..4] has a sufficient number of trailing zero bits. Nr is a nonce used by the Responder in the IKE_SA_INIT exchange, stripped of any headers. SPIr is the IKE Responder's SPI from the IKE header of the SA being established.

7.2.4. Receiving the Puzzle Solution

If the Responder requested the Initiator to solve a puzzle in the IKE_AUTH exchange, then it MUST silently discard all the IKE_AUTH request messages without the Puzzle Solution payload.

Once the message containing a solution to the puzzle is received, the Responder MUST verify the solution before performing computationlly intensive operations i.e., computing the D-H shared secret and the SK_* keys. The Responder MUST verify all four of the returned keys.

The Responder MUST silently discard the received message if any checked verification result is not correct (contains insufficient number of trailing zero bits). If the Responder successfully verifies the puzzle and calculates the SK_* key, but the message authenticity check fails, then it SHOULD save the calculated keys in the IKE SA state while waiting for the retransmissions from the Initiator. In this case the Responder may skip verification of the puzzle solution and ignore the Puzzle Solution payload in the retransmitted messages.

If the Initiator uses IKE Fragmentation, then it sends all fragments of a message simultaneously. Due to packets loss and/or reordering it is possible that the Responder receives subsequent fragments before receiving the first one, that contains the PS payload. In this case the Responder MAY choose to keep the received fragments until the first fragment containing the solution to the puzzle is received. In this case the Responder SHOULD NOT try to verify authenticity of the kept fragments until the first fragment with the PS payload is received and the solution to the puzzle is verified. After successful verification of the puzzle, the Responder can then calculate the SK_* key and verify authenticity of the collected fragments.

8. Payload Formats

8.1. PUZZLE Notification

The PUZZLE notification is used by the IKE Responder to inform the Initiator about the need to solve the puzzle. It contains the difficulty level of the puzzle and the PRF the Initiator should use.

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

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

o SPI Size (1 octet) - MUST be 0, meaning no Security Parameter Index (SPI) is present.

o Notify Message Type (2 octets) -- MUST be <TBA by IANA>, the value assigned for the PUZZLE notification.

o PRF (2 octets) -- Transform ID of the PRF algorithm that MUST be used to solve the puzzle. Readers should refer to the section "Transform Type 2 - Pseudo-Random Function Transform IDs" in [IKEV2-IANA] for the list of possible values.

o Difficulty (1 octet) -- Difficulty Level of the puzzle. Specifies the minimum number of trailing zero bits (ZBC), that each of the results of PRF must contain. Value 0 means that the Responder doesn’t request any specific difficulty level and the Initiator is free to select an appropriate difficulty level on its own (see Section 7.1.1.1 for details).

This notification contains no data.

8.2. Puzzle Solution Payload

The solution to the puzzle is returned back to the Responder in a dedicated payload, called the Puzzle Solution payload and denoted as PS in this document.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                     Puzzle Solution Data                      ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Puzzle Solution Data (variable length) -- Contains the solution to the puzzle - four different keys for the selected PRF. This field MUST NOT be empty. All of the keys MUST have the same size, therefore the size of this field is always a mutiple of 4 bytes. If the selected PRF accepts only fixed-size keys, then the size of each key MUST be of that fixed size. If the agreed upon PRF accepts keys of any size, then then the size of each key MUST be between 1 octet and the preferred key length of the PRF (inclusive). It is expected that in most cases the keys will be 4 (or even less) octets in length, however it depends on puzzle difficulty and on the Initiator’s strategy to find solutions, and thus the size is not mandated by this specification. The Responder determines the size of each key by dividing the size of the Puzzle Solution Data by 4 (the number of keys). Note that the size of Puzzle Solution Data is the size of Payload (as indicated in Payload Length field) minus 4 - the size of Payload Header.

The payload type for the Puzzle Solution payload is <TBA by IANA>.

9. Operational Considerations

The puzzle difficulty level should be set by balancing the requirement to minimize the latency for legitimate Initiators with making things difficult for attackers. A good rule of thumb is for taking about 1 second to solve the puzzle. A typical Initiator or botnet member at the time this document is written can perform slightly less than a million hashes per second per core, so setting the number of zero bits to 20 is a good compromise. It should be noted that mobile Initiators, especially phones are considerably weaker than that. Implementations should allow administrators to set the difficulty level, and/or be able to set the difficulty level dynamically in response to load.

Initiators SHOULD set a maximum difficulty level beyond which they won't try to solve the puzzle and log or display a failure message to the administrator or user.

Until the widespread adoption of puzzles happens, most Initiators will ignore them, as will all attackers. For puzzles to become a really powerfull defense measure against DDoS attacks they must be supported by the majority of legitimate clients.

10. Security Considerations

Care must be taken when selecting parameters for the puzzles, in particular the puzzle difficulty. If the puzzles are too easy for the majority of attacker, then the puzzle mechanism wouldn't be able to prevent (D)DoS attacks and would only impose an additional burden on legitimate Initiators. On the other hand, if the puzzles are too hard for the majority of Initiators, then many legitimate users would experience unacceptable delays in IKE SA setup (and unacceptable power consumption on mobile devices), that might cause them to cancel the connection attempt. In this case the resources of the Responder are preserved, however the DoS attack can be considered successful. Thus a sensible balance should be kept by the Responder while choosing the puzzle difficulty - to defend itself and to not over-defend itself. It is RECOMMENDED that the puzzle difficulty be chosen so, that the Responder's load remains close to the maximum it can tolerate. It is also RECOMMENDED to dynamically adjust the puzzle difficulty in accordance to the current Responder's load.

If the cookie is generated as suggested in Section 2.6 of [RFC7296], then an attacker can use the same SPIi and the same Ni for several requests from the same IPi. This will result in generating the same cookies for these requests until the Responder changes the value of its cookie generation secret. Since the cookies are used as an input data for puzzles in the IKE_SA_INIT exchange, generating same cookies allows the attacker to re-use puzzle solution, thus bypassing proof of work requirement. Note, that the attacker can get only limited benefit from this situation - once the half-open SA is created by the Responder all the subsequent initial requests with the same IPi and SPIi will be treated as retransmissions and discarded by the Responder. However, once this half-open SA is expired and deleted, the attacker can create a new one for free if the Responder haven't changed its cookie generation secret yet.

The Responder can use various countermeasures to completely eliminate or mitigate this scenatio. First, the Responder can change its cookie generation secret frequently especially if under attack, as recommended in the Section 2.6 of [RFC7296]. For example, if the Responder keeps two values of the secret (current and previous) and the secret lifetime is no more than a half of the current half-open SA retention time (see Section 4.1), then the attacker cannot get benefit from re-using puzzle solution. However, short cookie generation secret lifetime could have negative consequence on weak legitimate Initiators, since it could take too long for them to solve puzzles and their solutons would be discarded if the cookie generation secret has been already changed few times.

Another approach for the Responder is to modify cookie generation algorithm in such a way, that the generated cookies are always different or are repeated only within short time period. If the Responder includes timestamp in the <AdditionalInfo> as suggested in Section 7.1.1.3, then the cookies will repeat only within short time interval equal to timestamp resolution. Another approach for the Responder is to maintain a global counter that is incremented every time a cookie is generated and include this counter in the <AdditionalInfo>. This will make every cookies unique.

Implementations MUST use one of the above (or some other) countermeasures to completely eliminate or make insignificant the possible benefit an attacker can get from re-using puzzle solutions. Note, this issue doesn't exist in IKE_AUTH puzzles (Section 7.2) since the puzzles in IKE_AUTH are always unique if the Responder generates SPIr and Nr randomly in accordance with [RFC7296].

Solving puzzles requires a lot of CPU power that increases power consumption. This additional power consumption can negatively affect battery-powered Initiators, e.g. mobile phones or some IoT devices. If puzzles are too hard, then the required additional power consumption may appear to be unacceptable for some Initiators. The Responder SHOULD take this possibility into consideration while choosing the puzzle difficulty, and while selecting which percentage of Initiators are allowed to reject solving puzzles. See Section 7.1.4 for details.

If the Initiator uses NULL Authentication [RFC7619] then its identity is never verified. This condition may be used by attackers to perform a DoS attack after the IKE SA is established. Responders that allow unauthenticated Initiators to connect must be prepared to deal with various kinds of DoS attacks even after the IKE SA is created. See Section 5 for details.

To prevent amplification attacks implementations must strictly follow the retransmission rules described in Section 2.1 of [RFC7296].

11. IANA Considerations

This document defines a new payload in the "IKEv2 Payload Types" registry:

```
<TBA>       Puzzle Solution                  PS
```

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

```
<TBA>       PUZZLE
```

12. Acknowledgements

The authors thank Tero Kivinen, Yaron Sheffer, and Scott Fluhrer for their contributions to the design of the protocol. In particular, Tero Kivinen suggested the kind of puzzle where the task is to find a solution with a requested number of zero trailing bits. Yaron Sheffer and Scott Fluhrer suggested a way to make puzzle difficulty less erratic by solving several weaker puzles. The authors also thank David Waltermire and Paul Wouters for their careful reviews of the document, Graham Bartlett for pointing out to the possibility of the "Hash & URL" related attack, Stephen Farrell for catching the repeated cookie issue, and all others who commented the document.

13. References

13.1. 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, <http://www.rfc-editor.org/info/rfc2119>.

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

[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>.

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

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

13.2. Informative References

[bitcoins]
Nakamoto, S., "Bitcoin: A Peer-to-Peer Electronic Cash System", October 2008, <https://bitcoin.org/bitcoin.pdf>.

[RFC7619] Smyslov, V. and P. Wouters, "The NULL Authentication Method in the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 7619, DOI 10.17487/RFC7619, August 2015, <http://www.rfc-editor.org/info/rfc7619>.

[RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <http://www.rfc-editor.org/info/rfc7696>.

Authors' Addresses

Yoav Nir
Check Point Software Technologies Ltd.
5 Hasolelim st.
Tel Aviv 6789735
Israel

EMail: ynir.ietf@gmail.com

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

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

Network Working Group
Internet-Draft
Obsoletes: 4307 (if approved)
Updates: 7296 (if approved)
Intended status: Standards Track
Expires: September 30, 2017

Y. Nir
Check Point
T. Kivinen
INSIDE Secure
P. Wouters
Red Hat
D. Migault
Ericsson

March 29, 2017

Algorithm Implementation Requirements and Usage Guidance for IKEv2
draft-ietf-ipsecme-rfc4307bis-18

Abstract

The IPsec series of protocols makes use of various cryptographic algorithms in order to provide security services. The Internet Key Exchange (IKE) protocol is used to negotiate the IPsec Security Association (IPsec SA) parameters, such as which algorithms should be used. To ensure interoperability between different implementations, it is necessary to specify a set of algorithm implementation requirements and usage guidance to ensure that there is at least one algorithm that all implementations support. This document updates RFC 7296 and obsoletes RFC 4307 in defining the current algorithm implementation requirements and usage guidance for IKEv2, and does minor cleaning up of the IKEv2 IANA registry. This document does not update the algorithms used for packet encryption using IPsec Encapsulated Security Payload (ESP).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Internet Key Exchange (IKE) protocol [RFC7296] is used to negotiate the parameters of the IPsec SA, such as the encryption and authentication algorithms and the keys for the protected communications between the two endpoints. The IKE protocol itself is also protected by cryptographic algorithms which are negotiated between the two endpoints using IKE. Different implementations of IKE may negotiate different algorithms based on their individual local policy. To ensure interoperability, a set of "mandatory-to-implement" IKE cryptographic algorithms is defined.

This document describes the parameters of the IKE protocol and updates the IKEv2 specification. It changes the mandatory to implement authentication algorithms of Section 4 of [RFC7296] by saying RSA key lengths of less than 2048 SHOULD NOT be used. It does not describe the cryptographic parameters of the AH or ESP protocols.

1.1. Conventions Used in This Document

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

When used in the tables in this document, these terms indicate that the listed algorithm MUST, MUST NOT, SHOULD, SHOULD NOT or MAY be implemented as part of an IKEv2 implementation. Additional terms used in this document are:

SHOULD+ This term means the same as SHOULD. However, it is likely that an algorithm marked as SHOULD+ will be promoted at some future time to be a MUST.

SHOULD- This term means the same as SHOULD. However, an algorithm marked as SHOULD- may be deprecated to a MAY in a future version of this document.

MUST- This term means the same as MUST. However, it is expected at some point that this algorithm will no longer be a MUST in a future document. Although its status will be determined at a later time, it is reasonable to expect that if a future revision of a document alters the status of a MUST- algorithm, it will remain at least a SHOULD or a SHOULD- level.

IoT stands for Internet of Things.

1.2. Updating Algorithm Implementation Requirements and Usage Guidance

The field of cryptography evolves continuously. New stronger algorithms appear and existing algorithms are found to be less secure then originally thought. Therefore, algorithm implementation requirements and usage guidance need to be updated from time to time to reflect the new realityI The choices for algorithms must be conservative to minimize the risk of algorithm compromise. Algorithms need to be suitable for a wide variety of CPU architectures and device deployments ranging from high end bulk encryption devices to small low-power IoT devices.

The algorithm implementation requirements and usage guidance may need to change over time to adapt to the changing world. For this reason, the selection of mandatory-to-implement algorithms was removed from the main IKEv2 specification and placed in this separate document.

1.3. Updating Algorithm Requirement Levels

The mandatory-to-implement algorithm of tomorrow should already be available in most implementations of IKE by the time it is made mandatory. This document attempts to identify and introduce those algorithms for future mandatory-to-implement status. There is no guarantee that the algorithms in use today may become mandatory in the future. Published algorithms are continuously subjected to cryptographic attack and may become too weak or could become completely broken before this document is updated.

This document only provides recommendations for the mandatory-to-implement algorithms or algorithms too weak that are recommended not to be implemented. As a result, any algorithm listed at the IKEv2 IANA registry not mentioned in this document MAY be implemented. For clarification and consistency with [RFC4307] an algorithm will be denoted here as MAY only when it has been downgraded.

Although this document updates the algorithms to keep the IKEv2 communication secure over time, it also aims at providing recommendations so that IKEv2 implementations remain interoperable. IKEv2 interoperability is addressed by an incremental introduction or deprecation of algorithms. In addition, this document also considers the new use cases for IKEv2 deployment, such as Internet of Things (IoT).

It is expected that deprecation of an algorithm is performed gradually. This provides time for various implementations to update their implemented algorithms while remaining interoperable. Unless there are strong security reasons, an algorithm is expected to be downgraded from MUST to MUST- or SHOULD, instead of MUST NOT. Similarly, an algorithm that has not been mentioned as mandatory-to-implement is expected to be introduced with a SHOULD instead of a MUST.

The current trend toward Internet of Things and its adoption of IKEv2 requires this specific use case to be taken into account as well. IoT devices are resource constrained devices and their choice of algorithms are motivated by minimizing the footprint of the code, the computation effort and the size of the messages to send. This document indicates "(IoT)" when a specified algorithm is specifically listed for IoT devices. Requirement levels that are marked as "IoT" apply to IoT devices and to server-side implementations that might presumably need to interoperate with them, including any general-purpose VPN gateways.

1.4. Document Audience

The recommendations of this document mostly target IKEv2 implementers who need to create implementations that meet both high security expectations as well as high interoperability between various vendors and with different versions. Interoperability requires a smooth move to more secure cipher suites. This may differ from a user point of view that may deploy and configure IKEv2 with only the safest cipher suite.

This document does not give any recommendations for the use of algorithms, it only gives implementation recommendations regarding implementations. The use of algorithms by users is dictated by the security policy requirements for that specific user, and are outside the scope of this document.

IKEv1 is out of scope of this document. IKEv1 is deprecated and the recommendations of this document must not be considered for IKEv1, as most IKEv1 implementations have been "frozen" and will not be able to update the list of mandatory-to-implement algorithms.

2. Algorithm Selection

2.1. Type 1 - IKEv2 Encryption Algorithm Transforms

The algorithms in the below table are negotiated in the SA payload and used for the Encrypted Payload. References to the specification defining these algorithms and the ones in the following subsections are in the IANA registry [IKEV2-IANA]. Some of these algorithms are Authenticated Encryption with Associated Data (AEAD - [RFC5282]). Algorithms that are not AEAD MUST be used in conjunction with one of the integrity algorithms in Section 2.3.

Name	Status	AEAD?	Comment
ENCR_AES_CBC	MUST	No	(1)
ENCR_CHACHA20_POLY1305	SHOULD	Yes	
ENCR_AES_GCM_16	SHOULD	Yes	(1)
ENCR_AES_CCM_8	SHOULD	Yes	(IoT)
ENCR_3DES	MAY	No	
ENCR_DES	MUST NOT	No	

(1) - This requirement level is for 128-bit and 256-bit keys. 192-bit keys remain at MAY level. (IoT) - This requirement is for interoperability with IoT. Only 128-bit keys are at SHOULD level. 192-bit and 256-bit remain at the MAY level.

ENCR_AES_CBC is raised from SHOULD+ for 128-bit keys and MAY for 256-bit keys in [RFC4307] to MUST. 192-bit keys remain at the MAY level. ENCR_AES_CBC is the only shared mandatory-to-implement algorithm with RFC4307 and as a result it is necessary for interoperability with IKEv2 implementation compatible with RFC4307.

ENCR_CHACHA20_POLY1305 was not ready to be considered at the time of RFC4307. It has been recommended by the Crypto Forum Research Group (CFRG) of the IRTF as an alternative to AES-CBC and AES-GCM. It is also being standardized for IPsec for the same reasons. At the time of writing, there were not enough IKEv2 implementations supporting ENCR_CHACHA20_POLY1305 to be able to introduce it at the SHOULD+ level.

ENCR_AES_GCM_16 was not considered in RFC4307. At the time RFC4307 was written, AES-GCM was not defined in an IETF document. AES-GCM was defined for ESP in [RFC4106] and later for IKEv2 in [RFC5282]. The main motivation for adopting AES-GCM for ESP is encryption performance compared to AES-CBC. This resulted in AES-GCM being widely implemented for ESP. As the computation load of IKEv2 is relatively small compared to ESP, many IKEv2 implementations have not implemented AES-GCM. For this reason, AES-GCM is not promoted to a greater status than SHOULD. The reason for promotion from MAY to SHOULD is to promote the slightly more secure AEAD method over the traditional encrypt+auth method. Its status is expected to be raised once widely implemented. As the advantage of the shorter (and weaker) ICVs is minimal, the 8 and 12 octet ICV’s remain at the MAY level.

ENCR_AES_CCM_8 was not considered in RFC4307. This document considers it as SHOULD be implemented in order to be able to interact with Internet of Things devices. As this case is not a general use case for non-IoT VPNs, its status is expected to remain as SHOULD. The 8 octet size of the ICV is expected to be sufficient for most use cases of IKEv2, as far less packets are exchanged in those cases, and IoT devices want to make packets as small as possible. The SHOULD level is for 128-bit keys, 256-bit keys remains at MAY level.

ENCR_3DES has been downgraded from RFC4307 MUST- to MAY. All IKEv2 implementations already implement ENCR_AES_CBC, so there is no need to keep support for the much slower ENCR_3DES. In addition, ENCR_CHACHA20_POLY1305 provides a more modern alternative to AES.

ENCR_DES can be brute-forced using off-the-shelf hardware. It provides no meaningful security whatsoever and therefore MUST NOT be implemented.

2.2. Type 2 - IKEv2 Pseudo-random Function Transforms

Transform Type 2 algorithms are pseudo-random functions used to generate pseudo-random values when needed.

Name	Status	Comment
PRF_HMAC_SHA2_256	MUST	
PRF_HMAC_SHA2_512	SHOULD+	
PRF_HMAC_SHA1	MUST-	
PRF_AES128_XCBC	SHOULD	(IoT)
PRF_HMAC_MD5	MUST NOT	

(IoT) - This requirement is for interoperability with IoT

As no SHA2 based transforms were referenced in RFC4307, PRF_HMAC_SHA2_256 was not mentioned in RFC4307. PRF_HMAC_SHA2_256 MUST be implemented in order to replace SHA1 and PRF_HMAC_SHA1.

PRF_HMAC_SHA2_512 SHOULD be implemented as a future replacement for PRF_HMAC_SHA2_256 or when stronger security is required. PRF_HMAC_SHA2_512 is preferred over PRF_HMAC_SHA2_384, as the additional overhead of PRF_HMAC_SHA2_512 is negligible.

PRF_HMAC_SHA1 has been downgraded from MUST in RFC4307 to MUST- as cryptographic attacks against SHA1 are increasing, resulting in an industry-wide trend to deprecate its usage

PRF_AES128_XCBC is only recommended in the scope of IoT, as Internet of Things deployments tend to prefer AES based pseudo-random functions in order to avoid implementing SHA2. For the non-IoT VPN deployment it has been downgraded from SHOULD in RFC4307 to MAY as it has not seen wide adoption.

PRF_HMAC_MD5 has been downgraded from MAY in RFC4307 to MUST NOT. Cryptographic attacks against MD5, such as collision attacks mentioned in [TRANSCRIPTION], are resulting in an industry-wide trend to deprecate and remove MD5 (and thus HMAC-MD5) from cryptographic libraries.

2.3. Type 3 - IKEv2 Integrity Algorithm Transforms

The algorithms in the below table are negotiated in the SA payload and used for the Encrypted Payload. References to the specification defining these algorithms are in the IANA registry. When an AEAD algorithm (see Section 2.1) is proposed, this algorithm transform type is not in use.

Name	Status	Comment
AUTH_HMAC_SHA2_256_128	MUST	
AUTH_HMAC_SHA2_512_256	SHOULD	
AUTH_HMAC_SHA1_96	MUST-	
AUTH_AES_XCBC_96	SHOULD	(IoT)
AUTH_HMAC_MD5_96	MUST NOT	
AUTH_DES_MAC	MUST NOT	
AUTH_KPDK_MD5	MUST NOT	

(IoT) - This requirement is for interoperability with IoT

AUTH_HMAC_SHA2_256_128 was not mentioned in RFC4307, as no SHA2 based transforms were mentioned. AUTH_HMAC_SHA2_256_128 MUST be implemented in order to replace AUTH_HMAC_SHA1_96.

AUTH_HMAC_SHA2_512_256 SHOULD be implemented as a future replacement of AUTH_HMAC_SHA2_256_128 or when stronger security is required. This value has been preferred over AUTH_HMAC_SHA2_384, as the additional overhead of AUTH_HMAC_SHA2_512 is negligible.

AUTH_HMAC_SHA1_96 has been downgraded from MUST in RFC4307 to MUST- as cryptographic attacks against SHA1 are increasing, resulting in an industry-wide trend to deprecate its usage

AUTH_AES_XCBC_96 is only recommended in the scope of IoT, as Internet of Things deployments tend to prefer AES based pseudo-random functions in order to avoid implementing SHA2. For the non-IoT VPN deployment, it has been downgraded from SHOULD in RFC4307 to MAY as it has not been widely adopted.

AUTH_DES_MAC, AUTH_HMAC_MD5_96, and AUTH_KPDK_MD5 were not mentioned in RFC4307 so their default statuses were MAY. They have been downgraded to MUST NOT. There is an industry-wide trend to deprecate DES and MD5. MD5 support is being removed from cryptographic libraries in general because its non-HMAC use is known to be subject to collision attacks, for example as mentioned in [TRANSCRIPTION].

2.4. Type 4 - IKEv2 Diffie-Hellman Group Transforms

There are several Modular Exponential (MODP) groups and several Elliptic Curve groups (ECC) that are defined for use in IKEv2. These groups are defined in both the [RFC7296] base document and in extensions documents and are identified by group number. Note that it is critical to enforce a secure Diffie-Hellman exchange as this exchange provides keys for the session. If an attacker can retrieve one of the private numbers (a or b) and the complementary public value (g**b or g**a), then the attacker can compute the secret and the keys used and decrypt the exchange and IPsec SA created inside the IKEv2 SA. Such an attack can be performed off-line on a previously recorded communication, years after the communication happened. This differs from attacks that need to be executed during the authentication which must be performed online and in near real-time.

Number	Description	Status
14	2048-bit MODP Group	MUST
19	256-bit random ECP group	SHOULD
5	1536-bit MODP Group	SHOULD NOT
2	1024-bit MODP Group	SHOULD NOT
1	768-bit MODP Group	MUST NOT
22	1024-bit MODP Group with 160-bit Prime Order Subgroup	MUST NOT
23	2048-bit MODP Group with 224-bit Prime Order Subgroup	SHOULD NOT
24	2048-bit MODP Group with 256-bit Prime Order Subgroup	SHOULD NOT

Group 14 or 2048-bit MODP Group is raised from SHOULD+ in RFC4307 to MUST as a replacement for 1024-bit MODP Group. Group 14 is widely implemented and considered secure.

Group 19 or 256-bit random ECP group was not specified in RFC4307, as this group was not defined at that time. Group 19 is widely implemented and considered secure and therefore has been promoted to the SHOULD level.

Group 5 or 1536-bit MODP Group has been downgraded from MAY in RFC4307 to SHOULD NOT. It was specified earlier, but is now considered to be vulnerable to being broken within the next few years by a nation state level attack, so its security margin is considered too narrow.

Group 2 or 1024-bit MODP Group has been downgraded from MUST- in RFC4307 to SHOULD NOT. It is known to be weak against sufficiently funded attackers using commercially available mass-computing resources, so its security margin is considered too narrow. It is expected in the near future to be downgraded to MUST NOT.

Group 1 or 768-bit MODP Group was not mentioned in RFC4307 and so its status was MAY. It can be broken within hours using cheap of-the-shelves hardware. It provides no security whatsoever. It has therefore been downgraded to MUST NOT.

Group 22, 23 and 24 are MODP Groups with Prime Order Subgroups that are not safe-primes. The seeds for these groups have not been publicly released, resulting in reduced trust in these groups. These groups were proposed as alternatives for group 2 and 14 but never saw wide deployment. It has been shown that Group 22 with 1024-bit MODP is too weak and academia have the resources to generate malicious values at this size. This has resulted in Group 22 to be demoted to MUST NOT. Group 23 and 24 have been demoted to SHOULD NOT and are expected to be further downgraded in the near future to MUST NOT. Since Group 23 and 24 have small subgroups, the checks specified in "Additional Diffie-Hellman Test for the IKEv2" [RFC6989] section 2.2 first bullet point MUST be done when these groups are used.

2.5. Summary of Changes from RFC 4307

The following table summarizes the changes from RFC 4307.

RFC EDITOR: PLEASE REMOVE THIS PARAGRAPH AND REPLACE XXXX IN THE TABLE BELOW WITH THE NUMBER OF THIS RFC

Algorithm	RFC 4307	RFC XXXX
ENCR_3DES	MUST-	MAY
ENCR_NULL	MUST NOT[errata]	MUST NOT
ENCR_AES_CBC	SHOULD+	MUST
ENCR_AES_CTR	SHOULD	(*)
PRF_HMAC_MD5	MAY	MUST NOT
PRF_HMAC_SHA1	MUST	MUST-
PRF_AES128_XCBC	SHOULD+	SHOULD
AUTH_HMAC_MD5_96	MAY	MUST NOT
AUTH_HMAC_SHA1_96	MUST	MUST-
AUTH_AES_XCBC_96	SHOULD+	SHOULD
Group 2 (1024-bit)	MUST-	SHOULD NOT
Group 14 (2048-bit)	SHOULD+	MUST

(*) This algorithm is not mentioned in the above sections, so it defaults to MAY.

3. IKEv2 Authentication

IKEv2 authentication may involve a signatures verification. Signatures may be used to validate a certificate or to check the signature of the AUTH value. Cryptographic recommendations regarding certificate validation are out of scope of this document. What is mandatory to implement is provided by the PKIX Community. This document is mostly concerned with signature verification and generation for the authentication.

3.1. IKEv2 Authentication Method

Number	Description	Status
1	RSA Digital Signature	MUST
2	Shared Key Message Integrity Code	MUST
3	DSS Digital Signature	SHOULD NOT
9	ECDSA with SHA-256 on the P-256 curve	SHOULD
10	ECDSA with SHA-384 on the P-384 curve	SHOULD
11	ECDSA with SHA-512 on the P-521 curve	SHOULD
14	Digital Signature	SHOULD

RSA Digital Signature is widely deployed and therefore kept for interoperability. It is expected to be downgraded in the future as its signatures are based on the older RSASSA-PKCS1-v1.5 which is no longer recommended. RSA authentication, as well as other specific

Authentication Methods, are expected to be replaced with the generic Digital Signature method of [RFC7427].

Shared Key Message Integrity Code is widely deployed and mandatory to implement in the IKEv2 in the RFC7296. The status remains MUST.

ECDSA based Authentication Methods are also expected to be downgraded as these do not provide hash function agility. Instead, ECDSA (like RSA) is expected to be performed using the generic Digital Signature method. It’s status is SHOULD.

DSS Digital Signature is bound to SHA-1 and has the same level of security as 1024-bit RSA. It is currently at SHOULD NOT and is expected to be downgraded to MUST NOT in the future.

Digital Signature [RFC7427] is expected to be promoted as it provides hash function, signature format and algorithm agility.Its current status is SHOULD.

3.1.1. Recommendations for RSA key length

Description	Status
RSA with key length 2048	MUST
RSA with key length 3072 and 4096	SHOULD
RSA with key length between 2049 and 4095	MAY
RSA with key length smaller than 2048	SHOULD NOT

The IKEv2 RFC7296 mandates support for the RSA keys of size 1024 or 2048 bits, but key sizes less than 2048 are updated to SHOULD NOT as there is industry-wide trend to deprecate key lengths less than 2048 bits. Since these signatures only have value in real-time, and need no future protection, smaller keys were kept at SHOULD NOT instead of MUST NOT.

3.2. Digital Signature Recommendations

When a Digital Signature authentication method is implemented, the following recommendations are applied for hash functions:

Number	Description	Status	Comment
1	SHA1	MUST NOT	
2	SHA2-256	MUST	
3	SHA2-384	MAY	
4	SHA2-512	SHOULD	

When the Digital Signature authentication method is used with RSA signature algorithm, RSASSA-PSS MUST be supported and RSASSA-PKCS1-v1.5 MAY be supported.

The following table lists recommendations for authentication methods in RFC7427 [RFC7427] notation. These recommendations are applied only if Digital Signature authentication method is implemented.

Description	Status	Comment
RSASSA-PSS with SHA-256	MUST	
ecdsa-with-sha256	SHOULD	
sha1WithRSAEncryption	MUST NOT	
dsa-with-sha1	MUST NOT	
ecdsa-with-sha1	MUST NOT	
RSASSA-PSS with Empty Parameters	MUST NOT	(*)
RSASSA-PSS with Default Parameters	MUST NOT	(*)

(*) Empty or Default parameters means it is using SHA1, which is at level MUST NOT.

4. Algorithms for Internet of Things

Some algorithms in this document are marked for use with the Internet of Things (IoT). There are several reasons why IoT devices prefer a different set of algorithms from regular IKEv2 clients. IoT devices are usually very constrained, meaning the memory size and CPU power is so limited, that these clients only have resources to implement and run one set of algorithms. For example, instead of implementing AES and SHA, these devices typically use AES_XCBC as integrity algorithm so SHA does not need to be implemented.

For example, IEEE Std 802.15.4 [IEEE-802-15-4] devices have a mandatory to implement link level security using AES-CCM with 128 bit keys. The IEEE Recommended Practice for Transport of Key Management Protocol (KMP) Datagrams [IEEE-802-15-9] already provide a way to use

Minimal IKEv2 [RFC7815] over 802.15.4 to provide link keys for the 802.15.4 layer.

These devices might want to use AES-CCM as their IKEv2 algorithm, so they can reuse the hardware implementing it. They cannot use the AES-CBC algorithm, as the hardware quite often do not include support for AES decryption needed to support the CBC mode. So despite the AES-CCM algorithm requiring AEAD [RFC5282] support, the benefit of reusing the crypto hardware makes AES-CCM the preferred algorithm.

Another important aspect of IoT devices is that their transfer rates are usually quite low (in order of tens of kbits/s), and each bit they transmit has an energy consumption cost associated with it and shortens their battery life. Therefore, shorter packets are preferred. This is the reason for recommending the 8 octet ICV over the 16 octet ICV.

Because different IoT devices will have different constraints, this document cannot specify the one mandatory profile for IoT. Instead, this document points out commonly used algorithms with IoT devices.

5. Security Considerations

The security of cryptographic-based systems depends on both the strength of the cryptographic algorithms chosen and the strength of the keys used with those algorithms. The security also depends on the engineering of the protocol used by the system to ensure that there are no non-cryptographic ways to bypass the security of the overall system.

The Diffie-Hellman Group parameter is the most important one to choose conservatively. Any party capturing all IKE and ESP traffic that (even years later) can break the selected DH group in IKE, can gain access to the symmetric keys used to encrypt all the ESP traffic. Therefore, these groups must be chosen very conservatively. However, specifying an extremely large DH group also puts a considerable load on the device, especially when this is a large VPN gateway or an IoT constrained device.

This document concerns itself with the selection of cryptographic algorithms for the use of IKEv2, specifically with the selection of "mandatory-to-implement" algorithms. The algorithms identified in this document as "MUST implement" or "SHOULD implement" are not known to be broken at the current time, and cryptographic research so far leads us to believe that they will likely remain secure into the foreseeable future. However, this isn’t necessarily forever and it is expected that new revisions of this document will be issued from time to time to reflect the current best practice in this area.

6. IANA Considerations

This document renames some of the names in the "Transform Type 1 - Encryption Algorithm Transform IDs" registry of the "Internet Key Exchange Version 2 (IKEv2) Parameters". All the other names have ENCR_ prefix except 3, and all other entries use names in format of uppercase words separated with underscores except 6. This document changes those names to match others.

This document requests IANA to rename following entries for the AES-GCM cipher [RFC4106] and the Camellia cipher [RFC5529]:

Old name	New name
AES-GCM with a 8 octet ICV	ENCR_AES_GCM_8
AES-GCM with a 12 octet ICV	ENCR_AES_GCM_12
AES-GCM with a 16 octet ICV	ENCR_AES_GCM_16
ENCR_CAMELLIA_CCM with an 8-octet ICV	ENCR_CAMELLIA_CCM_8
ENCR_CAMELLIA_CCM with a 12-octet ICV	ENCR_CAMELLIA_CCM_12
ENCR_CAMELLIA_CCM with a 16-octet ICV	ENCR_CAMELLIA_CCM_16

In addition to add this RFC as reference to both ESP Reference and IKEv2 Reference columns for ENCR_AES_GCM entries, keeping the current references there also, and also add this RFC as reference to the ESP Reference column for ENCR_CAMELLIA_CCM entries, keeping the current reference there also.

The final registry entries should be:

Number	Name	ESP Reference	IKEv2 Reference
...			
18	ENCR_AES_GCM_8	[RFC4106][RFCXXXX]	[RFC5282][RFCXXXX]
19	ENCR_AES_GCM_12	[RFC4106][RFCXXXX]	[RFC5282][RFCXXXX]
20	ENCR_AES_GCM_16	[RFC4106][RFCXXXX]	[RFC5282][RFCXXXX]
...			
25	ENCR_CAMELLIA_CCM_8	[RFC5529][RFCXXXX]	-
26	ENCR_CAMELLIA_CCM_12	[RFC5529][RFCXXXX]	-
27	ENCR_CAMELLIA_CCM_16	[RFC5529][RFCXXXX]	-

7. Acknowledgements

The first version of this document was RFC 4307 by Jeffrey I. Schiller of the Massachusetts Institute of Technology (MIT). Much of the original text has been copied verbatim.

We would like to thank Paul Hoffman, Yaron Sheffer, John Mattsson, Tommy Pauly, Eric Rescorla and Pete Resnick for their valuable feedback and reviews.

8. References

8.1. 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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC4106] Viega, J. and D. McGrew, "The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)", RFC 4106, DOI 10.17487/RFC4106, June 2005, <http://www.rfc-editor.org/info/rfc4106>.

[RFC4307] Schiller, J., "Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2)", RFC 4307, DOI 10.17487/RFC4307, December 2005, <http://www.rfc-editor.org/info/rfc4307>.

[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>.

[RFC5282] Black, D. and D. McGrew, "Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol", RFC 5282, DOI 10.17487/RFC5282, August 2008, <http://www.rfc-editor.org/info/rfc5282>.

8.2. Informative References

[RFC7427] Kivinen, T. and J. Snyder, "Signature Authentication in the Internet Key Exchange Version 2 (IKEv2)", RFC 7427, DOI 10.17487/RFC7427, January 2015, <http://www.rfc-editor.org/info/rfc7427>.

[RFC6989] Sheffer, Y. and S. Fluhrer, "Additional Diffie-Hellman Tests for the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 6989, DOI 10.17487/RFC6989, July 2013, <http://www.rfc-editor.org/info/rfc6989>.

[RFC7815] Kivinen, T., "Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation", RFC 7815, DOI 10.17487/RFC7815, March 2016, <http://www.rfc-editor.org/info/rfc7815>.

[RFC5529] Kato, A., Kanda, M., and S. Kanno, "Modes of Operation for Camellia for Use with IPsec", RFC 5529, DOI 10.17487/RFC5529, April 2009, <http://www.rfc-editor.org/info/rfc5529>.

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

[TRANSCRIPTION] Bhargavan, K. and G. Leurent, "Transcript Collision Attacks: Breaking Authentication in TLS, IKE, and SSH", NDSS , feb 2016.

[IEEE-802-15-4] "IEEE Standard for Low-Rate Wireless Personal Area Networks (WPANs)", IEEE Standard 802.15.4, 2015.

[IEEE-802-15-9] "IEEE Recommended Practice for Transport of Key Management Protocol (KMP) Datagrams", IEEE Standard 802.15.9, 2016.

Authors' Addresses

Yoav Nir
Check Point Software Technologies Ltd.
5 Hasolelim st.
Tel Aviv 6789735
Israel

EMail: ynir.ietf@gmail.com

Tero Kivinen
INSIDE Secure
Eerikinkatu 28
HELSINKI FI-00180
FI

EMail: kivinen@iki.fi

Paul Wouters
Red Hat

EMail: pwouters@redhat.com

Daniel Migault
Ericsson
8400 boulevard Decarie
Montreal, QC   H4P 2N2
Canada

Phone: +1 514-452-2160
EMail: daniel.migault@ericsson.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 14, 2017

Y. Nir
Check Point
S. Josefsson
SJD

October 11, 2016

Curve25519 and Curve448 for IKEv2 Key Agreement
draft-ietf-ipsecme-safecurves-05

Abstract

This document describes the use of Curve25519 and Curve448 for ephemeral key exchange in the Internet Key Exchange (IKEv2) protocol.

Status of This Memo

This Internet-Draft will expire on April 14, 2017.

Copyright Notice

Table of Contents

1. Introduction

The "Elliptic Curves for Security" document [RFC7748] describes two elliptic curves: Curve25519 and Curve448, as well as the X25519 and X448 functions for performing key agreement using Diffie-Hellman operations with these curves. The curves and functions are designed for both performance and security.

Elliptic curve Diffie-Hellman [RFC5903] has been specified for the Internet Key Exchange (IKEv2 - [RFC7296]) for almost ten years. RFC 5903 and its predecessor specified the so-called NIST curves. The state of the art has advanced since then. More modern curves allow faster implementations while making it much easier to write constant-time implementations resilient to time-based side-channel attacks. This document defines two such curves for use in IKE. See [Curve25519] for details about the speed and security of the Curve25519 function.

1.1. Conventions Used in This Document

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

2. Curve25519 & Curve448

Implementations of Curve25519 and Curve448 in IKEv2 SHALL follow the steps described in this section. All cryptographic computations are done using the X25519 and X448 functions defined in [RFC7748]. All related parameters (for example, the base point) and the encoding (in particular, pruning the least/most significant bits and use of little-endian encoding) are compliant with [RFC7748].

An ephemeral Diffie-Hellman key exchange using Curve25519 or Curve448 is performed as follows: Each party picks a secret key d uniformly at random and computes the corresponding public key. "X" is used below to denote either X25519 or X448, and "G" is used to denote the corresponding base point:

```
   pub_mine = X(d, G)
```

Parties exchange their public keys (see Section 3.1) and compute a shared secret:

```
      SHARED_SECRET = X(d, pub_peer).
```

This shared secret is used directly as the value denoted g^ir in section 2.14 of RFC 7296. It is 32 octets when Curve25519 is used, and 56 octets when Curve448 is used.

3. Use and Negotiation in IKEv2

The use of Curve25519 and Curve448 in IKEv2 is negotiated using a Transform Type 4 (Diffie-Hellman group) in the SA payload of either an IKE_SA_INIT or a CREATE_CHILD_SA exchange. The value TBA1 is used for the group defined by Curve25519 and the value TBA2 is used for the group defined by Curve448.

3.1. Key Exchange Payload

The diagram for the Key Exchange Payload from section 3.4 of RFC 7296 is copied below for convenience:

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Next Payload  |C|  RESERVED   |         Payload Length        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   Diffie-Hellman Group Num    |           RESERVED            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   ~                       Key Exchange Data                       ~
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Payload Length - For Curve25519 the public key is 32 octets, so the Payload Length field will be 40, and for Curve448 the public key is 56 octets, so the Payload Length field will be 64.

- o The Diffie-Hellman Group Num is TBA1 for Curve25519, or TBA2 for Curve448.
- o The Key Exchange Data is the 32 or 56 octets as described in section 6 of [RFC7748]

3.2. Recipient Tests

Receiving and handling of incompatible point formats MUST follow the considerations described in section 5 of [RFC7748]. In particular, receiving entities MUST mask the most-significant bit in the final byte for X25519 (but not X448), and implementations MUST accept non-canonical values.

4. Security Considerations

Curve25519 and Curve448 are designed to facilitate the production of high-performance constant-time implementations. Implementors are encouraged to use a constant-time implementation of the functions. This point is of crucial importance especially if the implementation chooses to reuse its ephemeral key pair in many key exchanges for performance reasons.

Curve25519 is intended for the ~128-bit security level, comparable to the 256-bit random ECP group (group 19) defined in RFC 5903, also known as NIST P-256 or secp256r1. Curve448 is intended for the ~224-bit security level.

While the NIST curves are advertised as being chosen verifiably at random, there is no explanation for the seeds used to generate them. In contrast, the process used to pick Curve25519 and Curve448 is fully documented and rigid enough so that independent verification can and has been done. This is widely seen as a security advantage, since it prevents the generating party from maliciously manipulating the parameters.

Another family of curves available in IKE that were generated in a fully verifiable way, is the Brainpool curves [RFC6954]. For example, brainpoolP256 (group 28) is expected to provide a level of security comparable to Curve25519 and NIST P-256. However, due to the use of pseudo-random prime, it is significantly slower than NIST P-256, which is itself slower than Curve25519.

5. IANA Considerations

IANA is requested to assign two values from the IKEv2 "Transform Type 4 - Diffie-Hellman Group Transform IDs" registry, with names "Curve25519" and "Curve448" and this document as reference. The Recipient Tests field should also point to this document:

Number	Name	Recipient Tests	Reference
TBA1	Curve25519	RFCxxxx Section 3.2	RFCxxxx
TBA2	Curve448	RFCxxxx Section 3.2	RFCxxxx

Table 1: New Transform Type 4 Values

6. Acknowledgements

Curve25519 was designed by D. J. Bernstein and the parameters for Curve448 ("Goldilocks") were defined by Mike Hamburg. The specification of algorithms, wire format and other considerations are documented in RFC 7748 by Adam Langley, Mike Hamburg, and Sean Turner.

The example in Appendix A was calculated using the master version of OpenSSL, retrieved on August 4th, 2016.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC7296] Kivinen, T., Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 7296, October 2014.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, January 2016.

7.2. Informative References

[Curve25519]
Bernstein, J., "Curve25519: New Diffie-Hellman Speed Records", LNCS 3958, February 2006, <http://dx.doi.org/10.1007/11745853_14>.

[RFC5903] Fu, D. and J. Solinas, "Elliptic Curve Groups modulo a Prime (ECP Groups) for IKE and IKEv2", RFC 5903, June 2010.

[RFC6954] Merkle, J. and M. Lochter, "Using the Elliptic Curve Cryptography (ECC) Brainpool Curves for the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 6954, July 2013.

Appendix A. Numerical Example for Curve25519

Suppose we have both the initiator and the responder generating private keys by generating 32 random octets. As usual in IKEv2 and its extension, we will denote Initiator values with the suffix _i and responder values with the suffix _r:

```
     random_i = 75 1f b4 30 86 55 b4 76 b6 78 9b 73 25 f9 ea 8c
                dd d1 6a 58 53 3f f6 d9 e6 00 09 46 4a 5f 9d 94

     random_r = 0a 54 64 52 53 29 0d 60 dd ad d0 e0 30 ba cd 9e
                55 01 ef dc 22 07 55 a1 e9 78 f1 b8 39 a0 56 88
```

These numbers need to be fixed by unsetting some bits as described in section 5 of RFC 7748. This affects only the first and last octets of each value:

```
     fixed_i =  70 1f b4 30 86 55 b4 76 b6 78 9b 73 25 f9 ea 8c
                dd d1 6a 58 53 3f f6 d9 e6 00 09 46 4a 5f 9d 54

     fixed_r =  08 54 64 52 53 29 0d 60 dd ad d0 e0 30 ba cd 9e
                55 01 ef dc 22 07 55 a1 e9 78 f1 b8 39 a0 56 48
```

The actual private keys are considered to be encoded in little-endian format:

```
  d_i = 549D5F4A460900E6D9F63F53586AD1DD8CEAF925739B78B676B4558630B41F70

  d_r = 4856A039B8F178E9A1550722DCEF01559ECDBA30E0D0ADDD600D295352645408
```

The public keys are generated from this using the formula in Section 2:

```
   pub_i = X25519(d_i, G) =
                48 d5 dd d4 06 12 57 ba 16 6f a3 f9 bb db 74 f1
                a4 e8 1c 08 93 84 fa 77 f7 90 70 9f 0d fb c7 66

   pub_r = X25519(d_r, G) =
                0b e7 c1 f5 aa d8 7d 7e 44 86 62 67 32 98 a4 43
                47 8b 85 97 45 17 9e af 56 4c 79 c0 ef 6e ee 25
```

And this is the value of the Key Exchange Data field in the key exchange payload described in Section 3.1. The shared value is calculated as in Section 2:

```
   SHARED_SECRET = X25519(d_i, pub_r) = X25519(d_r, pub_i) =
                c7 49 50 60 7a 12 32 7f-32 04 d9 4b 68 25 bf b0
                68 b7 f8 31 9a 9e 37 08-ed 3d 43 ce 81 30 c9 50
```

Authors' Addresses

Yoav Nir
Check Point Software Technologies Ltd.
5 Hasolelim st.
Tel Aviv 6789735
Israel

Email: ynir.ietf@gmail.com

Simon Josefsson
SJD AB

Email: simon@josefsson.org

Network
Internet-Draft
Intended status: Standards Track
Expires: October 27, 2016

T. Pauly
Apple Inc.
S. Touati
Ericsson
R. Mantha
Cisco Systems

April 25, 2016

TCP Encapsulation of IKEv2 and IPSec Packets
draft-pauly-ipsecme-tcp-encaps-04

Abstract

This document describes a method to transport IKEv2 and IPSec packets over a TCP connection for traversing network middleboxes that may block IKEv2 negotiation over UDP. This method, referred to as TCP encapsulation, involves sending all packets for tunnel establishment as well as tunneled packets over a TCP connection. This method is intended to be used as a fallback option when IKE cannot be negotiated over UDP.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

IKEv2 [RFC7296] is a protocol for establishing IPSec tunnels, using IKE messages over UDP for control traffic, and using Encapsulating Security Payload (ESP) messages for its data traffic. Many network middleboxes that filter traffic on public hotspots block all UDP traffic, including IKEv2 and IPSec, but allow TCP connections through since they appear to be web traffic. Devices on these networks that need to use IPSec (to access private enterprise networks, to route voice-over-IP calls to carrier networks, or because of security policies) are unable to establish IPSec tunnels. This document defines a method for encapsulating both the IKEv2 control messages as well as the IPSec data messages within a TCP connection.

Using TCP as a transport for IPSec packets adds a third option to the list of traditional IPSec transports:

1. Direct. Currently, IKEv2 negotiations begin over UDP port 500. If no NAT is detected between the initiator and the receiver, then subsequent IKEv2 packets are sent over UDP port 500 and IPSec data packets are sent using ESP [RFC4303].

2. UDP Encapsulation [RFC3948]. If a NAT is detected between the initiator and the receiver, then subsequent IKEv2 packets are sent over UDP port 4500 with four bytes of zero at the start of the UDP payload and ESP packets are sent out over UDP port 4500. Some peers default to using UDP encapsulation even when no NAT are detected on the path as some middleboxes do not support IP protocols other than TCP and UDP.

3. TCP Encapsulation. If both of the other two methods are not available or appropriate, both IKEv2 negotiation packets as well as ESP packets can be sent over a single TCP connection to the peer.

Direct use of ESP or UDP Encapsulation should be preferred by IKEv2 implementations due to performance concerns when using TCP Encapsulation Section 12. Most implementations should use TCP Encapsulation only on networks where negotiation over UDP has been attempted without receiving responses from the peer, or if a network is known to not support UDP.

1.1. Prior Work and Motivation

Encapsulating IKEv2 connections within TCP streams is a common approach to solve the problem of UDP packets being blocked by network middleboxes. The goal of this document is to promote interoperability by providing a standard method of framing IKEv2 and ESP message within streams, and to provide guidelines for how to configure and use TCP encapsulation.

Some previous solutions include:

Cellular Network Access Interworking Wireless LAN (IWLAN) uses IKEv2 to create secure connections to cellular carrier networks for making voice calls and accessing other network services over Wi-Fi networks. 3GPP has recommended that IKEv2 and ESP packets be sent within a TLS connection to be able to establish connections on restrictive networks.

ISAKMP over TCP Various non-standard extensions to ISAKMP have been deployed that send IPSec traffic over TCP or TCP-like packets.

SSL VPNs Many proprietary VPN solutions use a combination of TLS and IPSec in order to provide reliability.

IKEv2 over TCP IKEv2 over TCP as described in [I-D.nir-ipsecme-ike-tcp] is used to avoid UDP fragmentation.

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. Configuration

One of the main reasons to use TCP encapsulation is that UDP traffic may be entirely blocked on a network. Because of this, support for TCP encapsulation is not specifically negotiated in the IKEv2 exchange. Instead, support for TCP encapsulation must be pre-configured on both the initiator and the responder.

The configuration defined on each peer should include the following parameters:

- One or more TCP ports on which the responder will listen for incoming connections. Note that the initiator may initiate TCP connections to the responder from any local port.

- Optionally, an extra framing protocol to use on top of TCP to further encapsulate the stream of IKEv2 and IPSec packets. See Appendix A for a detailed discussion.

This document leaves the selection of TCP ports up to implementations. It’s suggested to use TCP port 4500, which is allocated for IPSec NAT Traversal.

Since TCP encapsulation of IKEv2 and IPSec packets adds overhead and has potential performance trade-offs compared to direct or UDP-encapsulated tunnels (as described in Performance Considerations, Section 12), when possible, implementations SHOULD prefer IKEv2 direct or UDP encapsulated tunnels over TCP encapsulated tunnels.

3. TCP-Encapsulated Header Formats

In order to encapsulate IKEv2 and ESP messages within a TCP stream, a 16-bit length field precedes every message. If the first 32-bits of the message are zeros (a Non-ESP Marker), then the contents comprise an IKEv2 message. Otherwise, the contents comprise an ESP message. Authentication Header (AH) messages are not supported for TCP encapsulation.

Although a TCP stream may be able to send very long messages, implementations SHOULD limit message lengths to typical UDP datagram ESP payload lengths. The maximum message length is used as the effective MTU for connections that are being encrypted using ESP, so the maximum message length will influence characteristics of inner connections, such as the TCP Maximum Segment Size (MSS).

3.1. TCP-Encapsulated IKEv2 Header Format

```
                     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
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Non-ESP Marker                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                      IKEv2 header [RFC7296]                   ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1

The IKE header is preceded by a 16-bit length field in network byte order that specifies the length of the IKE packet within the TCP stream. As with IKEv2 over UDP port 4500, a zeroed 32-bit Non-ESP Marker is inserted before the start of the IKEv2 header in order to differentiate the traffic from ESP traffic between the same addresses and ports.

o Length (2 octets, unsigned integer) - Length of the IKE packet including the Length Field and Non-ESP Marker.

3.2. TCP-Encapsulated ESP Header Format

```
                        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
                                   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                   |            Length             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   ~                      ESP header [RFC4303]                     ~
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2

The ESP header is preceded by a 16-bit length field in network byte order that specifies the length of the ESP packet within the TCP stream.

- Length (2 octets, unsigned integer) - Length of the ESP packet including the Length Field.

4. TCP-Encapsulated Stream Prefix

Each TCP stream used for IKEv2 and IPSec encapsulation MUST begin with a fixed sequence of five bytes as a magic value, containing the characters "IKEv2" as ASCII values. This allows peers to differentiate this protocol from other protocols that may be run over TCP streams, since the value does not overlap with the valid start of any other known stream protocol. This value is only sent once, by the Initiator only, at the beginning of any TCP stream.

```
    0      1      2      3      4
   +------+------+------+------+------+
   | 0x69 | 0x6b | 0x65 | 0x76 | 0x32 |
   +------+------+------+------+------+
```

Figure 3

5. Applicability

TCP encapsulation is applicable only when it has been configured to be used with specific IKEv2 peers. If a responder is configured to use TCP encapsulation, it MUST listen on the configured port(s) in case any peers will initiate new IKEv2 sessions. Initiators MAY use TCP encapsulation for any IKEv2 session to a peer that is configured to support TCP encapsulation, although it is recommended that initiators should only use TCP encapsulation when traffic over UDP is blocked.

Any specific IKE SA, along with its Child SAs, is either TCP encapsulated or not. A mix of TCP and UDP encapsulation for a single SA is not allowed.

6. Connection Establishment and Teardown

When the IKEv2 initiator uses TCP encapsulation for its negotiation, it will initiate a TCP connection to the responder using the configured TCP port. The first bytes sent on the stream MUST be the stream prefix value [Section 4]. After this prefix, encapsulated IKEv2 messages will negotiate the IKE SA and initial Child SA [RFC7296]. After this point, both encapsulated IKE Figure 1 and ESP Figure 2 messages will be sent over the TCP connection.

In order to close an IKE session, either the initiator or responder SHOULD gracefully tear down IKE SAs with DELETE payloads. Once all SAs have been deleted, the initiator of the original connection MUST close the TCP connection.

An unexpected FIN or a RST on the TCP connection may indicate either a loss of connectivity, an attack, or some other error. If a DELETE payload has not been sent, both sides SHOULD maintain the state for their SAs for the standard lifetime or time-out period. The original initiator (that is, the endpoint that initiated the TCP connection and sent the first IKE_SA_INIT message) is responsible for re-establishing the TCP connection if it is torn down for any unexpected reason. Since new TCP connections may use different ports due to NAT mappings or local port allocations changing, the responder MUST allow packets for existing SAs to be received from new source ports.

A peer MUST discard a partially received message due to a broken connection.

The streams of data sent over any TCP connection used for this protocol MUST begin with the stream prefix value followed by a complete message, which is either an encapsulated IKE or ESP message. If the connection is being used to resume a previous IKE session, the responder can recognize the session using either the IKE SPI from an encapsulated IKE message or the ESP SPI from an encapsulated ESP message. If the session had been fully established previously, it is suggested that the initiator send an UPDATE_SA_ADDRESSES message if MOBIKE is supported, or an INFORMATIONAL message (a keepalive) otherwise. If either initiator or responder receives a stream that cannot be parsed correctly, it MUST close the TCP connection.

Multiple TCP connections between the initiator and the responder are allowed, but their use must take into account the initiator capabilities and the deployment model such as to connect to multiple gateways handling different ESP SAs when deployed in a high availability model. It is also possible to negotiate multiple IKE SAs over the same TCP connection.

The processing of the TCP packets is the same whether its within a single or multiple TCP connections.

7. Interaction with NAT Detection Payloads

When negotiating over UDP port 500, IKE_SA_INIT packets include NAT_DETECTION_SOURCE_IP and NAT_DETECTION_DESTINATION_IP payloads to determine if UDP encapsulation of IPSec packets should be used. These payloads contain SHA-1 digests of the SPIs, IP addresses, and ports. IKE_SA_INIT packets sent on a TCP connection SHOULD include these payloads, and SHOULD use the applicable TCP ports when creating and checking the SHA-1 digests.

If a NAT is detected due to the SHA-1 digests not matching the expected values, no change should be made for encapsulation of subsequent IKEv2 or ESP packets, since TCP encapsulation inherently supports NAT traversal. Implementations MAY use the information that a NAT is present to influence keep-alive timer values.

8. Using MOBIKE with TCP encapsulation

When an IKEv2 session is transitioned between networks using MOBIKE [RFC4555], the initiator of the transition may switch between using TCP encapsulation, UDP encapsulation, or no encapsulation. Implementations that implement both MOBIKE and TCP encapsulation MUST support dynamically enabling and disabling TCP encapsulation as interfaces change.

The encapsulation method of ESP packets MUST always match the encapsulation method of the IKEv2 negotiation, which may be different when an IKEv2 endpoint changes networks. When a MOBIKE-enabled initiator changes networks, the UPDATE_SA_ADDRESSES notification SHOULD be sent out first over UDP before attempting over TCP. If there is a response to the UPDATE_SA_ADDRESSES notification sent over UDP, then the ESP packets should be sent directly over IP or over UDP port 4500 (depending on if a NAT was detected), regardless of if a connection on a previous network was using TCP encapsulation. Similarly, if the responder only responds to the UPDATE_SA_ADDRESSES notification over TCP, then the ESP packets should be sent over the TCP connection, regardless of if a connection on a previous network did not use TCP encapsulation.

9. Using IKEv2 Message Fragmentation with TCP encapsulation

IKEv2 Message Fragmentation [RFC7383] is not required when using TCP encapsulation, since a TCP stream already handles the fragmentation of its contents across packets. Since fragmentation is redundant in this case, implementations might choose to not negotiate IKEv2 fragmentation. Even if fragmentation is negotiated, an implementation MAY choose to not fragment when going over a TCP connection.

If an implementation supports both MOBIKE and IKEv2 fragmentation, it SHOULD negotiate IKEv2 fragmentation over a TCP encapsulated session in case the session switches to UDP encapsulation on another network.

10. Considerations for Keep-alives and DPD

Encapsulating IKE and IPSec inside of a TCP connection can impact the strategy that implementations use to detect peer liveness and to maintain middlebox port mappings. Peer liveness should be checked using IKEv2 Informational packets [RFC7296].

In general, TCP port mappings are maintained by NATs longers than UDP port mappings, so IPSec ESP NAT keep-alives [RFC3948] SHOULD NOT be sent when using TCP encapsulation. Any implementation using TCP encapsulation MUST silently drop incoming NAT keep-alive packets, and not treat them as errors. NAT keep-alive packets over a TCP encapsulated IPSec connection will be sent with a length value of 1 byte, whose value is 0xFF [Figure 2].

Note that depending on the configuration of TCP and TLS on the connection, TCP keep-alives [RFC1122] and TLS keep-alives [RFC6520] may be used. These MUST NOT be used as indications of IKEv2 peer liveness.

11. Middlebox Considerations

Many security networking devices such as Firewalls or Intrusion Prevention Systems, network optimization/acceleration devices and Network Address Translation (NAT) devices keep the state of sessions that traverse through them.

These devices commonly track the transport layer and/or the application layer data to drop traffic that is anomalous or malicious in nature.

A network device that monitors up to the application layer will commonly expect to see HTTP traffic within a TCP socket running over port 80, if non-HTTP traffic is seen (such as TCP encapsulated IKEv2), this could be dropped by the security device.

A network device that monitors the transport layer will track the state of TCP sessions, such as TCP sequence numbers. TCP encapsulation of IKEv2 should therefore use standard TCP behaviors to avoid being dropped by middleboxes.

12. Performance Considerations

Several aspects of TCP encapsulation for IKEv2 and IPSec packets may negatively impact the performance of connections within the tunnel. Implementations should be aware of these and take these into consideration when determining when to use TCP encapsulation.

12.1. TCP-in-TCP

If the outer connection between IKEv2 peers is over TCP, inner TCP connections may suffer effects from using TCP within TCP. In particular, the inner TCP's round-trip-time estimation will be affected by the burstiness of the outer TCP. This will make loss-recovery of the inner TCP traffic less reactive and more prone to spurious retransmission timeouts.

12.2. Added Reliability for Unreliable Protocols

Since ESP is an unreliable protocol, transmitting ESP packets over a TCP connection will change the fundamental behavior of the packets. Some application-level protocols that prefer packet loss to delay (such as Voice over IP or other real-time protocols) may be negatively impacted if their packets are retransmitted by the TCP connection due to packet loss.

12.3. Quality of Service Markings

Quality of Service (QoS) markings, such as DSCP and Traffic Class, should be used with care on TCP connections used for encapsulation. Individual packets SHOULD NOT use different markings than the rest of the connection, since packets with different priorities may be routed differently and cause unnecessary delays in the connection.

12.4. Maximum Segment Size

A TCP connection used for IKEv2 encapsulation SHOULD negotiate its maximum segment size (MSS) in order to avoid unnecessary fragmentation of packets.

13. Security Considerations

IKEv2 responders that support TCP encapsulation may become vulnerable to new Denial-of-Service (DoS) attacks that are specific to TCP, such as SYN-flooding attacks. Responders should be aware of this additional attack-surface.

Attackers may be able to disrupt the TCP connection by sending spurious RST packets. Due to this, implementations SHOULD make sure that IKE session state persists even if the underlying TCP connection is torn down.

14. IANA Considerations

This memo includes no request to IANA.

TCP port 4500 is already allocated to IPSec. This port MAY be used for the protocol described in this document, but implementations MAY prefer to use other ports based on local policy.

15. Acknowledgments

The authors would like to acknowledge the input and advice of Stuart Cheshire, Delziel Fernandes, Yoav Nir, Christoph Paasch, Yaron Sheffer, David Schinazi, Graham Bartlett, Byju Pularikkal, March Wu and Kingwel Xie. Special thanks to Eric Kinnear for his implementation work.

16. References

16.1. 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, <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>.

16.2. Informative References

[I-D.nir-ipsecme-ike-tcp] Nir, Y., "A TCP transport for the Internet Key Exchange", draft-nir-ipsecme-ike-tcp-01 (work in progress), July 2012.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <http://www.rfc-editor.org/info/rfc1122>.

[RFC2817] Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, DOI 10.17487/RFC2817, May 2000, <http://www.rfc-editor.org/info/rfc2817>.

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

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <http://www.rfc-editor.org/info/rfc4303>.

[RFC4555] Eronen, P., "IKEv2 Mobility and Multihoming Protocol (MOBIKE)", RFC 4555, DOI 10.17487/RFC4555, June 2006, <http://www.rfc-editor.org/info/rfc4555>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, DOI 10.17487/RFC6520, February 2012, <http://www.rfc-editor.org/info/rfc6520>.

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

Appendix A. Using TCP encapsulation with TLS

This section provides recommendations on the support of TLS with the TCP encapsulation.

When using TCP encapsulation, implementations may choose to use TLS [RFC5246], to be able to traverse middle-boxes, which may block non HTTP traffic.

If a web proxy is applied to the ports for the TCP connection, and TLS is being used, the initiator can send an HTTP CONNECT message to establish a tunnel through the proxy [RFC2817].

The use of TLS should be configurable on the peers. The responder may expect to read encapsulated IKEv2 and ESP packets directly from the TCP connection, or it may expect to read them from a stream of TLS data packets. The initiator should be pre-configured to use TLS or not when communicating with a given port on the responder.

When new TCP connections are re-established due to a broken connection, TLS must be re-negotiated. TLS Session Resumption is recommended to improve efficiency in this case.

The security of the IKEv2 session is entirely derived from the IKVEv2 negotiation and key establishment, therefore When TLS is used on the TCP connection, both the initiator and responder MUST allow for the NULL cipher to be selected.

Implementations must be aware that the use of TLS introduces another layer of overhead requiring more bytes to transmit a given IKEv2 and IPSec packet.

Appendix B. Example exchanges of TCP Encapsulation with TLS

B.1. Establishing an IKEv2 session

```
                   Client                               Server
                 ----------                           ----------
         --------------------  TCP Connection  -------------------
     1)  (IP_I:Port_I  -> IP_R:TCP443 or TCP4500)
         TcpSyn                    ---------->
                                   <----------         TcpSyn,Ack
         TcpAck                    ---------->

         ---------------------  TLS Session  ---------------------
     2)  ClientHello               ---------->
                                                       ServerHello
                                                      Certificate*
                                                ServerKeyExchange*
                                   <----------     ServerHelloDone
         ClientKeyExchange
         CertificateVerify*
         [ChangeCipherSpec]
         Finished                  ---------->
                                                [ChangeCipherSpec]
                                   <----------            Finished

         ---------------------- IKEv2 Session --------------------
     3)  IKE_SA_INIT               ---------->
         HDR, SAi1, KEi, Ni,
         [N(NAT_DETECTION_*_IP)]
                                   <----------         IKE_SA_INIT
                                               HDR, SAr1, KEr, Nr,
                                            [N(NAT_DETECTION_*_IP)]
         first IKE_AUTH            ---------->
         HDR, SK {IDi, [CERTREQ]
         CP(CFG_REQUEST), IDr,
         SAi2, TSi, TSr, ...}
                                   <----------      first IKE_AUTH
                                       HDR, SK {IDr, [CERT], AUTH,
                                              EAP, SAr2, TSi, TSr}
         EAP                       ---------->
         repeat 1..N times
                                   <----------                 EAP
         final IKE_AUTH            ---------->
         HDR, SK {AUTH}
                                   <----------      final IKE_AUTH
                                     HDR, SK {AUTH, CP(CFG_REPLY),
                                                SA, TSi, TSr, ...}
         --------------   IKEv2 Tunnel Established   -------------
```

Figure 4

1. Client establishes a TCP connection with the server on port 443 or 4500.

2. Client initiates TLS handshake. During TLS handshake, the server SHOULD NOT request the client’s’ certificate, since authentication is handled as part of IKEv2 negotiation.

3. Client and server establish an IKEv2 connection. This example shows EAP-based authentication, although any authentication type may be used.

B.2. Deleting an IKEv2 session

```
                   Client                                Server
                 ----------                            ----------
         ---------------------- IKEv2 Session --------------------
     1)  INFORMATIONAL              ---------->
         HDR, SK {[N,] [D,]
                [CP,] ...}
                                    <----------       INFORMATIONAL
                                                 HDR, SK {[N,] [D,]
                                                         [CP], ...}

         ---------------------  TLS Session  ---------------------
     2)  close_notify               ---------->
                                    <----------        close_notify
         --------------------  TCP Connection  -------------------
     3)  TcpFin                     ---------->
                                    <----------                 Ack
                                    <----------              TcpFin
         Ack                        ---------->
         --------------------- Tunnel Deleted  -------------------
```

Figure 5

1. Client and server exchange INFORMATIONAL messages to notify IKE SA deletion.

2. Client and server negotiate TLS session deletion using TLS CLOSE_NOTIFY.

3. The TCP connection is torn down.

Unless the TCP connection and/or TLS session are being used for multiple IKE SAs, the deletion of the IKE SA should lead to the disposal of the underlying TLS and TCP state.

B.3. Re-establishing an IKEv2 session

```
                  Client                                    Server
                ----------                                ----------
        --------------------  TCP Connection  -------------------
    1)  (IP_I:Port_I  -> IP_R:TCP443 or TCP4500)
        TcpSyn                     ---------->
                                   <----------          TcpSyn,Ack
        TcpAck                     ---------->
        ---------------------  TLS Session  ---------------------
    2)  ClientHello                ---------->
                                   <----------         ServerHello
                                                [ChangeCipherSpec]
                                                          Finished
        [ChangeCipherSpec]         ---------->
        Finished
    3)  <--------------------> IKEv2/ESP flow <----------------->
```

Figure 6

1. If a previous TCP connection was broken (for example, due to a RST), the client is responsible for re-initiating the TCP connection. The initiator's address and port (IP_I and Port_I) may be different from the previous connection's address and port.

2. In ClientHello TLS message, the client SHOULD send the Session ID it received in the previous TLS handshake if available. It is up to the server to perform either an abbreviated handshake or full handshake based on the session ID match.

3. After TCP and TLS are complete, the IKEv2 and ESP packet flow can resume. If MOBIKE is being used, the initiator SHOULD send UPDATE_SA_ADDRESSES.

B.4. Using MOBIKE between UDP and TCP Encapsulation

```
                    Client                                    Server
                  ----------                                ----------
        (IP_I1:UDP500 -> IP_R:UDP500)
    1)  ----------------- IKE_SA_INIT Exchange -----------------
        (IP_I1:UDP4500 -> IP_R:UDP4500)
        Intial IKE_AUTH          ----------->
        HDR, SK { IDi, CERT, AUTH,
        CP(CFG_REQUEST),
        SAi2, TSi, TSr,
        N(MOBIKE_SUPPORTED) }
                                 <-----------    Initial IKE_AUTH
                                       HDR, SK { IDr, CERT, AUTH,
                                             EAP, SAr2, TSi, TSr,
                                            N(MOBIKE_SUPPORTED) }
        <--------------- IKEv2 tunnel establishment ------------>

    2)  ------------ MOBIKE Attempt on new network --------------
        (IP_I2:UDP4500 -> IP_R:UDP4500)
        INFORMATIONAL            ----------->
        HDR, SK { N(UPDATE_SA_ADDRESSES),
        N(NAT_DETECTION_SOURCE_IP),
        N(NAT_DETECTION_DESTINATION_IP) }


    3)  --------------------  TCP Connection  -------------------
        (IP_I2:PORT_I -> IP_R:TCP443 or TCP4500)
        TcpSyn                   ----------->
                                 <-----------          TcpSyn,Ack
        TcpAck                   ----------->

    4)  ---------------------  TLS Session  ---------------------
        ClientHello              ----------->
                                                      ServerHello
                                                     Certificate*
                                               ServerKeyExchange*
                                 <-----------     ServerHelloDone
        ClientKeyExchange
        CertificateVerify*
        [ChangeCipherSpec]
        Finished                 ----------->
                                               [ChangeCipherSpec]
                                 <-----------            Finished
    5)  ---------------------- IKEv2 Session --------------------
        INFORMATIONAL            ----------->
        HDR, SK { N(UPDATE_SA_ADDRESSES),
        N(NAT_DETECTION_SOURCE_IP),
        N(NAT_DETECTION_DESTINATION_IP) }

                                 <-----------       INFORMATIONAL
                            HDR, SK { N(NAT_DETECTION_SOURCE_IP),
                                N(NAT_DETECTION_DESTINATION_IP) }
    6)  <---------------- IKEv2/ESP data flow ------------------>
```

Figure 7

1. During the IKE_SA_INIT exchange, the client and server exchange MOBIKE_SUPPORTED notify payloads to indicate support for MOBIKE.

2. The client changes its point of attachment to the network, and receives a new IP address. The client attempts to re-establish the IKEv2 session using the UPDATE_SA_ADDRESSES notify payload, but the server does not respond because the network blocks UDP traffic.

3. The client beings up a TCP connection to the server in order to use TCP encapsulation.

4. The client initiates and TLS handshake with the server.

5. The client sends the UPDATE_SA_ADDRESSES notify payload on the TCP encapsulated connection.

6. The IKEv2 and ESP packet flow can resume.

Authors' Addresses

Tommy Pauly
Apple Inc.
1 Infinite Loop
Cupertino, California 95014
US

Email: tpauly@apple.com

Samy Touati
Ericsson
300 Holger Way
San Jose, California 95134
US

Email: samy.touati@ericsson.com

Ravi Mantha
Cisco Systems
SEZ, Embassy Tech Village
Panathur, Bangalore 560 037
India

Email: ramantha@cisco.com

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

V. Smyslov
ELVIS-PLUS
March 4, 2019

Using compression in the Internet Key Exchange Protocol Version 2 (IKEv2)
draft-smyslov-ipsecme-ikev2-compression-07

Abstract

This document describes a method for reducing the size of the IKEv2 messages by using lossless compression. Making IKEv2 messages smaller is desirable for low power consumption battery powered devices. It also helps to avoid IP fragmentation of IKEv2 messages. This document describes how compression is negotiated maintaining backward compatibility and how it is used in IKEv2.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

Sending large UDP messages may cause IP fragmentation to take place, that would interact badly with some Network Address Translators (NAT). One of the possible solutions to the problem is IKEv2 fragmentation described in [RFC7383]. However, the IKEv2 fragmentation cannot be used for unencrypted messages and thus cannot be used in the initial IKEv2 exchange (IKE_SA_INIT). Usually the IKE_SA_INIT messages are relatively small and this restriction doesn’t cause problems. However with adoption of more and more new algorithms and new IKEv2 extensions there is a tendency for these messages to grow up in size.

The lossless compression can be used to reduce the size of the IKEv2 messages. Each IKEv2 message contains different types of data structured in payloads. Depending on the type of payload the compressibility of the data it contains varies greatly. Some types of payloads, like the Nonce payload, contain random or pseudo-random data that is almost uncompressible. On the other hand, such payloads like the Security Association payload or Notification payload usually have a lot of redundancy in their encoding and hence are highly compressible. Since many emerging IKEv2 extensions add new type of notification or new parameter to the Security Association payload contained in the IKE_SA_INIT messages, the ability to compress these messages would help keep their size bounded.

Compression can also be applied to the messages followed the IKE_SA_INIT exchange. In this case the reduced size of the messages would make the necessity to use the IKEv2 fragmentation less likely or would decrease the number of fragments the messages are splitted into, increasing the protocol reliability and productivity.

Another field where using compression may be useful is the Internet of Things (IoT) devices utilizing a lower power consumption technology. For many such devices the power consumption for transmitting extra bits over network is much higher than the power consumption for spending extra CPU cycles to compress data before transmission. The appendix A of [IPSEC-IOT-REQS] gives some estimate data. Since many such devices are battery powered without ability to recharge or to replace the battery which serves for the lifecycle of the device (a few years), the task of reducing the power consumption for such devices is very important.

This document specifies how lossless compression is used in IKEv2. In order to enable compression in the IKE_SA_INIT exchange a new payload is introduced that contains other payloads in compressed form. The processing of the Encrypted payload is modified to accommodate compression in subsequent exchanges. The document also specifies how the use of compression is negotiated between the peers maintaining backward compatibility.

2. Terminology and Notation

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

3. Protocol Description

Compression is accommodated differently in the initial IKEv2 exchange and in subsequent exchanges. The difference comes out from the fact, that the messages of all the IKEv2 exchanges except for the initial exchange contain the Encrypted payload. In this case the compression is added as an additional step while constructing the Encrypted payload. The initial IKEv2 exchange requires introduction a new payload, which would contain other payloads in compressed form.

3.1. Using Compression in the IKE_SA_INIT Exchange

The use of compression is not negotiated in a usual for IKEv2 manner - by exchanging appropriate Notification or Vendor ID payloads. Instead a different negotiation mechanism is used.

If an Initiator wants to use compression for the IKE SA being created, it constructs the IKE_SA_INIT request message in a following way. A new payload which is called Compressed payload and described in the Section 4.1 is included into the request message. This payload contains other payloads in compressed form as well as an indication of what compression algorithm is used. When selecting compression algorithm the Initiator must guess what algorithms are supported by the peer and choose an appropriate one. If the guess is wrong the Responder informs about this fact and the mutually appropriate algorithm is then negotiated by the cost of an extra round trip and a message recompression. The Critical bit in the Compressed payload header MUST be set to 1.

```
Initiator
---------
HDR, C!{SA, KE, [N+,] [V+]}, Ni, [N+,] [V+]  -->
```

Not all payloads that are usually present in the IKE_SA_INIT messages are subject for compression. Some payloads contain random or pseudo-random data that is almost uncompressible. Other payloads must be processed as early as possible, before the responder spends resources decompressing them. In particular, the Nonce payload the Puzzle Solution payload and the COOKIE notification payload MUST NOT be included into the Compressed payload. Obviously, if the compression algorithm ID is from private range (241-255), then the corresponding Vendor ID payload MUST NOT be included into the Compressed payload either. See Section 5 for more details about interaction compression with other IKEv2 extensions.

If the Responder doesn't support IKEv2 compression, then it is expected to return the UNSUPPORTED_CRITICAL_PAYLOAD notification in response to such request message, as prescribed in the Section 2.5 of [RFC7296]. Depending on the implementation it may also return the INVALID_SYNTAX notification or doesn't respond at all.

```
                                                        Legacy Responder
                                                        ----------------
                               <--  HDR, N(UNSUPPORTED_CRITICAL_PAYLOAD)

                                                                     or

                                             <--  HDR, N(INVALID_SYNTAX)

                                                                     or

                                                          (No response)
```

If the Initiator receives the UNSUPPORTED_CRITICAL_PAYLOAD notification with the Compressed payload type in its notification data or if it receives the INVALID_SYNTAX notification or if it receives no response after several retransmissions then the Initiator MUST restart the IKE_SA_INIT exchange with no compression.

If the Responder supports IKEv2 compression, but doesn’t support the particular compression algorithm the Initiator has chosen, then the Responder sends back a new error notification: INVALID_COMPRESSION_ALGORITHM. This notification is described in the Section 4.2. Its notification data contains the list of IDs of compression algorithms supported by the Responder.

```
                                                               Responder
                                                               ---------
                              <--  HDR, N(INVALID_COMPRESSION_ALGORITHM)
```

If the Initiator receives the INVALID_COMPRESSION_ALGORITHM notification, then it looks through the list of algorithms included into the notification data and selects an appropriate one. After that it MUST restart the IKE_SA_INIT exchange using the newly selected algorithm for compression. If no mutually appropriate algorithms are found, then the Initiator MUST restart the IKE_SA_INIT exchange with no compression.

Once the Responder receives the IKE_SA_INIT request with appropriate compression algorithm in the Compressed payload, the included payloads are decompressed and along with the outer payloads form the uncompressed request message, which is then processed as usual. If the Responder agrees to use compression in the SA being created then the Responder MUST include the Compressed payload in the response message. The compression algorithm indicated in the Compressed payload MUST be the algorithm from the request.

```
                                                       Responder
                                                       ---------
                      <--  HDR, C!{SA, KE, [N+,] [V+]}, Nr, [N+,] [V+]
```

If for some reason the Responder doesn't want to use compression in the SA being created (e.g. using compression is disabled by administrator) then it MUST send back an uncompressed IKE_SA_INIT response message. In this case the endpoints MUST NOT use compression in subsequent exchanges.

3.2. Using Compression in Subsequent Exchanges

Once the endpoints have used compression in the IKE_SA_INIT exchange, they may continue to use it in subsequent exchanges. However compression is used differently in these exchanges. Messages of every IKEv2 exchange except for the initial exchange are protected by the Encrypted payload. With compression the rules for forming and processing of the Encrypted payload are modified as follows.

The content of the Encrypted payload is compressed before it is encrypted and authenticated. According to the IKEv2 specification the Next Payload field in an Encrypted payload indicates the payload type of the first payload inside the Encrypted payload. If case of using compression, the Next Payload field in the Encrypted payload MUST be set to XXX (TBA by IANA) - the value for the payload type of a Compressed payload. However, the Compressed payload itself MUST NOT appear inside the Encrypted payload, only its payload type is used to indicate that the content of the Encrypted payload was compressed before encryption.

Since in this case the Next Payload field in the Encrypted payload no longer indicates a type of the first inner payload, this information is moved to the Next Payload field of the last inner payload (which is set zero in the IKEv2 specification). This modification is done before the payloads are compressed.

```
 Uncompressed:  SK(Next=P1) {P1(Next=P2), P2(Next=P3), ... Pn(Next=0)}
 Compressed:    SK(Next=C)  {P1(Next=P2), P2(Next=P3), ... Pn(Next=P1)}
```

Preparing payloads for compression

This modification doesn't cause ambiguity on the receiver, since the total size of the inner payloads can be easily determined after decryption, and while walking through the list of them the receiver always knows whether the current payload is the last or not.

After the use of compression is negotiated in the initial exchange each endpoint is free to decide whether to apply compression or not on per-message basis. However, if applying compression to the content of the Encrypted payload doesn't reduce its size then the compression MUST NOT be used for this message. Implementations MUST be prepared to receive both compressed and uncompressed messages.

4. Payload Formats

4.1. Compressed Payload

The Compressed payload, denoted C!{...} in this document (the exclamation mark means that this payload is critical), contains other payloads in compressed form. The payload type for the Compressed payload is XXX (TBA by IANA).

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| First Payload |   Algorithm   |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
~                      Compressed Payloads                      ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Compressed Payload

o Next Payload (1 octet) - Identifier for the payload type of the next payload in the message.

o Critical (1 bit) - MUST be set to 1.

o RESERVED (7 bits) - MUST be sent as zero; MUST be ignored on receipt (as specified in [RFC7296]).

o Payload Length (2 octets, unsigned integer) - Length in octets of the current payload, including the generic payload header.

o First Payload (1 octet) - Identifier for the payload type of the first payload contained in Compressed Payloads field.

o Algorithm (1 octet) - ID of the algorithm used to compress inner payloads. The possible values for compression algorithm ID are listed in "IKEv2 Notification IPCOMP Transform IDs" registry in [IKEV2-IANA].

- o Compressed Payloads (variable length) - This field contains IKEv2 payloads in compressed form. The Next Payload field of the last included payload MUST be set to 0.

There MUST NOT be more than one Compressed payloads in a message. The Compressed payload MUST NOT appear inside the Encrypted payload and the Encrypted payload MUST NOT appear inside the Compressed payload.

4.2. INVALID_COMPRESSION_ALGORITHM Notification

The INVALID_COMPRESSION_ALGORITHM notification is sent by Responder if the compression algorithm chosen by Initiator is inappropriate. The Notification Data contains the list of supported compression algorithm IDs.

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Next Payload  |C|  RESERVED   |         Payload Length        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |Protocol ID(=0)| SPI Size (=0) |      Notify Message Type      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ~                Supported Compression Algorithms               ~
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

INVALID_COMPRESSION_ALGORITHM Notification

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

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

- o Notify Message Type (2 octets) - MUST be XXX (TBA by IANA), the value assigned for the INVALID_COMPRESSION_ALGORITHM notification.

- o Supported Compression Algorithms (variable length) - List of compression algorithm IDs supported by the Responder. Each algorithm ID occupies one octet. The possible values for compression algorithm IDs are listed in "IKEv2 Notification IPCOMP Transform IDs" registry in [IKEV2-IANA].

5. Interaction with other IKEv2 Extensions

IKEv2 Compression is compatible with most of the IKEv2 extensions, since It neither affects their operation, nor is affected by them. However, some IKEv2 extensions require special handling.

5.1. Interaction with IKEv2 Fragmentation

When compression is used with IKEv2 Fragmentation [RFC7383] the compression MUST take place before splitting the original content of the Encrypted payload into chunks. In other words, the content of the Encrypted payload must be compressed as a whole, before it is fragmented.

The Compressed payload MUST NOT appear inside the Encrypted Fragment payload and the Encrypted Fragment payload MUST NOT appear inside the Compressed payload.

5.2. Interaction with IKEv2 Resumption

The IKEv2 Session Resumption [RFC5723] defines a mechanism for restoring an IKE SA state after a failure. The newly defined IKE_SESSION_RESUME exchange in conjunction with the usual IKE_AUTH exchange is used to create a new IKE SA that is based on the information contained in the resumption ticket.

Implementations supporting compression MUST store the flag whether the compression was negotiated and the negotiated compression algorithm in the resumption ticket and MUST restore these values from the ticket while resuming IKE SA. It means that the use of compression must not be re-negotiated in the IKE_SESSION_RESUME exchange and thus the Compressed payload MUST NOT appear in this exchange.

5.3. Interaction with IKEv2 Redirect

The IKEv2 Redirect mechanism defined in [RFC5685] allows the responder to redirect the initiator to a different host. The redirect can take place either in the IKE_SA_INIT exchange or later, when IKE SA is already created.

All notifications concerning IKEv2 Redirect that may appear in the IKE_SA_INIT exchange, MUST be placed outside the Compressed payload. This would allow the responder to make a decision whether to redirect the initiator without spending additional resources on decompression.

5.4. Interaction with IKEv2 Puzzles

IKEv2 puzzles defined in [RFC8019] allow the Responder to mitigate DoS attacks by requiring the Initiator to spend additional resources for creating IKE SA.

When IKEv2 Compression is used with IKEv2 puzzles, the Puzzle Solution payload MUST NOT be placed inside the Compressed payload.

The Responder MUST NOT use compression until the Initiator solves the puzzle.

6. Security Considerations

It was shown in [COMP-LEAK] that using compression inside an encrypted channel may result in a leakage of some information about a plaintext. Recently some practical exploits were discovered that rely on using compression in security protocols ([CRIME], [BREACH]). However, it is believed that the way a compression is added to the IKEv2 would not weaken the protocol security. The existing exploits rely on ability for an attacker to insert data into an encrypted stream, i.e. to perform a chosen-plaintext attack. IKEv2 messages don’t contain application data, which restricts attacker’s ability to perform chosen-plaintext attack. Moreover, the data usually exchanged over the IKE SA contain no secret information and in most cases no sensitive information. The possible exceptions could be some weak Extensible Authentication Protocol (EAP) methods, which might transfer secret information within an IKE SA. It is RECOMMENDED that implementations don’t use the IKEv2 Compression for the messages containing the EAP payload if there is a possibility that the EAP method transfers secret information.

7. IANA Considerations

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

<TBA>	Compressed	C

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

<TBA>	INVALID_COMPRESSION_ALGORITHM

8. References

8.1. Normative References

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

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

[RFC5685] Devarapalli, V. and K. Weniger, "Redirect Mechanism for the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 5685, DOI 10.17487/RFC5685, November 2009, <https://www.rfc-editor.org/info/rfc5685>.

[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>.

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

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

[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>.

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

8.2. Informative References

[IPSEC-IOT-REQS]
Migault, D. and T. Guggemos, "Requirements for Diet-ESP the IPsec/ESP protocol for IoT", draft-mglt-6lo-diet-esp-requirements-01 (work in progress), February 2015.

[COMP-LEAK]
Kelsey, J., "Compression and Information Leakage of Plaintext", <http://www.iacr.org/cryptodb/archive/2002/FSE/3091/3091.pdf>.

[CRIME] Rizzo, J. and T. Duong, "The CRIME attack", <https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU>.

[BREACH] Prado, A., Harris, N., and Y. Gluck, "SSL, gone in 30 seconds: A BREACH beyond CRIME", <https://media.blackhat.com/us-13/US-13-Prado-SSL-Gone-in-30-seconds-A-BREACH-beyond-CRIME-Slides.pdf>.

Author’s Address

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

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