IPSecME Working Group   Y. Nir
Internet-Draft   Check Point
Intended status: Standards Track   V. Smyslov
Expires: October 17, 2016   ELVIS-PLUS
                                                          April 15, 2016

Protecting Internet Key Exchange Protocol version 2 (IKEv2) Implementations from Distributed Denial of Service Attacks
draft-ietf-ipsecme-ddos-protection-06

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 effectively defend itself against DoS attacks from spoofed IP-addresses. However, bot-nets 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 thwart (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", "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, and sending multiple such requests can either cause the Responder to allocate too much resources and fail, or else if resource allocation is somehow throttled, legitimate Initiators would also be prevented 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.

Yes, there's a stage 4 where the Responder actually creates Child SAs, but when talking about (D)DoS, we never get to this stage.

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's much heavier CPU-wise. Stage #3 includes public key operations, typically more than one.

To attack such a Responder, an attacker can attempt to either 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 should be easy because those requests are cheap.

There are obvious ways for the Responder to protect itself even 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, do the same thing only more of it. They make it harder for an attacker to

reach the goal of getting a half-open SA. They don't have to be so hard that an attacker can't afford to solve a single puzzle; it's enough that they increase the cost of a half-open SAs for the attacker so that it can create only a few.

Reducing the amount of time an abandoned half-open SA is kept attacks the issue from the other side. It reduces the value the attacker gets from managing to create a half-open SA. For example, if a half-open SA is kept for 1 minute and the capacity is 60,000 half-open SAs, an attacker would need to create 1,000 half-open SAs per second. Reduce the retention time to 3 seconds, and the attacker needs to create 20,000 half-open SAs per second. By introducing a puzzle, each half-open SA becomes more expensive for an attacker, making it more likely to thwart 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 ways 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.

It seems that the first thing cannot be dealt with at the IKE level. It's probably better left to Intrusion Prevention System (IPS) technology.

On the other hand, sending an IKE_AUTH request is surprisingly cheap. It requires a proper IKE header with the correct IKE SPIs, and it requires a single Encrypted payload. The content of the payload might as well 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 does not check. Depending on the Responder implementation, this can be repeated with the same half-open SA. Puzzles can 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 cost of half-open SA up to what is tolerable for legitimate clients.

Puzzles have their place as part of #4.

4. Defense Measures while 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". Retransmission policies in practice wait at least 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 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 makes sense to use a 64-bit prefix as the basis for rate limiting in IPv6.

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 require solving 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 downside 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, there is a huge advantage in blocking the DoS attack using rate-limiting for legitimate clients that are away from the attacking nodes. In such cases, adverse impacts caused by the attack or by the measures used to counteract the attack can be avoided.

4.3. The Stateless Cookie

Section 2.6 of [RFC7296] offers a mechanism to mitigate DoS attack: 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 bot-nets, 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 from [RFC7296]. It is loosely based on the proof-of-work technique used in Bitcoins [bitcoins]. This sets an upper bound, determined by the attacker's CPU, to the number of negotiations it can initiate in a unit of time.

A puzzle is sent to the Initiator in two cases:

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

o 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 notification in response to a IKE_SA_INIT request, the notification includes three fields:

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 for it to make any difference in mitigating DDoS attacks. 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 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 part is indeed valid.

o That the PRFs 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: Three solutions for 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 MAY choose to prefer previously authenticated peers who present a Session Resumption ticket (see [RFC5723] for details). The Responder MAY require such Initiators to pass a return routability check by including the COOKIE notification in the IKE_SESSION_RESUME response message, as allowed by 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 a malicious peer to perform an attack, described in Section 3

- it can just send a garbage in the IKE_AUTH message thus forcing the Responder to perform CPU costly 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 the malicious Initiator 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 Attacks using "Hash and URL" Certificate Encoding

In IKEv2 each side may use "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 responder by providing an URL pointing to a large file possibly containing garbage. While downloading the file the responder consumes CPU, memory and network bandwidth.

To prevent this kind of attacks 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 by analyzing the first few bytes, if they are DER-encoded. However, since the content of the file being downloaded can be under attacker's control, implementations should not blindly trust the decoded length and SHOULD check whether it makes sense before continue downloading. 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.

5. Defense Measures after IKE SA is created

Once IKE SA is created there is usually not much traffic over it. In most cases this traffic consists of exchanges aimed to create additional Child SAs, rekey, or delete them and check the liveness of the peer. With a typical setup and typical Child SA lifetimes, there

are typically no more than a few such exchanges, often less. 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 overwhelming number of Child SAs with the same Traffic Selectors etc. Such behavior may be caused by buggy implementation, misconfiguration or be intentional. The latter becomes more of a real threat if the peer uses NULL Authentication, described in [RFC7619]. In this case the peer remains anonymous, allowing it to escape any responsibility for its actions. See Section 3 of [RFC7619] for details.

The following recommendations for defense against possible DoS attacks after IKE SA is established are mostly intended for implementations that allow unauthenticated IKE sessions; however, they may also be useful in other cases.

o If the IKEv2 window size is greater than one, then the peer could initiate multiple simultaneous exchanges that could increase host resource consumption. Since currently there is no way in IKEv2 to decrease window size once it was increased (see Section 2.3 of [RFC7296]), the window size cannot be dynamically adjusted depending on the load. For that reason, it is NOT RECOMMENDED to ever increase the IKEv2 window size above its default value of one if the peer uses NULL Authentication.

o If the peer initiates requests to rekey IKE SA or Child SA too often, implementations can respond to some of these requests with the TEMPORARY_FAILURE notification, indicating that the request should be retried after some period of time.

o If the peer creates too many Child SA with the same or overlapping Traffic Selectors, implementations can respond with the NO_ADDITIONAL_SAS notification.

o If the peer initiates too many exchanges of any kind, implementations can introduce an artificial delay before responding to each request message. This delay would decrease the rate the implementation need to process requests from any particular peer, making it possible to process requests from the others. Note, that if the Responder receives retransmissions of the request message during the delay period, the retransmitted messages should be silently discarded. The delay should not be too long to avoid causing the IKE SA to be deleted on the other

end due to timeout. It is believed that a few seconds is enough. Note however, that even a few seconds may be too long in those settings, that rely on immediate response to the request message, e.g. for the purposes of quick detection of a dead peer.

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

In IKEv2 client can request various configuration attributes from server. Most often those attributes include internal IP addresses. Malicious clients can try to exhaust 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, that it 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 defeating the DDoS attack.

Implementations may be deployed in different environments, so it is RECOMMENDED that the parameters be settable. As an 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 work on 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, searching for two things:

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 at 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 the the tunnels are established using EAP (see Section 2.16 of [RFC7296]), users enter the 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 failure more than 300 times per minute, that 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 that IP address or prefix, or an inordinate amount of IKE_AUTH failures. A DDoS attack may be viewed as multiple such attacks. If they are mitigated well enough, there will not be a need enact countermeasures on all Initiators. Typical 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 and in this case the number of failed IKE_AUTH exchanges never exceeds the threshold of attack detection. This is a good thing as it prevents Initiators that are not close to the attackers from being affected.

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 the monitoring of the number of half-open SAs, and setting a soft limit per peer IP or prefix. The soft limit can be set to 3-5, and 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 force the attacker to use real source addresses, and help avoid the need to impose a greater burden in the form of cookies 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 increase the difficulty of puzzles imposed on IKE_SA_INIT requests coming from suspicious nodes/prefixes. It should still be doable by all legitimate peers, but it can degrade 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 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 should monitor incoming requests to detect whether it is under attack. If the Responder learns that (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 so high, 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 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 Puzzle

If the Responder makes a decision to use puzzles, then it MUST include two notifications in its response message - the COOKIE notification and the PUZZLE notification. 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 get better chances to be served.

7.1.1.1. Selecting 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 next to 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 the order of magnitude, or even more. The Responder may set difficulty level to 0, meaning that the Initiator is requested to spend as much power to solve 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 chances it will have to be served by the Responder. In diverse environments it is RECOMMENDED that the Initiator sets difficulty level to 0, unless the attack volume is very high.

If the Responder sets 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 Puzzle Algorithm

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

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

To support this feature in case of puzzles, the algorithm that is used to compute puzzle needs to be negotiated during IKE_SA_INIT exchange. The negotiation is performed as follows. The initial request message sent by Initiator contains SA payload with the list of transforms the Initiator supports and is willing to use in the IKE SA being established. The Responder parses received SA payload and finds mutually supported set of transforms of type PRF. It selects most preferred transform from this set 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 the use in core IKE SA crypto operations. If there are no mutually supported PRFs, then negotiation will fail anyway and there is no reason to return a puzzle. In this case the Responder returns 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 must always be present in SA payload in IKE_SA_INIT request message.

7.1.1.3. Generating Cookie

If Responder supports puzzles then 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:

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

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

- o The number of consecutive puzzles given to the Initiator.

- o 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 a local matter of Responder, as the cookie would remain an opaque blob 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 returned cookie is detectable. So, the cookie would be generated as:

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

Alternatively, the Responder may continue to generate cookie as suggested in Section 2.6 of [RFC7296], but associate the additional information, that would be stored locally, 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 the cookie was generated.

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

7.1.2. Solving 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 also 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 when 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 do it. In this case the Initiator MUST ignore the received message and continue to wait until either the valid one is received or the retransmission timer fires. If it fails to receive the valid message after several retransmissions of IKE_SA_INIT request, then it means that something is wrong and the IKE SA cannot be established.

If the Initiator supports puzzles and is ready to deal with them, then it tries to solve the given puzzle. After the puzzle is solved the Initiator restarts the request and returns 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 back to the Responder.

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

7.1.3. Computing Puzzle

General principals 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 solution to the puzzle 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 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 Notification 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 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 of them, 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, then it 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 for any reason the message contains it. 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, then it 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 the four returned keys.

If any checked result contains fewer bits than were requested, it 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:

o 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 got.

o The higher number of zero bits the Initiator got, the higher priority its request should receive.

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

o 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. Making Decision whether to Serve the Request

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

- Accept request.

- Reject request.

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

The Responder SHOULD accept an incoming request if its priority is high - it means that the Initiator spent quite a lot of resources. The Responder MAY also accept some of 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 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 rest of requests can be either discarded or responded to with new puzzles.

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 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 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 fact allows malicious Initiator to perform an attack, described in Section 3. Unlike DoS attack in 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 garbage. This costs nothing to the Initiator, but the Responder has to do relatively costly operations of 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 D-H shared secret. The difficulty level of the puzzle should be selected so that the Initiator would spend substantially more time to solve the puzzle than the Responder to compute the shared secret.

The Responder should constantly monitor 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 the 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 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 the legitimate clients would experience an additional delay while establishing 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 estimate the computational power of the Initiator and to select the difficulty level accordingly. Unlike puzzles in IKE_SA_INIT, the requested difficulty level for IKE_AUTH puzzles MUST NOT be zero. In other words, the Responder must always set specific difficulty level and must not let the Initiator to choose it on its own.

7.2.1.2. Selecting 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 response message 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 payload MUST be placed outside the Encrypted payload, so that the Responder would be 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 Puzzle

The puzzles in the IKE_AUTH exchange are 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 IKE_SA_INIT exchange, stripped of any headers. SPIr is IKE Responder's SPI from the IKE header of the SA being established.

7.2.4. Receiving 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 the four 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 is possible, that due to packet loss and/or reordering the Responder could receive non-first IKE Fragment messages before receiving the first one, containing 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. However, 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 could 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 necessity 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 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 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                      ~
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

o 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 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 PRF agreed upon 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 difficulty level should be set by balancing the requirement to minimize the latency for legitimate Initiators and making things difficult for attackers. A good rule of thumb is for taking about 1 second to solve the puzzle. A typical Initiator or bot-net member in 2014 can perform slightly less than a million hashes per second per core, so setting the difficulty level to n=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.

10. Security Considerations

When selecting parameters for the puzzles, in particular the puzzle difficulty, care must be taken. If the puzzles appeared too easy for majority of the attackers, then the puzzles mechanism wouldn't be able to prevent DoS attack and would only impose an additional burden on the legitimate Initiators. On the other hand, if the puzzles appeared to be too hard for majority of the Initiators then many legitimate users would experience unacceptable delay in IKE SA setup (or unacceptable power consumption on mobile devices), that might cause them to cancel 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 remain 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.

Solving puzzles requires a lot of CPU power, that would increase power consumption. This would influence battery-powered Initiators, e.g. mobile phones or some IoT devices. If puzzles are hard then the required additional power consumption may appear to be unacceptable for some Initiators. The Responder SHOULD take this possibility into considerations while choosing the puzzles 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, that may be used by attackers to perform DoS attack after IKE SA is established. Responders that allow unauthenticated Initiators to connect must be prepared to deal with various kinds of DoS attacks even after 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 contribution into design of the protocol. In particular, Tero Kivinen suggested the kind of puzzle where the task is to find a solution with 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 for his carefull review of the document, Graham

Bartlett for pointing out to the possibility of "Hash & URL" related attack, 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>.

[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>.

[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>.

[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