Network Working Group
Internet-Draft
Expires: December 18, 2003

R. Moskowitz
ICSAlabs, a Division of TruSecure Corporation
P. Nikander
P. Jokela
Ericsson Research Nomadic Lab

June 19, 2003

Host Identity Protocol
draft-moskowitz-hip-07

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on December 18, 2003.

Copyright Notice

Abstract

This memo specifies the details of the Host Identity Protocol (HIP). The overall description of protocol and the underlying architectural thinking is available in the separate HIP architecture specification. The Host Identity Protocol is used to establish a rapid authentication between two hosts and to provide continuity of communications between those hosts independent of the networking layer.

The various forms of the Host Identity (HI), Host Identity Tag (HIT),

and Local Scope Identifier (LSI), are covered in detail. It is described how they are used to support authentication and the establishment of keying material, which is then used by IPsec Encapsulated Security payload (ESP) to establish a two-way secured communication channel between the hosts. The basic state machine for HIP provides a HIP compliant host with the resiliency to avoid many denial-of-service (DoS) attacks. The basic HIP exchange for two public hosts shows the actual packet flow. Other HIP exchanges, including those that work across NATs are covered elsewhere.

Table of Contents

1. Introduction

The Host Identity Protocol (HIP) provides a rapid exchange of Host Identities (HI) between two hosts. The exchange also establishes a pair IPsec Security Associations (SA), to be used with IPsec Encapsulated Security Payload (ESP) [5]. The HIP protocol is designed to be resistant to Denial-of-Service (DoS) and Man-in-the-middle (MitM) attacks, and when used to enable ESP, provides DoS and MitM protection to upper layer protocols, such as TCP and UDP.

1.1 A new name space and identifiers

The Host Identity Protocol introduces a new namespace, the Host Identity. The affects of this change are explained in the companion document, the HIP architecture [18] specification.

There are three representations of the Host Identity, the full Host Identifier (HI), the Host Identity Tag (HIT), and the Local Scope Identity (LSI). Three representations are used, as each meets a different design goal of HIP, and none of them can be removed and meet these goals. The HI represents directly the Identity, a public key. Since there are different public key algorithms that can be used with different key lengths, the HI is not good for using as the HIP packet identifier, or as a index into the various operational tables needed to support HIP.

A hash of the HI, the Host Identity Tag (HIT), thus becomes the

operational representation. It is 128 bits long. It is used in the HIP payloads, and it is intended be used to index the corresponding state in the end hosts.

In many environments, 128 bits is still considered large. For example, currently used IPv4 based applications are constrained with 32 bit API fields. Thus, the third representation, the 32 bit LSI, is needed. The LSI provides a compression of the HIT with only a local scope so that it can be carried efficiently in any application level packet and used in API calls.

1.2 The HIP protocol

The base HIP exchange consists of four packets. The four-packet design helps to make HIP DoS resilient. The protocol exchanges Diffie-Hellman keys in the 2nd and 3rd packets, and authenticates the parties in the 3rd and 4th packets. Additionally, it starts the cookie exchange in the 2nd packet, completing it with the 3rd packet.

The exchange uses the Diffie-Hellman exchange to hide the Host

Identity of the Initiator in packet 3. The Responder's Host Identity is not protected. It should be noted, however, that both the Initiator and the Responder HITs are transported as such (in cleartext) in the packets, allowing an eavesdropper with a priori knowledge about the parties to verify their identies.

Data packets start after the 4th packet. The 3rd and 4th HIP packets may carry a data payload in the future. However, the details of this are to be defined later as more implementation experience is gained.

Finally, HIP is designed as an end-to-end authentication and key establishment protocol. It lacks much of the fine-grain policy control found in IKE that allows IKE to support complex gateway policies. Thus, HIP is not a complete replacement for IKE.

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 [2].

3. Host Identifiers

The structure of the Host Identifier is the public key of an asymmetric key pair. Correspondingly, the host itself is entity that holds the private key from the key pair. See the HIP architecture specification [18] for more details about the difference between an identity and the corresponding identifier.

DSA is the MUST implement algorithm for all HIP implementations, other algorithms MAY be supported. DSA was chosen as the default algorithm due to its small signature size.

A Host Identity Tag (HIT) is used in protocols to represent the Host Identity. Another representation of the Host Identity, the Local Scope Identity (LSI), can also be used in protocols and APIs. LSI's advantage over HIT is its size; its disadvantage is its local scope.

3.1 Host Identity Tag (HIT)

The Host Identity Tag is a 128 bit entity. There are two advantages of using a hash over the actual Identity in protocols. Firstly, its fix length makes for easier protocol coding and also better manages the packet size cost of this technology. Secondly, it presents a consistent format to the protocol whatever underlying identity technology is used.

There are two types of HITs. HITs of the first type, called *type 1 HIT*, consist of an initial 2 bit prefix of 01, followed by 126 bits of the SHA-1 hash of the public key. HITs of the second type consist of a Host Assigning Authority (HAA) field, and only the last 64 bits come from a SHA-1 hash of the Host Identity. This latter format for HIT is recommended for 'well known' systems. It is possible to support a resolution mechanism for these names in hierarchical directories, like the DNS. Another use of HAA is in policy controls, see Section 12.

This document fully specifies only type 1 HITs. HITs that consists of the HAA field and the hash are specified in [19].

Any conforming implementation MUST be able to deal with HITs that are not type 1 ones. However, in that case the implementation must explicitly learn and record the binding between the Host Identifier and the HIT, and it may not be able form such HITs from Host Identifiers.

3.1.1 Generating a HIT from a HI

The 126 or 64 hash bits in a HIT MUST be generated by taking the

least significant 126 or 64 bits of the SHA-1 [17] hash of the Host Identifier as it is represented in the Host Identity field in a HIP payload packet.

For Identities that are DSA public keys, the HIT is formed as follows.

1. The DSA public key is encoded as defined in RFC2536 [12] Section 2, taking the fields T, Q, P, G, and Y, concatenated. Thus, the length of the data to be hashed is 1 + 20 + 3 * 64 + 3 * 8 * T octets long, where T is the size parameter as defined in RFC2536 [12]. The size parameter T, affecting the field lengths, MUST be selected as the minimum value that is long enough to accomodate P, G, and Y. The fields MUST be encoded in network byte order, as defined in RFC2536 [12].

2. A SHA-1 hash [17] is calculated over the encoded key.

3. The least signification 126 or 64 bits of the hash result are used to create the HIT, as defined above.

The following pseudo-code illustrates the process. The symbol := denotes assignment; the symbol += denotes appending. The pseudo-function encode_in_network_byte_order takes two parameters, an integer (bignum) and length, and returns the integer encoded into a byte string of the given length.

```
buffer := encode_in_network_byte_order ( DSA.T , 1 )
buffer += encode_in_network_byte_order ( DSA.Q , 20 )
buffer += encode_in_network_byte_order ( DSA.P , 64 + 8 * T )
buffer += encode_in_network_byte_order ( DSA.G , 64 + 8 * T )
buffer += encode_in_network_byte_order ( DSA.Y , 64 + 8 * T )

digest := SHA-1 ( buffer )

hit_126 := concatenate ( 01 ,      low_order_bits ( digest, 126 ) )
hit_haa := concatenate ( 10 , HAA, low_order_bits ( digest,  64 ) )
```

3.2 Local Scope Identity (LSI)

LSIs are 32-bit localized representations of a Host Identity. The purpose of an LSI is to facilitate using Host Identities in existing IPv4 based protocols and APIs. The owner of the Host Identity does not set its own LSI; each host selects its partner's 32 bit representation for a Host Identity.

A *local LSI* is an LSI that a remote host has assigned to a host.

In some implementations, local LSIs may be assigned to some interface as an IP address. A *remote LSI* is an LSI that the host has assigned to represent some remote host (and that the remote host has accepted).

The LSIs MUST be allocated from the 1.0.0.0/8 subnet. That makes it easier to differentiate between LSIs and IPv4 addresses at the API level. By default, the low order 24 bits SHOULD be equal with the low order 24 bits of the corresponding HIT. That allows easier mapping between LSIs and HITs, and makes the LSI assigned to a host to be a fixed one.

It is possible that the HITs of two remote hosts have equal low order 24 bits. Since HITs are basically random, if a host is communicating with 1000 other hosts, the risk of such collision is roughly 0.006%, and for a host communicating with 10000 other hosts, the risk is about 0.06%. However, given a population of 100000 hosts, each communicating with 1000 other hosts, the probability that there was no collisions at all is only about 2%. In other words, even though collisions are fairly rare events for any given host, they will happen, and the hosts MUST be able to cope with them.

If a host is forming a remote LSI for a HIT whose low order 24 bits are equal with another already existing remote LSI, the host MUST select another LSI to represent that host. It may also be hard for a host to use a remote LSI that is equal to its own local LSI. Thus, if the low order 24 bits of a remote HIT are equal to the low order 24 bits of a local LSI, the host MAY select a different LSI to represent the remote host. In either case, the host SHOULD assing the low order 24 bits of the LSI randomly. All hosts MUST be prepared to handle local LSIs whose low order 24 bits do not match with any of their own HITs.

If the LSI assigned by a peer to represent a host is unacceptable, the host MAY terminate the HIP four-way handshake and start anew.

3.3 Security Parameter Index (SPI)

SPIs are used in ESP to find the right security association for received packets. The ESP SPIs have added significance when used with HIP; they are a compressed representation of the HIT in every packet. Thus they MAY be used by intermediary systems in providing services like address mapping. Note that since the SPI has

significance at the receiver, only the < DST, SPI >, where DST is a destination IP address, uniquely identifies the receiver HIT at every given point of time. The same SPI value may be used by several hosts. The same < DST, SPI > may denote different hosts at different points of time, depending on which host is currently reachable at the

DST.

Each host selects for itself the SPI it wants to see in packets received from its peer. This allows it to select different SPIs for different peers. The SPI selection SHOULD be random. A different SPI SHOULD be used for each HIP exchange with a particular host; this is to avoid a replay attack. Additionally, when a host rekeys, the SPI MUST change. Furthermore, if a host changes over to use a different IP address, it MAY change the SPI used.

One method for SPI creation that meets these criteria, would be to concatenate the HIT with a 32 bit random or sequential number, hash this (using SHA1), and then use the high order 32 bits as the SPI.

The selected SPI is communicated to the peer in the third (I2) and fourth (R2) packets of the base HIP exchange. Changes in SPI are signalled with NES packets.

3.4 Difference between an LSI and the SPI

There is a subtle difference between an LSI and a SPI.

The LSI is relatively longed lived. A system selects the LSI it locally uses to represent its peer, it SHOULD reuse a previous LSI for a HIT during a HIP exchange. This COULD be important in a timeout recovery situation. The LSI ONLY appears in the 3rd and 4th HIP packets (each system providing the other with its LSI). The LSI is used anywhere in system processes where IP addresses have traditionally have been used, like in TCBs and FTP port commands.

The SPI is short-lived. It changes with each HIP exchange and with a HIP rekey and/or movement. A system notifies its peer of the SPI to use in ESP packets sent to it. Since the SPI is in all but the first two HIP packets, it can be used in intermediary systems to assist in address remapping.

3.5 TCP and UDP pseudoheader computation

When computing TCP and UDP checksums on sockets bound to HITs or LSIs, the IPv6 pseudo-header format [10] is used. Additionally, the HITs MUST be used in the place of the IPv6 addresses in the IPv6 pseudoheader. Note that the pseudo-header for actual HIP payloads is computed differently; see Section 6.1.2.

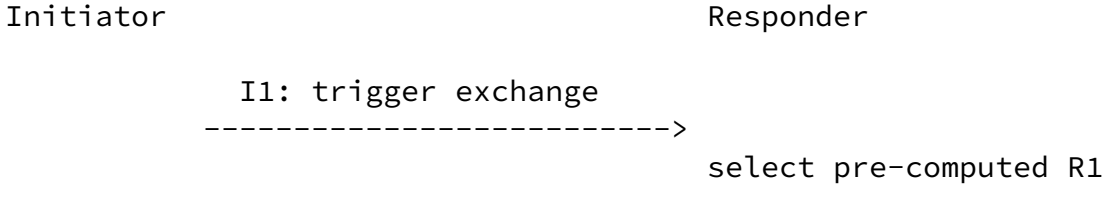
4. The Host Identity Protocol

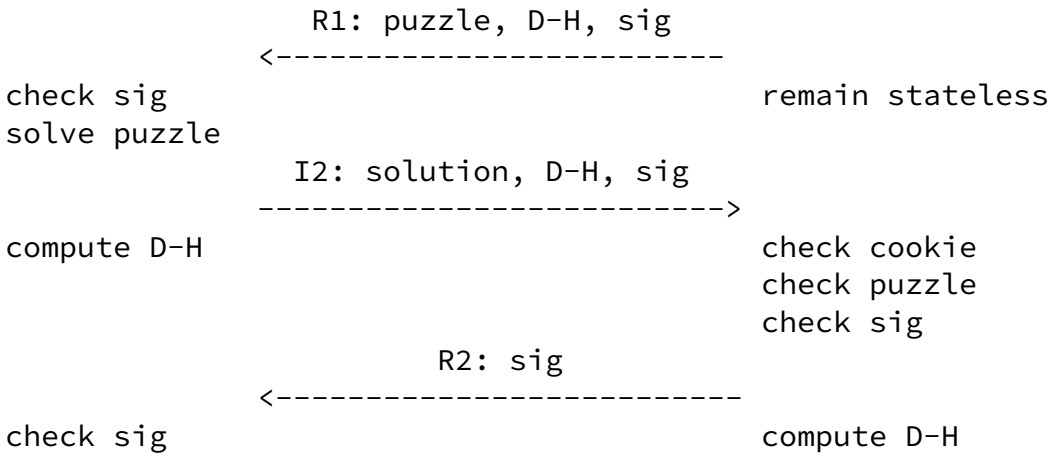
The Host Identity Protocol is IP protocol TBD. The HIP payload could be carried in every datagram. However, since HIP datagrams are relatively large (at least 40 bytes), and ESP already has all of the functionality to maintain and protect state, the HIP payload is 'compressed' into an ESP payload after the HIP exchange. Thus in practice, HIP packets only occur in datagrams to establish or change HIP state.

4.1 Base HIP exchange

The base HIP exchange serves to manage the establishment of state between an Initiator and a Responder. The Initiator first sends a trigger packet, I1, to the responder. The second packet, R1, starts the actual exchange. In contains a puzzle, a cryptographic challenge that the Initiator must solve before continuing the exchange. In its reply, I2, the Initiator must display the solution. Without a solution the I2 message is simply discarded.

The last three packets of the exchange, R1, I2, and R2, constitute a standard authenticated Diffie-Hellman key exchange. The base exchange is illustrated below.





4.1.1 HIP Cookie Mechanism

The purpose of the HIP cookie mechanism is to protect the Responder from a number of denial-of-service threats. It allows the Responder

to delay state creation until receiving I2. Furthermore, the puzzle included in the cookie allows the Responder to use a fairly cheap calculation to check that the Initiator is "sincere" in the sense that it has churned CPU cycles in solving the puzzle.

The Cookie mechanism has been explicitly designed to give space for various implementation options. It allows a responder implementation to completely delay session specific state creation until a valid I2 is received. In such a case a validly formatted I2 can be rejected earliest only once the responder has checked its validity by computing one hash function. On the other hand, the design also allows a responder implementation to keep state about received I1s, and match the received I2s against the state, thereby allowing the implementation to avoid the computational cost of the hash function. The drawback of this latter approach is the requirement of creating state. Finally, it also allows an implementation to use any combination of the space-saving and computation-saving mechanism.

One possible way how a Responder can remain stateless but drop most spoofed I2s is to base the selection of the cookie on some function over the Initiator's identity. The idea is that the Responder has a (perhaps varying) number of pre-calculated R1 packets, and it selects

one of these based on the information carried in I1. When the Responder then later receives I2, it checks that the cookie in the I2 matches with the cookie send in the R1, thereby making it impractical for the attacker to first exchange one I1/R1, and then generate a large number of spoofed I2s that seemingly come from different IP addresses or use different HITs. The method does not protect from an attacker that uses fixed IP addresses and HITs, though. Against such an attacker it is probably best to create a piece of local state, and remember that the puzzle check has previously failed. See Appendix D for one possible implementation. Note, however, that the implementations MUST NOT use the exact implementation given in the appendix, and SHOULD include sufficient randomness to the algorithm so that algorithm complexity attacks become impossible [21].

The Responder can set the difficulty for Initiator, based on its concern of trust of the Initiator. The Responder SHOULD use heuristics to determine when it is under a denial-of-service attack, and set the difficulty value K appropriately.

The Responder starts the cookie exchange when it receives an I1. The Responder supplies a random number I, and requires the Initiator to find a number J. To select a proper J, the Initator must create the concatenation of I, the HITs of the parties, and J, and take a SHA-1 hash over this concatenation. The lowest order K bits of the result MUST be zeros. To accomplish this, the Initiator will have to generate a number of Js until one produces the hash target. The

Initiator SHOULD give up after trying 2^(K+2) times, and start over the exchange. (See Appendix C.) The Responder needs to re-create the contactenation of I, the HITs, and the provided J, and compute the hash once to prove that the Initiator did its assigned task.

To prevent pre-computation attacks, the Responder MUST select I in such a way that the Inititiator cannot guess it. Furthermore, the construction MUST allow the Responder to verify that the value were indeed selected by it and not by the Initiator. See Appendix D for an example on how to implement this.

It is RECOMMENDED that the Responder generates a new cookie and a new R1 once every few minutes. Furthermore, it is RECOMMENDED that the responder remembers an old cookie at least 60 seconds after it has been deprecated. These time values allow a slower Initiator to solve

the cookie puzzle while limiting the usability that an old, solved cookie has to an attacker.

In R1, the values I and K are sent in network byte order. Similarily, in I2 the values I and J are sent in network byte order. The SHA-1 hash is created by concatenating, in network byte order, the following data, in the following order:

64-bit random value I, in network byte order, as appearing in R1 and I2.

128-bit Initiator HIT, in network byte order, as appearing in the HIP Payload in R1 and I2.

128-bit Responder HIT, in network byte order, as appearing in the HIP Payload in R1 and I2.

64-bit random value J, in network byte order, as appearing in I2.

In order to be a valid response cookie, the K low-order bits of the resulting SHA-1 digest must be zero.

Notes:

The length of the data to be hashed is 48 bytes.

All the data in the hash input MUST be in network byte order.

The order of the Initiator and Responder HITs are different in the R1 and I2 packets, See Section 6.1. Care must be taken to copy the values in right order to the hash input.

Precomputation by the Responder Sets up the challenge difficulty K.

Generates a random number I.
Creates a signed R1 and caches it.

Responder Sends I and K in a HIP Cookie in an R1.

Saves I and K for a Delta time.

Initiator Generates repeated attempts to solve the challenge until a matching J is found:

```
Ltrunc( SHA-1( I | HIT-I | HIT-R | J ), K ) == 0
Send I and J in HIP Cookie in I2.
```

Responder Verify that the received I is a saved one.

```
Match the Response with a K based on I.
Compute V := Ltrunc( SHA-1( I | HIT-I | HIT-R | J ), K )
Reject if V != 0
Accept if V == 0
```

4.1.2 Authenticated Diffie-Hellman protocol

4.1.3 HIP Birthday

The Birthday is a reboot count used to manage state reestablishment when one peer rebooted or timed out its SA. The Birthday is increased every time the system boots. The Birthday also has to be increased in accordance with the system's SA timeout parameter. If the system has open SAs, it MUST increase its Birthday. This impacts a system's approach to precomputing R1 packets.

Birthday SHOULD be a counter. It cannot be reset by the user and a system is unlikely to need a birthday larger than 2^64. Date-time in GMT can be used if a cross-boot counter is not possible, but it has a potential problem if the system time is set back by the user.

4.2 Sending data on HIP packets

A future version of this document may define how to send ESP protected data on various HIP packets. However, currently the HIP header is a terminal header, and not followed by any other headers.

4.3 Distributing certificates

Certificates MAY be distributed using the CERT packet. [XXX: This

section needs more text.].

5. The Host Identity Protocol packet flow and state machine

A typical HIP packet flow is shown below.

```
I --> Directory: lookup of R
I <-- Directory: return R's addresses, HI, and HIT
I1      I --> R (Hi. Here is my I1, let's talk HIP)
R1      I <-- R (OK.  Here is my R1, handle this HIP cookie)
I2      I --> R (Compute, compute, here is my counter I2)
R2      I <-- R (OK.  Let's finish HIP with my R2)
I --> R (ESP protected data)
I <-- R (ESP protected data)
```

5.1 HIP Scenarios

The HIP protocol and state machine is designed to recover from one of the parties crashing and losing its state. The following scenarios describe the main use cases covered by the design.

No prior state between the two systems.

The system with data to send is the Initiator. The process follows standard 4 packet exchange, establishing the SAs.

The system with data to send has no state with receiver, but receiver has a residual SA.

Intiator acts as in no prior state, sending I1 and getting R1. When Receiver gets I2, the old SA is 'discovered' and deleted; the new SAs are established.

System with data to send has an SA, but receiver does not.

Receiver 'detects' when it receives an unknown SPI. Receiver sends an R1 with a NULL Initiator HIT. Sender gets the R1 with a later birthdate, discards old SA and continues exchange to establish new SAs for sending data.

A peer determines that it needs to reset Sequence number or rekey.

It sends NES. Receiver sends NES response, establishes new SAs for peers.

5.2 Refusing a HIP exchange

A HIP aware host may choose not to accept a HIP exchange. If the

host's policy is to only be an initiator, it should begin its own HIP exchange. A host MAY choose to have such a policy since only the Initiator HI is protected in the exchange. There is a risk of a race condition if each host's policy is to only be an initiator, at which point the HIP exchange will fail.

If the host's policy does not permit it to enter into a HIP exchange with the Initiator, it should send an ICMP Protocol Unreachable, Administratively Prohibited message. A more complex HIP packet is not used here as it actually opens up more potential DoS attacks than a simple ICMP message.

5.3 Reboot and SA timeout restart of HIP

Simulating a loss of state is a potential DoS attack. The following process has been crafted to manage state recovery without presenting a DoS opportunity.

If a host reboots or times out, it has lost its HIP state. If the system that lost state has a datagram to deliver to its peer, it simply restarts the HIP exchange. The peer sends an R1 HIP packet, but does not reset its state until it receives the I2 HIP packet. The I2 packet MUST have a Birthday greater than the current SA's Birthday. This is to handle DoS attacks that simulate a reboot of a peer. Note that either the original Initiator or the Responder could end up restarting the exchange, becoming the new Initiator. An example of the initial Responder needing to send a datagram but not having state occurs when the SAs timed out and a server on the Responder sends a keep-alive to the Initiator.

If a system receives an ESP packet for an unknown SPI, the assumption is that it has lost the state and its peer did not. In this case, the system treats the ESP packet like an I1 packet and sends an R1 packet. The Initiator HIT is typically NULL in the R1, since the system usually does not know the peer's HIT any more.

The system receiving the R1 packet first checks to see if it has an established and recently used SA with the party sending the R1. If such an SA exists, the system checks the Birthday, if the Birthday is

greater than the current SA's Birthday, it processes the R1 packet and resends the ESP packet (along with or) after the I2 packet. The peer system processes the I2 in the normal manner, and replies with an R2. This will reestablish state between the two peers. [XXX: Potential DoS attack if hundreds of peers 'loose' their state and all send R1 packets at once to a server. However, that would require the attacker having specific knowledge about the SAs used, and an ability to trigger R1s as the SAs are used.]

5.4 HIP State Machine

HIP has very little state. In the base HIP exchange, there is an Initiator and a Responder. Once the SAs are established, this distinction is lost. If the HIP state needs to be re-established, the controlling parameters are which peer still has state and which has a datagram to send to its peer. The following state machine attempts to capture these processes.

The state machine is presented in a single system view, reresenting either an Initiator or a Responder. There is not a complete overlap of processing logic here and in the packet definitions. Both are needed to completely implement HIP.

5.4.1 HIP States

E0 State machine start

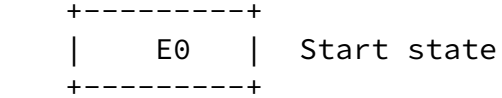
E1 Initiating HIP

E2 Waiting to finish HIP

E3 HIP SA established

E-FAILED HIP SA establishment failed

5.4.2 HIP State Processes



```
   Datagram to send, send I1 and go to E1
   Receive I1, send R1 and stay at E0
   Receive I2, process
        if successful, send R2 and go to E3
        if fail, stay at E0
   Receive ESP for unknown SA, send R1 and stay at E0
   Receive ANYOTHER, drop and stay at E0

   +---------+
   |    E1   |  Initiating HIP
   +---------+

   Receive I1, send R1 and stay at E1
   Receive I2, process
        if successful, send R2 and go to E3
```

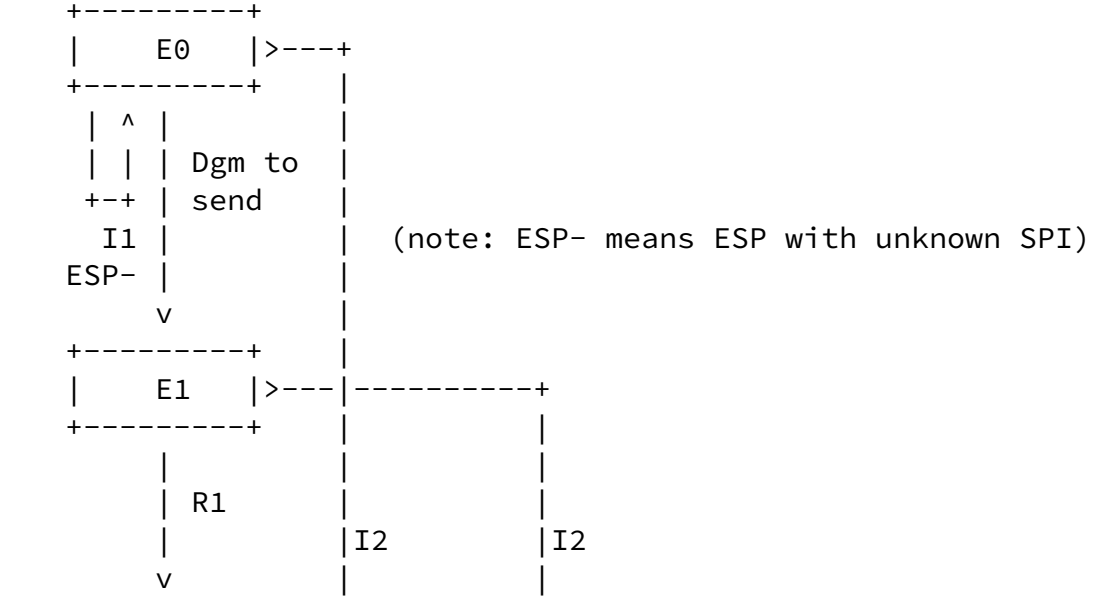


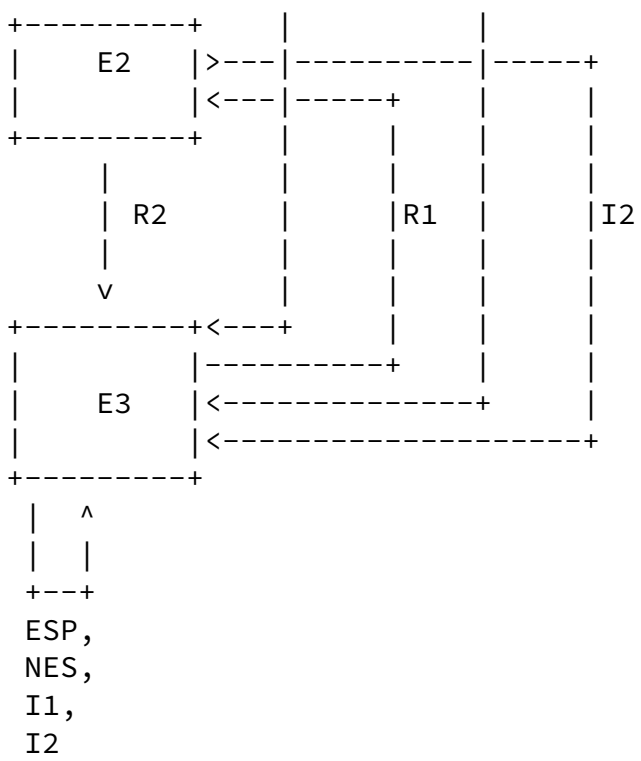
```
        if fail, stay at E1
   Receive R1, process
        if successful, send I2 and go to E2
        if fail, go to E-FAILED
   Receive ANYOTHER, drop and stay at E1
   Timeout, increment timeout counter
        If counter is less than N1, send I1 and stay at E1
        If counter is greater than N1, go to E-FAILED

   +---------+
   |    E2   | Waiting to finish HIP
   +---------+

   Receive I1, send R1 and stay at E2
   Receive I2, process
        if successful, send R2 and go to E3
        if fail, stay at E2
   Receive R2, process
        if successful, go to E3
        if fail, go to E-FAILED
   Receive ANYOTHER, drop and stay at E2
   Timeout, increment timeout counter
        If counter is less than N2, send I2 and stay at E2
        If counter is greater than N2, go to E-FAILED
```

```
+---------+
|    E3   | HIP SA established
+---------+

Receive I1, send R1 and stay at E3
Receive I2, process with Birthday check
     if successful, send R2, drop old SA and cycle at E3
     if fail, stay at E3
Receive R1, process with SA and Birthday check
     if successful, send I2 with last datagram, drop old SA
              and go to E2
     if fail, stay at E3
Receive R2, drop and stay at E3

Receive ESP for SA, process and stay at E3
Receive NES, process
     if successful, send NES and stay at E3
     if failed, stay at E3
```

5.4.3 Simplified HIP State Diagram

Receive packets cause a move to new state

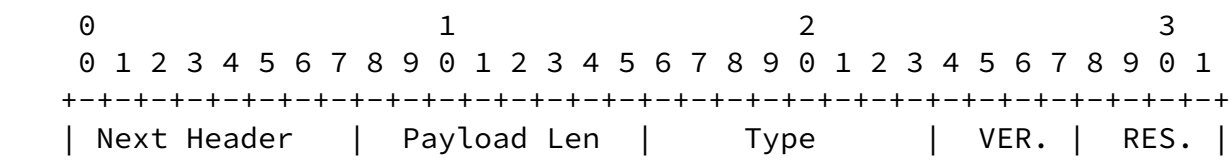


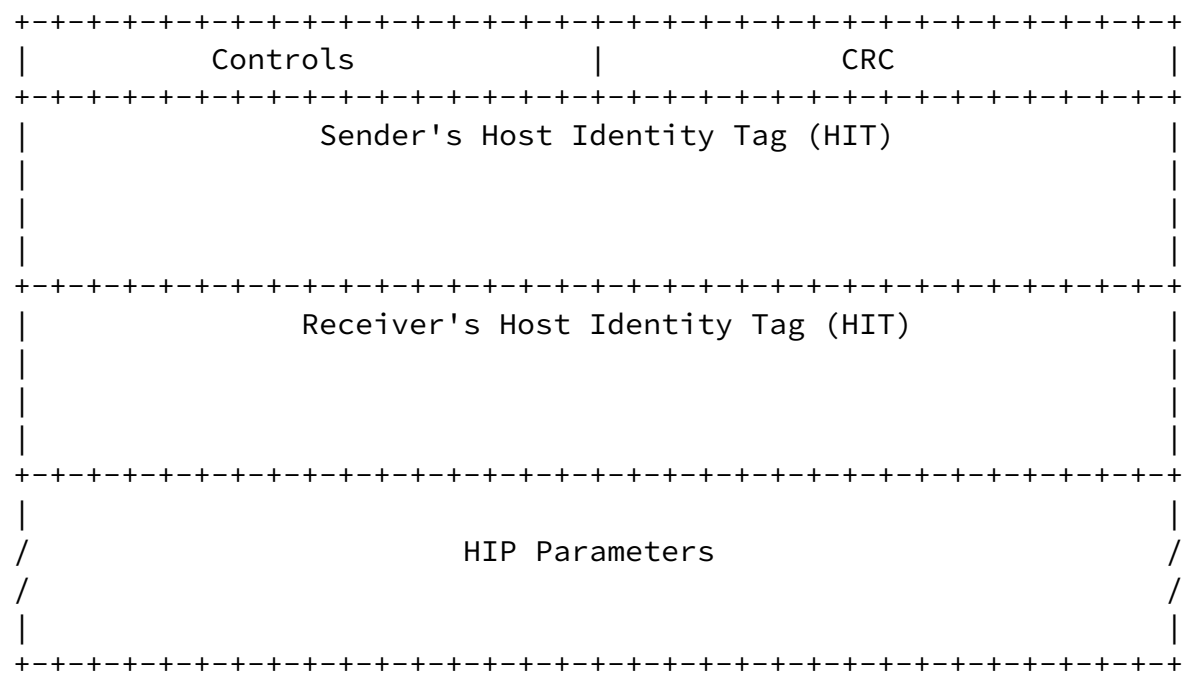


6. Packet formats

6.1 Payload format

All HIP packets start with a fixed header.





The HIP header is logically an IPv6 destination option. However, this document does not describe processing for Next Header values other than decimal 59, IPPROTO_NONE, the IPV6 no next header value. Future documents MAY do so. However, implementations MUST ignore trailing data if a Next Header value is received that is not implemented.

The Header Length field contains the length of the HIP Header and the length of HIP parameters in 8 bytes units, excluding the first 8 bytes. Since all HIP headers MUST contain the sender's and receiver's HIT fields, the minimum value for this field is 4, and conversely, the maximum length of the HIP Parameters field is (255*8)-32 = 2008 bytes.

The Packet Type indicates the HIP packet type. The individual packet types are defined in the relevant sections. If a HIP host receives a HIP packet that contains an unknown packet type, it MUST silently drop the packet.

The HIP Version is four bits. The current version is 1. The version number is expected to be incremented only if there are incompatible changes to the protocol. Most extensions can be handled by defining new packet types, new parameter types, or new controls.

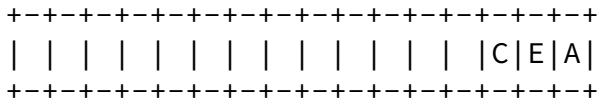
The following four bits are reserved for future use. They MUST be zero when send, and they SHOULD be ignored when handling a received packet.

The HIT fields are always 128 bits (16 bytes) long.

6.1.1 HIP Controls

The HIP control section transfers information about the structure of the packet and capabilities of the host.

The following fields have been defined:



C - Certificate One or more certificate packets (CER) follows this HIP packet (see Section 7.7).

E - ESP sequence numbers The ESP transform requires 64-bit sequence numbers. See Section 11.4 for processing this control.

A - Anonymous If this is set, the senders HI in this packet is anonymous, i.e., one not listed in a directory. Anonymous HIs SHOULD NOT be stored. This control is set in packets R1 and/or I2. The peer receiving an anonymous HI may choose to refuse it by silently dropping the exchange.

The rest of the fields are reserved for future use and MUST be set to zero on sent packets and ignored on received packets.

6.1.2 CRC

The checksum field is located at the same location within the header as the checksum field in UDP packets, enabling hardware assisted checksum generation and verification. Note that since the checksum covers the source and destination addresses in the IP header, it must be recomputed on HIP based NAT boxes.

If IPv6 is used to carry the HIP packet, the pseudo-header [10] contains the source and destination IPv6 addresses, HIP packet length in the pseudo-header length field, a zero field, and the HIP protocol

number (TBD) in the Next Header field. The length field is in bytes and can be calculated from the HIP header length field: (HIP Header Length + 1) * 8.

In case of using IPv4, the the IPv6 pseudo header format [10] is still used, but in the pseudo-header source and destination addresses are IPv4 addresses expressed in IPv4-in-IPv6 format [3].

6.2 HIP parameters

The HIP Parameters are used to carry the public key associated with the sender's HIT, together with other related security information. The HIP Parameters consists of ordered parameters, encoded in TLV format.

The following parameter types are currently defined.

TLV Type	Length	Data
SPI_LSI	16	Remote's SPI, Remote's LSI.
BIRTHDAY_COOKIE	40	System Boot Counter plus 3 64 bit fields: Random #I, K or random # J, Hash target
DIFFIE_HELLMAN	variable	public key
HIP_TRANSFORM	variable	HIP Encryption Transform
ESP_TRANSFORM	variable	ESP Encryption and Authentication Transform
HOST_ID	variable	Host Identity
HOST_ID_FQDN	variable	Host Identity with Fully Qualified Domain Name
CERT	variable	HI certificate
NES_INFO	XXX	ESP sequence number, Old SPI, New SPI
ENCRYPTED	variable	Encrypted part of I2 or CER packets
HIP_SIGNATURE	variable	Signature of the packet

HIP_SIGNATURE2	variable	Signature of the packet R1

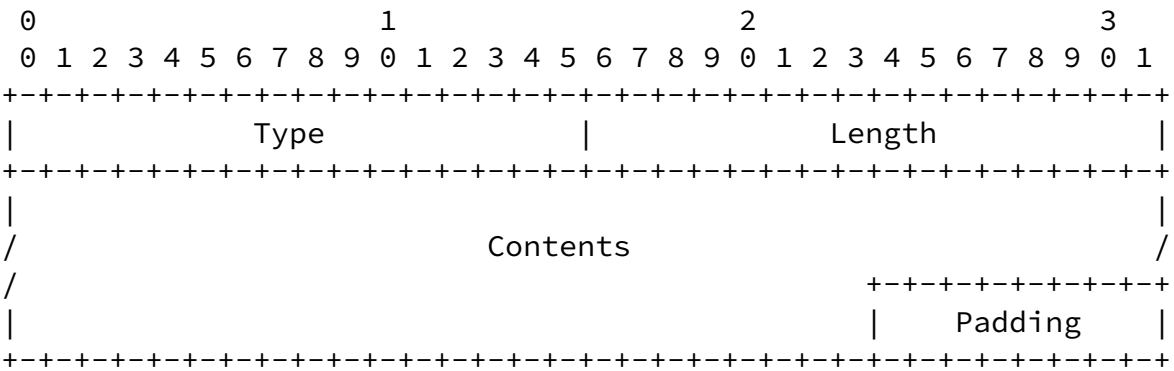
6.3 TLV format

The TLV encoded parameters are described in the following subsections. The type-field value also describes the order of these fields in the packet. The parameters MUST be included into the packet so that the types form an increasing order. If the order does not follow this rule, the packet is considered to be malformed and it MUST be discarded.

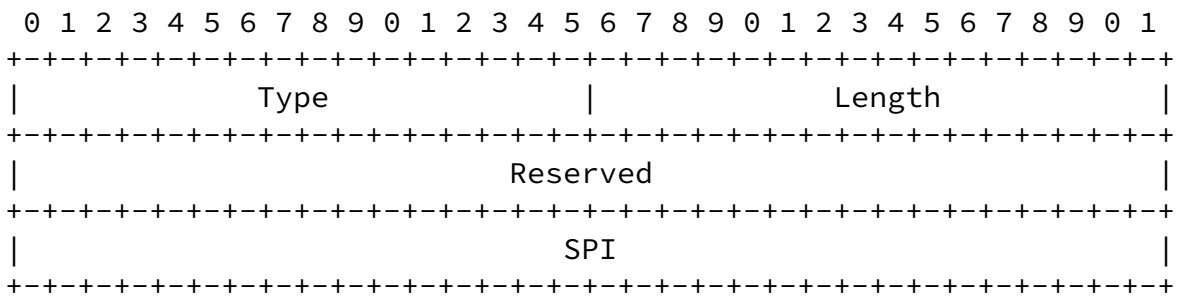
All the TLV parameters have a length which is a multiple of 8 bytes. When needed, padding MUST be added to the end of the parameter so that the total length becomes a multiple of 8 bytes. This rule ensures proper alignment of data. If padding is added, the Length field MUST NOT include the padding.

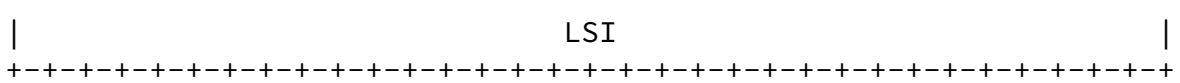
Consequently, the Length field indicates the length of the Contents field (in bytes). The total length of the TLV parameter (including Type, Length, Contents, and Padding) is related to the Length field according to the following formula:

Total Length = 11 + Length - (Length + 3) % 8;



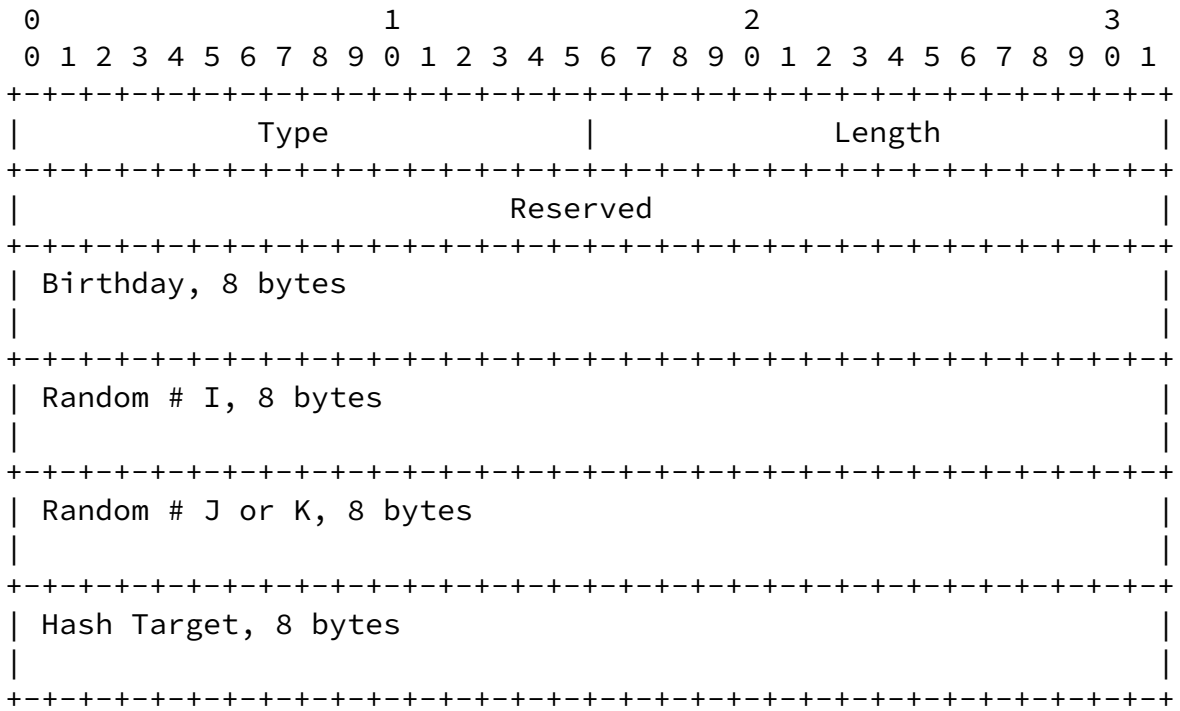
6.3.1 SPI_LSI





Type 1
Length 12
Reserved Zero when sent, ignored when received
SPI Security Parameter Index
LSI Local Scope Identifier

6.3.2 BIRTHDAY_COOKIE

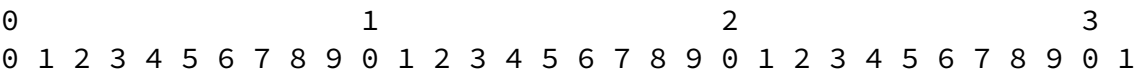


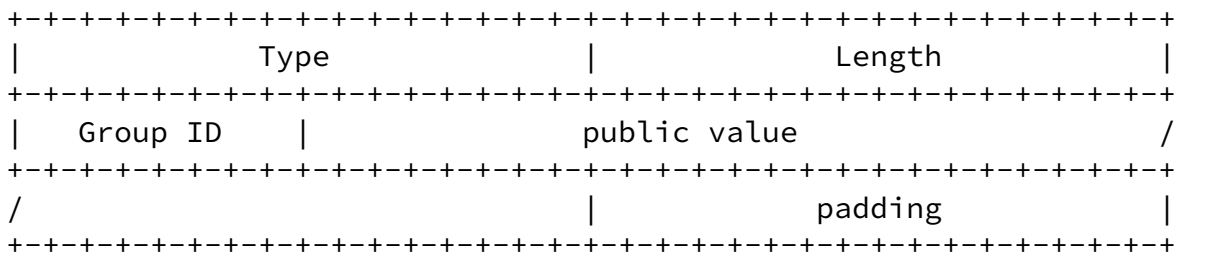
Type 2 (in R1) or 3 (in I2)

Length	36
Reserved	Zero when sent, ignored when received
Birthday	System boot counter
Random # I	random number
K or	K is the number of verified bits (in R1 packet)
Random # J	random number (in I2 packet)
Hash Target	calculated hash value

Birthday, Random #I, K, Random #J, and Hash Target are in network byte order.

6.3.3 DIFFIE_HELLMAN





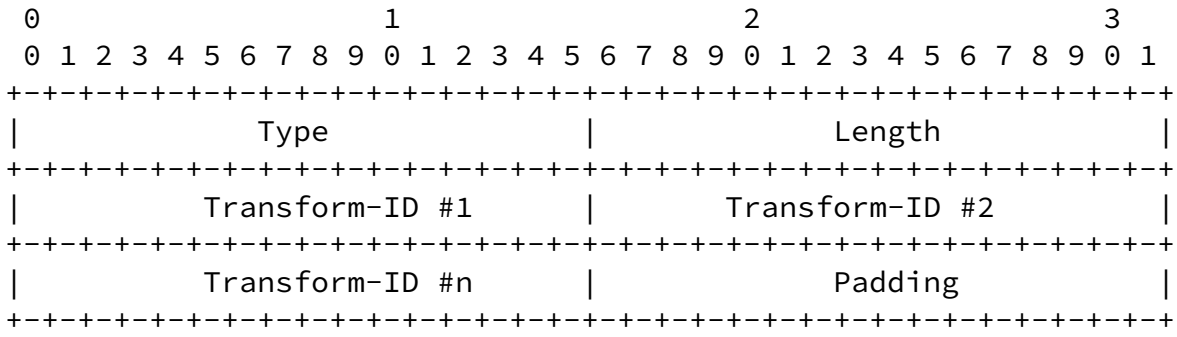
Type	6
Length	length in octets, excluding T and L fields and padding
Group ID	defines values for p and g
public value	

The following Group IDs have been defined:

Group	Value
Reserved	0
OAKLEY well known group 1	1
OAKLEY well known group 2	2
1536-bit MODP group	3
2048-bit MODP group	4
3072-bit MODP group	5
4096-bit MODP group	6
6144-bit MODP group	7
8192-bit MODP group	8

MODP Diffie-Hellman groups are defined in [14]. OAKLEY groups are defined in [7]. The OAKLEY well known group 5 is the same as 1536-bit MODP group.

6.3.4 HIP_TRANSFORM



Type 16
Length length in octets, excluding T and L fields and padding
Transform-ID Defines the HIP Transform to be used

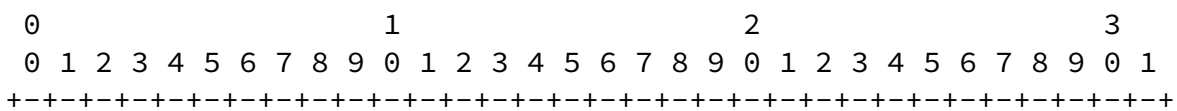
The following encryption algorithms are defined.

Transform-ID	Value
RESERVED	0
ENCR_NULL	1
ENCR_3DES	2
ENCR_AES_128	3

There MUST NOT be more than three (3) HIP Transform-IDs in one HIP transform TLV. The limited number of transforms sets the maximum size of HIP_TRANSFORM TLV. The HIP_TRANSFORM TLV MUST contain at least one of the mandatory Transform-IDs.

Mandatory implementations: ENCR_3DES and ENCR_NULL

6.3.5 ESP_TRANSFORM



```
|             Type              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Suite-ID #1          |          Suite-ID #2          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Suite-ID #n          |             Padding           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type 18
Length length in octets, excluding T and L fields and padding
Suite-ID Defines the ESP Suite to be used

The following Suite-IDs are defined ([15],[16]):

Suite-ID	Value
RESERVED	0
ESP-AES-CBC with HMAC-SHA1	1
ESP-3DES-CBC with HMAC-SHA1	2
ESP-3DES-CBC with HMAC-MD5	3
ESP-BLOWFISH-CBC with HMAC-SHA1	4
ESP-NULL with HMAC-SHA1	5
ESP-NULL with HMAC-MD5	6

There MUST NOT be more than six (6) ESP Suite-IDs in one ESP_TRANSFORM TLV. The limited number of Suite-IDs sets the maximum size of ESP_TRANSFORM TLV. The ESP_TRANSFORM MUST contain at least one of the mandatory Suite-IDs.

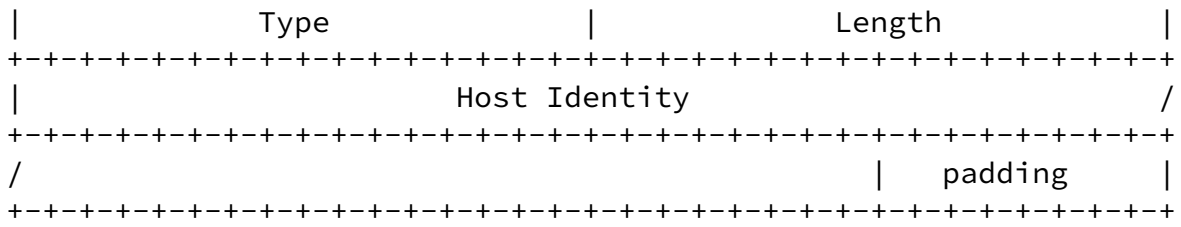
Mandatory implementations: ESP-3DES-CBC with HMAC-SHA1 and ESP-NULL

with HMAC-SHA1

6.3.6 HOST_ID

When the host sends a Host Identity to a peer, it MAY send the identity without any verification information or use certificates to proof the HI. If certificates are sent, they are sent in a separate HIP packet (CER).

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

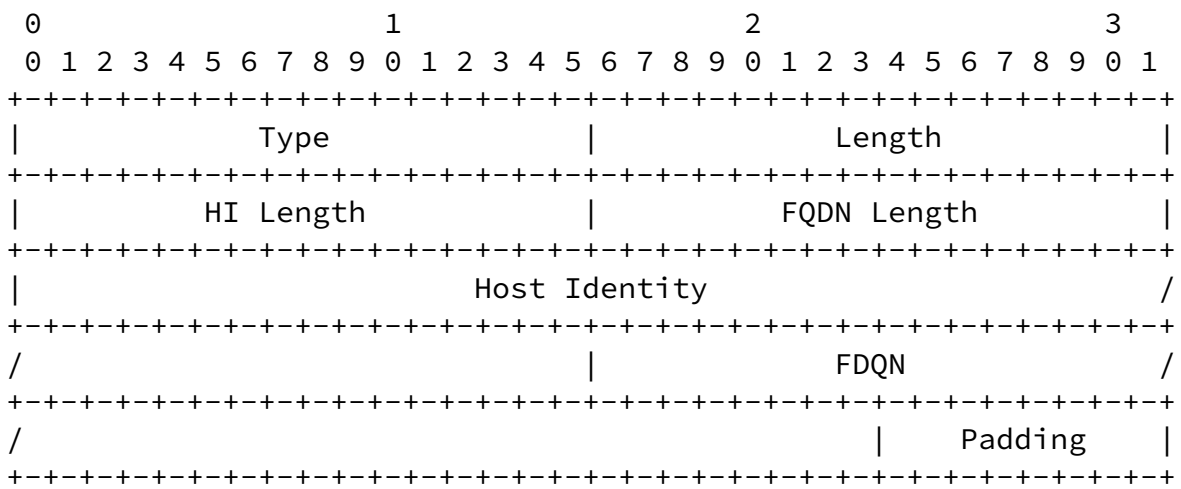


Type 32
Length length in octets, excluding T and L fields and padding
Host Identity actual host identity

The Host Identity is represented in RFC2535 [11] format. The algorithms used in RDATA format are the following:

Algorithms	Values
RESERVED	0
DSA	3 [RFC2536] (REQUIRED)
RSA	5 [RFC3110] (OPTIONAL)

6.3.7 HOST_ID_FQDN



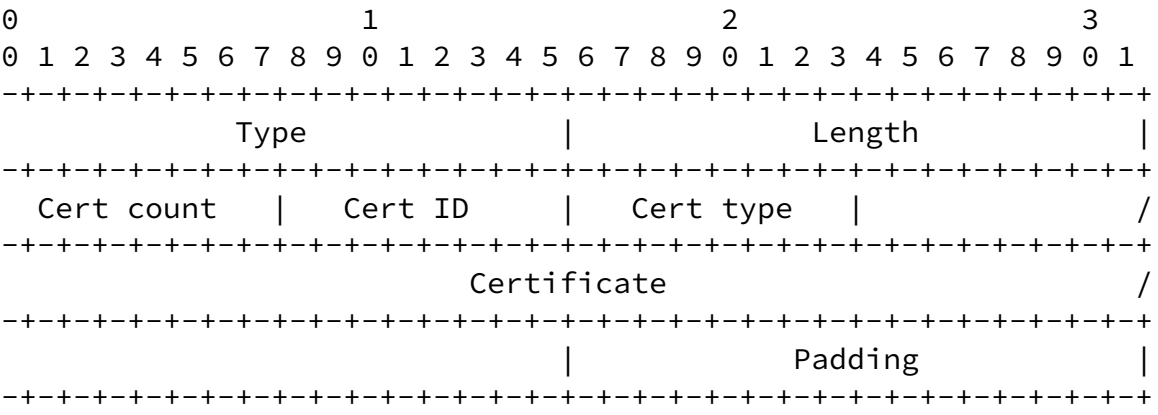
Type 33
Length length in octets, excluding T and L fields and padding
Host Identity
length length of the HI
FQDN length length of the FQDN

```
Host Identity  actual host identity
FQDN           Fully Qualified Domain Name, in the binary format.
```

The Host Identity is represented in RFC2535 [11] format. The format for the FQDN is defined in RFC1035 [1] Sect. 3.1.

If there is no FQDN, the HOST_ID TLV is sent instead.

6.3.8 CERT




```
Type         64
Length       length in octets, excluding T and L fields and padding
Cert count   total count of certificates that are sent, possibly
             in different CER packets
Cert ID      the order number for this certificate
Cert Type    describes the type of the certificate
```

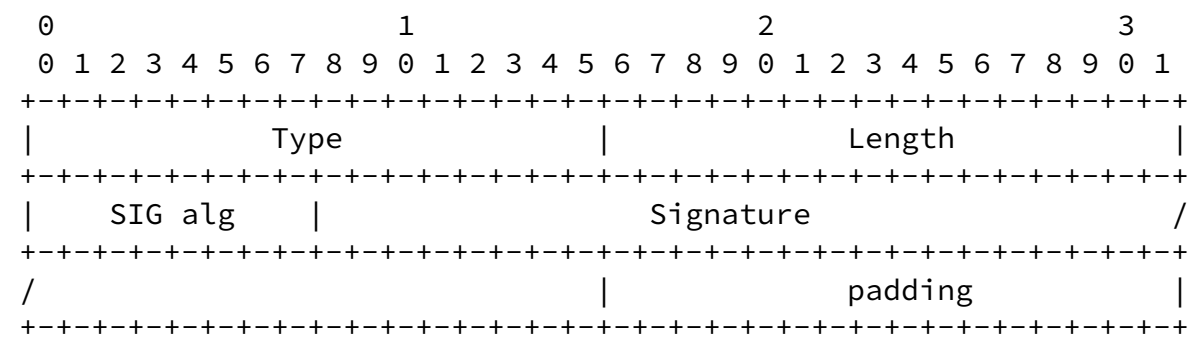
The receiver must know the total number (Cert count) of certificates that it will receive from the sender, related to the R1 or I2. The Cert ID identifies the particular certificate and its order in the certificate chain. The numbering in Cert ID MUST go from 1 to Cert count.

The following certificate types have been identified:

```
Cert format    Type number
X.509 v3       1
```

The encoding format for X.509v3 certificate is defined in [9].

6.3.9 HIP_SIGNATURE



Type	65534 (2^16-2)
Length	length in octets, excluding T and L fields and padding
SIG alg	Signature algorithm
Signature	the signature is calculated over the HIP packet, excluding the HIP_SIGNATURE TLV field. The checksum field MUST be set to zero and the HIP header length in the HIP common header MUST be calculated to the beginning of the HIP_SIGNATURE TLV when the signature is calculated.

Signature calculation and verification process:

Packet sender:

1. Create the HIP packet without the HIP_SIGNATURE TLV

2. Calculate the length field in the HIP header

3. Compute the signature

4. Add the HIP_SIGNATURE TLV to the packet

5. Recalculate the length field in the HIP header

Packet receiver:

1. Verify the HIP header length field

2. Save the HIP_SIGNATURE TLV and remove it from the packet

3. Recalculate the HIP packet length in the HIP header and zero checksum field.

4. Compute the signature and verify it against the received signature

The signature algorithms are defined in Section 6.3.5. The signature in the Signature field is encoded using the proper method depending on the signature algorithm (e.g. in case of DSA, according to [12]).

The verification can use either the HI received from a HIP packet, the HI from a DNS query, if the FQDN has been received either in the HOST_ID_FQDN or in the CER packet, or one reveived by some other means.

6.3.10 HIP_SIGNATURE_2

The TLV structure is the same as in Section 6.3.9. The fields are:

Type	65533 (2^16-3)
Length	length in octets, excluding T and L fields and padding
SIG alg	Signature algorithm
Signature	the signature is calculated over the R1 packet, excluding the HIP_SIGNATURE_2 TLV field. Initiator's HIT and Checksum field MUST be set to zero and the HIP packet length in the HIP header MUST be calculated to the beginning of the HIP_SIGNATURE_2 TLV when the signature is calculated.

Zeroing the Initiator's HIT makes it possible to create R1 packets beforehand to minimize the effects of possible DoS attacks.

Signature calculation and verification process: see the process in Section 6.3.9 HIP_SIGNATURE. Just replace the HIP_SIGNATURE with HIP_SIGNATURE_2 and zero Initiator's HIT at the receiver's end-point.

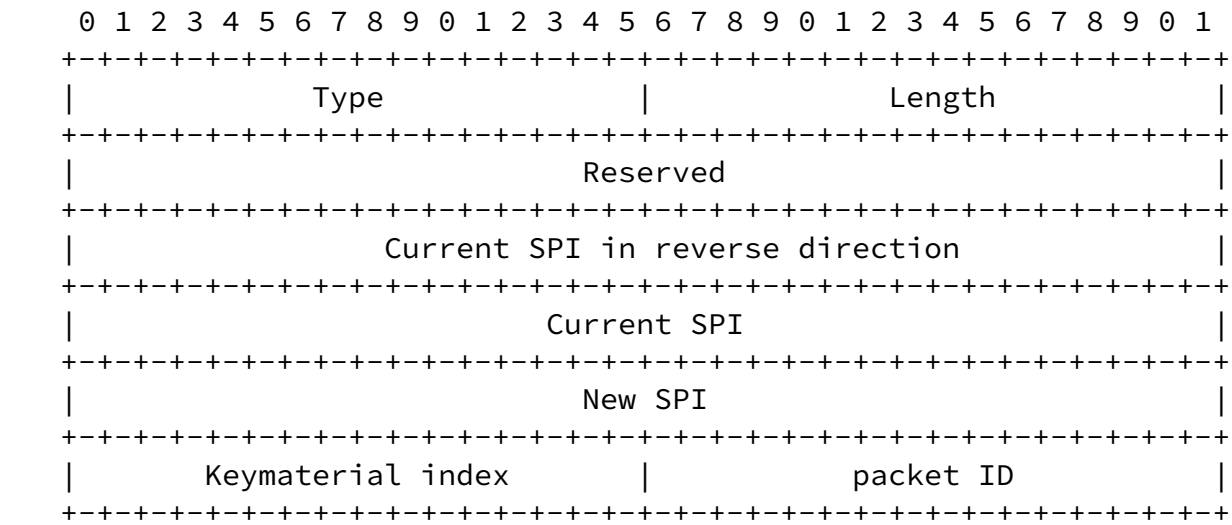
The signature algorithms are defined in Section 6.3.5. The signature in the Signature field is encoded using the proper method depending on the signature algorithm (e.g. in case of DSA, according to [12]).

The verification can use either the HI received from a HIP packet, the HI from a DNS query, if the FQDN has been received either in the HOST_ID_FQDN or in the CER packet, or one reveived by some other means.

6.3.11 NES_INFO

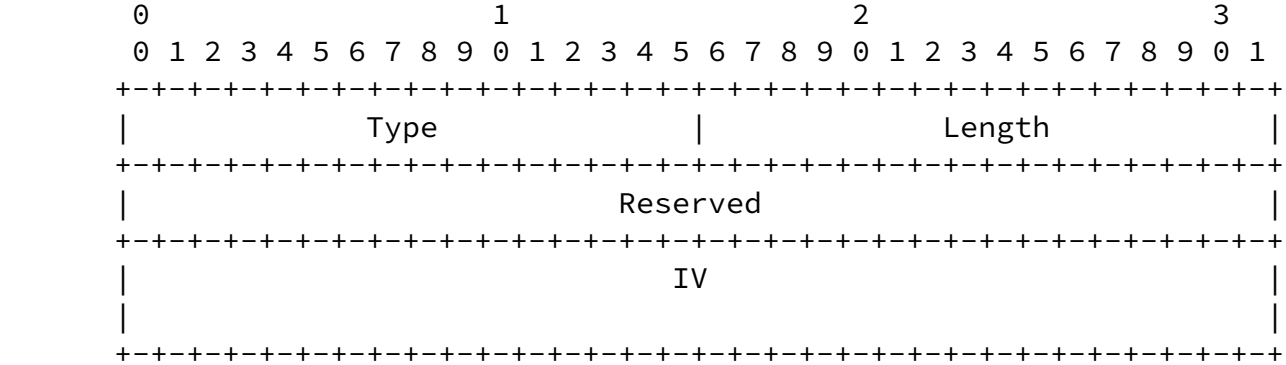
[XXX: The contents of the NES_INFO payload are subject to change,

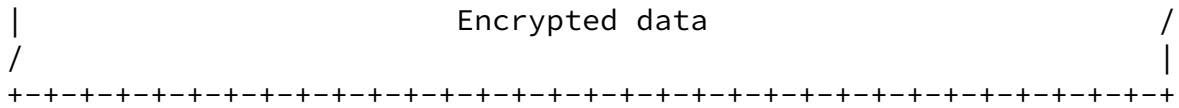
since it is desireable to unify the NES and REA functionality. However, the details of that need to be worked out.]



Type 4
Length length in octets, excluding T and L fields
ESP sequence number
Old SPI
New SPI

6.3.12 ENCRYPTED





Type	20
Length	length in octets, excluding T and L fields
Reserved	zero when sent, ignored when received
IV	Initialization vector, if needed, zero otherwise
Encrypted data	the data is encrypted using an encryption algorithm as defined in HIP transform

The encrypted data is in TLV format itself. Consequently, the first fields in the contents are Type and Length.

7. HIP Packets

There are eight basic HIP packets. Four are for the base HIP exchange,one is for rekeying, one is a broadcast for use when there is no IP addressing (e.g., before DHCP exchange), one is used to send certificates and one is for sending unencrypted data.

Packets consist of the fixed header as described in Section 6.1, followed by the parameters. The parameter part in turn consists of zero or more TLV coded parameters.

In addition to the base packets, some other packets may be defined later in separate standards. E.g. the mobility and multihoming management support is not included in this base specification.

Packet representation uses the following operations:

()	parameter
x{y}	operation x on content y
<x>i	x exists i times
[]	optional parameter
x|y	x or y

An OPTIONAL upper layer payload MAY follow the HIP header. The payload proto field in the header indicates if there is additional data following the HIP header. The P-bit in the control field of the HIP packet header indicates whether the sender is capable of sending and receiving this additional data. The HIP packet, however, MUST NOT

be fragmented. This limits the size of the possible additional data in the packet.

7.1 I1 - the HIP Initiator packet

The HIP header values for the I1 packet:

```
    Type = 1
    SRC HIT = Initiator's HIT
    DST HIT = Responder's HIT, or NULL

   IP(HIP())
```

The I1 packet contains only the fixed HIP header.

Valid control bits: None

The Initiator gets the Responder's HIT either from a DNS lookup of the responder's FQDN, from some other repository, or from a local table. If the initiator does not know the responder's HIT, it may

attempt anonymous mode by using NULL (all zeros) as the responder's HIT.

Since this packet is so easy to spoof even if it were signed, no attempt is made to add to its generation or processing cost.

Implementation MUST be able to handle a storm of reveived I1 packets, discarding those with common content that arrive within a small time delta.

7.2 R1 - the HIP Responder packet

The HIP header values for the R1 packet:

```
   Header:
     Packet Type = 2
     SRC HIT = Responder's HIT
     DST HIT = Initiator's HIT

   IP ( HIP ( BIRTHDAY_COOKIE,
              DIFFIE_HELLMAN,
```

```
                HIP_TRANSFORM,
                ESP_TRANSFORM,
                ( HOST_ID | HOST_ID_FQDN ),
                HIP_SIGNATURE_2 ) )
```

Valid control bits: C, A

The R1 packet may be followed by one or more CER packets. In this case, the C-bit in the control field MUST be set.

If the Responder has multiple HIs, the HIT used MUST match Initiator's request. If the Initiator used anonymous mode, the Responder may select freely among its HIs.

The Initiator HIT MUST match the one received in I1. If the R1 is a response to an ESP packet with an unknown SPI, the Initiator HIT SHOULD be zero.

The Birthday is a reboot count used to manage state reestablishment when one peer rebooted or timed out its SA.

The Cookie contains random I and difficulty K. K is number of bits that the Initiator must match get zero in the puzzle.

The Diffie-Hellman value is ephemeral, but can be reused over a number of connections. In fact, as a defense against I1 storms, an implementation MAY use the same Diffie-Hellman value for a period of

time, for example, 15 minutes. By using a small number of different Cookies for a given Diffie-Hellman value, the R1 packets can be pre-computed and delivered as quickly as I1 packets arrive. A scavenger process should clean up unused DHs and Cookies.

The HIP_TRANSFORM contains the encryption algorithms supported by the responder to protect the HI exchange, in order of preference. All implementations MUST support the 3DES [8] transform.

The ESP_TRANSFORM contains the ESP modes supported by the responder, in order of preference. All implementations MUST support 3DES [8] with HMAC-SHA-1-96 [4].

The SIG is calculated over the whole HIP envelope, after setting the

Initiator HIT and header checksum temporarily to zero. This allows the Responder to use precomputed R1s. The Initiator SHOULD validate this SIG. It SHOULD check that the HI received matches with the one expected, if any.

7.3 I2 - the HIP Second Initiator packet

The HIP header values for the I2 packet:

```
     Type = 3
     SRC HIT = Initiator's HIT
     DST HIT = Responder's HIT

   IP ( HIP ( SPI_LSI,
              BIRTHDAY_COOKIE,
              DIFFIE_HELLMAN,
              HIP_TRANSFORM,
              ESP_TRANSFORM,
              ENCRYPTED { HOST_ID | HOST_ID_FQDN },
              HIP_SIGNATURE ) )
```

Valid control bits: C, E, A

The HITs used MUST match the ones used previously.

The Birthday is a reboot count used to manage state reestablishment when one peer rebooted or timed out its SA.

The Cookie contains I from R1 and the computed J. The low order K bits of the SHA-1(I | ... | J) MUST match be zero.

The Diffie-Hellman value is ephemeral. If precomputed, a scavenger process should clean up unused DHs.

The HIP_TRANSFORM contains the encryption used to protect the HI exchange selected by the initiator. All implementations MUST support the 3DES transform.

The Initiator's HI is encrypted using the HIP_TRANSFORM. The keying material is derived from the Diffie-Hellman exchanged as defined in Section 9.

The ESP_TRANSFORM contains the ESP mode selected by the initiator. All implementations MUST support 3DES [8] with HMAC-SHA-1-96 [4].

The HIP SIG is calculated over whole HIP envelope. The Responder MUST validate this SIG. It MAY use either the HI in the packet or the HI acquired by some other means.

7.4 R2 - the HIP Second Responder packet

The HIP header values for the R2 packet:

```
     Packet Type = 4
     SRC HIT = Responder's HIT
     DST HIT = Initiator's HIT

   IP ( HIP ( SPI_LSI,
              HIP_SIGNATURE ) )
```

Valid control bits: E

The signature is calculated over whole HIP envelope. The Initiator MUST validate this signature.

7.5 NES - the HIP New SPI Packet

The HIP New SPI Packet serves three functions. Firstly, it provides the peer system with a new SPI to use when sending packets. Secondly, it optionally provides a new Diffie-Hellman key to produce new keying material. Thirdly, it provides any intermediate system with the mapping of the old SPI to the new one. This is important to systems like NATs [20] that use SPIs to maintain address translation state.

The new SPI Packet is a HIP packet with SPI and D-H in the HIP payload. The HIP packet contains the current ESP Sequence Number and SPI to provide DoS and replay protection.

The HIP header values for the NES packet:

```
Packet Type = 5
SRC HIT = Sender's HIT
DST HIT = Recipients's HIT

IP ( HIP ( [ DIFFIE_HELLMAN, ] NES_INFO , HIP_SIGNATURE ) )
```

Valid control bits: None

During the life of an SA established by HIP, one of the hosts may need to reset the Sequence Number to one (to prevent wrapping) and rekey. The reason for rekeying might be an approaching sequence number wrap in ESP, or a local policy on use of a key. A new SPI or rekeying ends the current SAs and starts a new ones on both peers. Intermediate systems that use the SPI will have to inspect HIP packets for a HIP New SPI packet. The packet is signed for the benefit of the Intermediate systems.

This packet has a potential DoS attack of a packet within the replay window and proper SPI, but a malformed signature. Implementations MUST recognize when they are under attack and manage the attack. If it is still receiving ESP packets with increasing Sequence Numbers, the NES packets are obviously attacks and can be ignored.

Since intermediate systems may need the new SPI values, the contents of this packet cannot be encrypted.

Intermediate systems that use the SPI will have to inspect ALL HIP packets for a NES packet. This is a potential DoS attack against the Intermediate system, as the signature processing may be relatively expensive. A further step against attack for the Intermediate systems is to implement ESP's replay protection of windowing the sequence number. This requires the intermediate system to track ALL ESP packets to follow the Sequence Number.

7.6 BOS - the HIP Bootstrap Packet

In some situations, an initiator may not be able to learn of a responder's information from DNS or another repository. Some examples of this are DHCP and NetBios servers. Thus, a packet is needed to provide information that would otherwise be gleaned from a repository. This HIP packet is either self-signed in applications like SoHo, or from a trust anchor in large private or public deployments. This packet MAY be broadcasted in IPv4 or multicasted to the all hosts multicast group in IPv6. The packet MUST NOT be sent more often than once in every second. Implementations MAY ignore received BOS packets.

The HIP header values for the BOS packet:

```
Packet Type = 7
SRC HIT = Announcer's HIT
DST HIT = NULL

IP ( HIP ( ( HOST_ID | HOST_ID_FQDN ), HIP_SIGNATURE ) )
```

The BOS packet may be followed by a CER packet if the HI is signed. In this case, the C-bit in the control field MUST be set. If the BOS packet is broadcasted or multicasted, the following CER packet(s) MUST be broadcasted or multicasted to the same multicast group and scope, respectively.

Valid control bits: C, A

7.7 CER - the HIP Certificate Packet

The Optional CER packets over the Announcer's HI by a higher level authority known to the Recipient is an alternative method for the Recipient to trust the Announcer's HI (over DNSSEC or PKI).

The HIP header values for CER packet:

```
Packet Type = 8
SRC HIT = Announcer's HIT
DST HIT = Recipients's HIT

IP ( HIP ( ENCRYPTED { <CERT>i }, HIP_SIGNATURE ) )
```

Valid control bits: None

Certificates in the CER packet MAY be encrypted. The encryption algorithm is provided in the HIP transform of the previous (R1 or I2) packet.

7.8 PAYLOAD - the HIP Payload Packet

The HIP header values for the PAYLOAD packet:

```
Packet Type = 64

IP ( HIP ( ), payload )
```

Valid control bits: None

Payload Proto field in the Header MUST be set to correspond the correct protocol number of the payload.

The PAYLOAD packet is used to carry a non-ESP protected data. By

usign HIP header we ensure interoperability with NAT and other middle boxes.

Processing rules of the PAYLOAD packet are the following:

Receiving: if there is an existing HIP security association with the given HITs, and the IP addresses match the IP addresses associated with the HITs, pass the packet to the upper layer, associated with metadata indicating that the packet was NOT integrity or confidentiality protected.

Sending: if the IPsec SPD defines BYPASS for a given destination HIT, send it with the PAYLOAD packet. Otherwise use the ESP as specified in the SPD.

8. Packet processing

[XXX: This section is currently in its very beginning. It needs much more text.]

8.1 R1 Management

All compliant implementations MUST produce R1 packets. An R1 packet MAY be precomputed. An R1 packet MAY be reused for time Delta T. R1 information MUST not be discarded until Delta S after T. Time S is the delay needed for the last I2 to arrive back to the responder. A spoofed I1 can result in an R1 attack on a system. An R1 sender MUST have a mechanism to rate limit R1s to an address.

8.2 Processing NES packets

The ESP Sequence Number and current SPI are included to provide replay protection for the receiving peer. The old SA MUST NOT be deleted until all ESP packets with a lower Sequence Number have been received and processed, or a reasonable time has elapsed (to account for lost packets). If the Sequence Number is the replay window is greater than the number in the NES packet, the NES packet MUST be ignored. If the SPI number does not match with an existing SPI number used, the NES packet must be ignored.

The peer that initiates a New SPI exchange MUST include a Diffie-Hellmen key. Its peer MUST respond with a New SPI packet, an MAY include a Diffie-Hellman key if the receiving system's policy is to increase the new KEYMAT by changing its key pair.

When a host receives a New SPI Packet with a Diffie-Hellman, its next ESP packet MUST use the KEYMAT generated by the new Kij. The sending host MUST expect at least a replay window worth of ESP packets using the old Kij. Out of order delivery could result in needing the old Kij after packets start arriving using the new SA's Kij. Once past the rekeying start, the sending host can drop the old SA and its Kij.

The first packet sent by the receiving system MUST be a HIP New SPI packet. It MAY also include a datagram, using the new SAs. This packet supplies the new SPI for the rekeying system, which cannot send any packets until it receives this packet. If it does not receive a HIP New SPI packet within a reasonable round trip delta, it MUST assume it or the HIP Rekey packet was lost and MAY resend the HIP New SPI packet or renegotiate HIP as if in a reboot condition. The choice is a local policy decision.

This packet MAY contain a Diffie-Hellman key, if the receiving system's policy is to increase the new KEYMAT by changing its key

pair.

9. HIP KEYMAT

HIP keying material is derived from the Diffie-Hellman Kij produced during the base HIP exchange. The initiator has Kij during the creation of the I2 packet, and the responder has Kij once it receives the I2 packet. This is why I2 can already contain encrypted information.

The KEYMAT is derived by feeding Kij and the HITs into the following operation; the | operation denotes concatenation.

```
KEYMAT = K1 | K2 | K3 | ...
      where

K1   = SHA-1( Kij | sort(HIT-I | HIT-R) | 0x01 )
```

```
K2   = SHA-1( Kij | K1 | 0x02 )
K3   = SHA-1( Kij | K2 | 0x03 )
...
K255 = SHA-1( Kij | K254 | 0xff )
K256 = SHA-1( Kij | K255 | 0x00 )
etc.
```

Sort(HIT-I | HIT-R) is defined as the numeric network byte order comparison of the HITs, with lower HIT preceding higher HIT, resulting in the concatenation of the HITs in the said order. The initial keys are drawn sequentially in the following order:

HIP Initiator key

HIP Responder key (currently unused)

Initiator ESP key

Initiator AUTH key

Responder ESP key

Responder AUTH key

The number of bits drawn for a given algorithm is the "natural" size of the keys. For the manatory algorithms, the following sizes apply:

3DES 192 bits

SHA-1 160 bits

NULL 0 bits

Subsequent rekeys without Diffie-Hellman just requre drawing out more sets of ESP keys. In the situation where Kij is the result of a HIP rekey exchange with Diffie-Hellman, there is only the need from one set of ESP keys, without the HIP keys. These are then the only keys taken from the KEYMAT.

10. HIP Fragmentation Support

A HIP implementation must support IP fragmentation / reassembly. Fragment reassembly MUST be implemented in both IPv4 and IPv6, but fragment generation MUST be implemented only in IPv4 (IPv4 stacks and networks will usually do this by default) and SHOULD be implemented in IPv6. In the IPv6 world, the minimum MTU is larger, 1280 bytes, than in the IPv4 world. The larger MTU size is usually sufficient for most HIP packets, and therefore fragment generation may not be needed. If a host expects to send HIP packets that are larger than the minimum IPv6 MTU, it MUST implement fragment generation even for IPv6.

In the IPv4 world, HIP packets may encounter low MTUs along their routed path. Since HIP does not provide a mechanism to use multiple IP datagrams for a single HIP packet, support of path MTU discovery does not bring any value to HIP in the IPv4 world. HIP aware NAT systems MUST perform any IPv4 reassembly/fragmentation.

All HIP implementations MUST employ a reassembly algorithm that is sufficiently resistant against DoS attacks.

11. ESP with HIP

HIP sets up a pair of Security Associations (SA) to enable ESP in an end-to-end manner that can span addressing realms (i.e. across NATs). This is accomplished through the various informations that are exchanged within HIP. Since HIP is designed for host usage, not for gateways, only ESP transport mode is supported with HIP. The SA is not bound to an IP address; all internal control of the SA is by the HIT and LSI. Thus a host can easily change its address using Mobile IP, DHCP, PPP, or IPv6 readdressing and still maintain the SAs. And since the transports are bound to the SA (LSI or HIT), any active transport is also maintained. Thus, real world conditions like loss of a PPP connection and its reestablishment or a mobile cell change will not require a HIP negotiation or disruption of transport services.

Since HIP does not negotiate any lifetimes, all lifetimes are local policy. The only lifetimes a HIP implementation MUST support are sequence number rollover (for replay protection), and SA timeout. An SA times out if no packets are received using that SA. The default timeout value is 15 minutes. Implementations MAY support lifetimes for the various ESP transforms. Note that HIP does not offer any service comparable with IKE's Quick Mode. A Diffie-Hellman calculation is needed for each rekeying.

11.1 Security Association Management

An SA is indexed by the 2 SPIs and 2 HITs (both HITs since a system can have more than one HIT). An inactivity timer is recommended for all SAs. If the state dictates the deletion of an SA, a timer is set to allow for any late arriving packets. The SA MUST include the I that created it for replay detection.

11.2 Security Parameters Index (SPI)

The SPIs in ESP provide a simple compression of the HIP data from all packets after the HIP exchange. This does require a per HIT- pair Security Association (and SPI), and a decrease of policy granularity over other Key Management Protocols like IKE.

When a host rekeys, it gets a new SPI from its partner.

11.3 Supported Transforms

All HIP implementations MUST support 3DES [8] and HMAC-SHA-1-96 [4]. If the Initiator does not support any of the transforms offered by

the Responder in the R1 HIP packet, it MUST use 3DES and HMAC-SHA-1-96 and state so in the I2 HIP packet.

In addition to 3DES, all implementations MUST implement the ESP NULL encryption and authentication algorithms. These algoritms are provided mainly for debugging purposes, and SHOULD NOT be used in production environments. The default configuration in implementations MUST be to reject NULL encryption or authentication.

11.4 Sequence Number

The Sequence Number field is MANDATORY in ESP. Anti-replay protection MUST be used in an ESP SA established with HIP.

This means that each host MUST rekey before its sequence number reaches 2^32. Note that in HIP rekeying, unlike IKE rekeying, only one Diffie-Hellman key can be changed, that of the rekeying host. However, if one host rekeys, the other host SHOULD rekey as well.

In some instances, a 32 bit sequence number is inadequate. In either the I2 or R2 packets, a peer MAY require that a 64 bit sequence number be used. In this case the higher 32 bits are NOT included in the ESP header, but are simply kept local to both peers. 64 bit sequence numbers must only be used for ciphers that will not be open to cryptoanalysis as a result. AES is one such cipher.

12. HIP Policies

There are a number of variables that will influence the HIP exchanges that each host must support. All HIP implementations MUST support more than one simultaneous HIs, at least one one of which SHOULD be reserved for anonymous usage. Although anonymous HIs will be rarely used as responder HIs, they will be common for initiators. Support for more than two HIs is RECOMMENDED.

Many initiators would want to use a different HI for different responders. The implementations SHOULD provide for an ACL of initiator HIT to responder HIT. This ACL SHOULD also include preferred transform and local lifetimes. For HITs with HAAs, wildcarding SHOULD be supported. Thus if a Community of Interest, like Banking, gets an RAA, a single ACL could be used. A global wildcard would represent the general policy to be used. Policy selection would be from most specific to most general.

The value of K used in the HIP R1 packet can also vary by policy. K should never be greater than 20, but for trusted partners it could be as low as 0.

Responders would need a similar ACL, representing which hosts they accept HIP exchanges, and the preferred transform and local lifetimes. Wildcarding SHOULD be support supported for this ACL also.

13. Security Considerations

HIP is designed to provide secure authentication of hosts and to provide a fast key exchange for IPsec ESP. HIP also attempts to limit the exposure of the host to various denial-of-service and man-in-the-middle attacks. In so doing, HIP itself is subject to its own DoS and MitM attacks that potentially could be more damaging to a host's ability to conduct business as usual.

HIP enabled ESP is IP address independent. This might seem to make it easier for an attacker, but ESP with replay protection is already as well protected as possible, and the removal of the IP address as a check should not increase the exposure of ESP to DoS attacks. Furthermore, this is in line with the forthcoming revision of ESP.

Denial-of-service attacks take advantage of the cost of start of state for a protocol on the responder compared to the 'cheapness' on the initiator. HIP makes no attempt to increase the cost of the start of state on the initiator, but makes an effort to reduce the cost to the responder. This is done by having the responder start the 3-way cookie exchange instead of the initiator, making the HIP protocol 4 packets long. In doing this, packet 2 becomes a 'stock' packet that the responder MAY use many times. The duration of use is a paranoia versus throughput concern. Using the same Diffie- Hellman values and random puzzle I has some risk. This risk needs to be

balanced against a potential storm of HIP I1 packets.

This shifting of the start of state cost to the initiator in creating the I2 HIP packet, presents another DoS attack. The attacker spoofs the I1 HIP packet and the responder sends out the R1 HIP packet. This could conceivably tie up the 'initiator' with evaluating the R1 HIP packet, and creating the I2 HIP packet. The defense against this attack is to simply ignore any R1 packet where a corresponding I1 or ESP data was not sent.

A second form of DoS attack arrives in the I2 HIP packet. Once the attacking initiator has solved the cookie challenge, it can send packets with spoofed IP source addresses with either invalid encrypted HIP payload component or a bad HIP SIG. This would take resources in the responder's part to reach the point to discover that the I2 packet cannot be completely processed. The defense against this attack is after N bad I2 packets, the responder would discard any I2s that contain the given Initiator HIT. Thus will shut down the attack. The attacker would have to request another R1 and use that to launch a new attack. The responder could up the value of K while under attack. On the downside, valid I2s might get dropped too.

A third form of DoS attack is emulating the restart of state after a reboot of one of the partners. To protect against such an attack, a system Birthday is included in the R1 and I2 packets to prove loss of state to a peer. The inclusion of the Birthday creates a very deterministic process for state restart. Any other action is a DoS attack.

A fourth form of DoS attack is emulating the end of state. HIP has no end of state packet. It relies on a local policy timer to end state.

Man-in-the-middle attacks are difficult to defend against, without third-party authentication. A skillful MitM could easily handle all parts of HIP; but HIP indirectly provides the following protection from a MitM attack. If the responder's HI is retrieved from a signed DNS zone, a certificate, or through some other secure means, the initiator can use this to validate the R1 HIP packet.

Likewise, if the initiator's HI is in a secure DNS zone, a trusted certificate, or otherwise securely available, the responder can retrieve it after it gets the I2 HIP packet and validate that. However, since an initiator may choose to use an anonymous HI, it knowingly risks a MitM attack. The responder may choose not to accept a HIP exchange with an anonymous initiator.

New SPIs and rekeying provide another opportunity for an attacker. Replay protection is included to prevent a system from accepting an old new SPI packet. There is still the opening for an attacker to produce a packet with exactly the right Sequence Number and old SPI with a malformed signature, consuming considerable computing resources. All implementations must design to mitigate this attack. If ESP protected datagrams are still being received, there is an obvious attack. If the peer is quiet, it is easier for an attacker to launch this sort of attack, but again, the system should be able to recognize a regular influx of malformed signatures and take some action.

There is a similar attack centered on the readdress packet. Similar defense mechanisms are appropriate here.

Since not all hosts will ever support HIP, ICMP 'Destination Protocol Unreachable' are to be expected and present a DoS attack. Against an Initiator, the attack would look like the responder does not support HIP, but shortly after receiving the ICMP message, the initiator would receive a valid R1 HIP packet. Thus to protect against this attack, an initiator should not react to an ICMP message until a reasonable delta time to get the real responder's R1 HIP packet. A similar attack against the responder is more involved. First an ICMP

message is expected if the I1 was a DoS attack and the real owner of the spoofed IP address does not support HIP. The responder SHOULD NOT act on this ICMP message to remove the minimal state from the R1 HIP packet (if it has one), but wait for either a valid I2 HIP packet or the natural timeout of the R1 HIP packet. This is to allow for a sophisticated attacker that is trying to break up the HIP exchange. Likewise, the initiator should ignore any ICMP message while waiting for an R2 HIP packet, deleting state only after a natural timeout.

14. IANA Considerations

IANA has assigned IP Protocol number TBD to HIP.

15. Acknowledgments

The drive to create HIP came to being after attending the MALLOC meeting at IETF 43. Baiju Patel and Hilarie Orman really gave the original author, Bob Moskowitz, the assist to get HIP beyond 5 paragraphs of ideas. It has matured considerably since the early drafts thanks to extensive input from IETFers. Most importantly, its design goals are articulated and are different from other efforts in this direction. Particular mention goes to the members of the NameSpace Research Group of the IRTF. Noel Chiappa provided the framework for LSIs and Kieth Moore the impetuous to provide resolvability. Steve Deering provided encouragement to keep working, as a solid proposal can act as a proof of ideas for a research group.

Many others contributed; extensive security tips were provided by Steve Bellovin. Rob Austein kept the DNS parts on track. Paul Kocher taught the original authors, Bob Moskowitz, how to make the cookie exchange expensive for the Initiator to respond, but easy for the Responder to validate. Bill Sommerfeld supplied the Birthday concept to simplify reboot management. Rodney Thayer and Hugh Daniels provide extensive feedback. In the early times of this draft, John Gilmore kept Bob Moskowitz challenged to provide something of value.

During the later stages of this document, when the editing baton was transfered to Pekka Nikander, the input from the early implementors were invaluable. Without having actual implementations, this document would not be on the level it is now.

References

[1] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

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

[3] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[4] Madson, C. and R. Glenn, "The Use of HMAC-SHA-1-96 within ESP and AH", RFC 2404, November 1998.

[5] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[6] Maughan, D., Schneider, M. and M. Schertler, "Internet Security Association and Key Management Protocol (ISAKMP)", RFC 2408, November 1998.

[7] Orman, H., "The OAKLEY Key Determination Protocol", RFC 2412, November 1998.

[8] Pereira, R. and R. Adams, "The ESP CBC-Mode Cipher Algorithms", RFC 2451, November 1998.

[9] Housley, R., Ford, W., Polk, T. and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 2459, January 1999.

[10] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[11] Eastlake, D., "Domain Name System Security Extensions", RFC 2535, March 1999.

[12] Eastlake, D., "DSA KEYs and SIGs in the Domain Name System (DNS)", RFC 2536, March 1999.

[13] Vixie, P., "Extension Mechanisms for DNS (EDNS0)", RFC 2671, August 1999.

[14] Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, May 2003.

[15] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol",

draft-ietf-ipsec-ikev2-08 (work in progress), June 2003.

[16] Bellovin, S. and W. Aiello, "Just Fast Keying (JFK)", draft-ietf-ipsec-jfk-04 (work in progress), July 2002.

[17] NIST, "FIPS PUB 180-1: Secure Hash Standard", April 1995.

[18] Moskowitz, R. and P. Nikander, "Host Identity Protocol Architecture", draft-moskowitz-hip-arch-03 (work in progress), May 2003.

[19] Moskowitz, R. and P. Nikander, "Using Domain Name System (DNS) with Host Identity Protocol (HIP)", draft-nikander-hip-dns-00 (work in progress), June 2003.

[20] Moskowitz, R., "Host Identity Payload Implementation", draft-moskowitz-hip-impl-02 (work in progress), January 2001.

[21] Crosby, SA. and DS. Wallach, "Denial of Service via Algorithmic Complexity Attacks", in Proceedings of Usenix Security Symposium 2003, Washington, DC., August 2003.

Authors' Addresses

Robert Moskowitz
ICSAlabs, a Division of TruSecure Corporation
1000 Bent Creek Blvd, Suite 200
Mechanicsburg, PA
USA

EMail: rgm@icsalabs.com

Pekka Nikander
Ericsson Research Nomadic Lab

JORVAS FIN-02420
FINLAND

Phone: +358 9 299 1
EMail: pekka.nikander@nomadiclab.com

Petri Jokela
Ericsson Research Nomadic Lab

JORVAS FIN-02420
FINLAND

Phone: +358 9 299 1
EMail: petri.jokela@nomadiclab.com

Appendix A. Backwards compatibility API issues

Tom Henderson floated again the thought that that the LSI could be completely local and does not need to be exchanged. Applications continue to use IP addresses in socket calls, and kernel does whatever NATting (including application NATting) is required. It was pointed out that this approach was going to be prone to some kinds of data flows escaping the HIP protection, unless the local housekeeping in an implementation was especially good. Example: FTP opens control connection to IP address. One or both parties move. FTP later opens data connection to the old IP address. Kernel must identify that the application really means to connect to the host that was previously at that IP address -- but obviously if the old address is reused by another host, this becomes difficult.

Related to this, the discussion also opened up the question of DNS resolution. Should the HIT/LSI be returned to applications as a (spoofed) address in the resolution process, allowing apps to use the socket API with HIT or LSI values instead of an IP address? While

this seems to be the original intention of LSIs, there are a couple of difficulties especially in the IPv4 case:

How does kernel know whether value being passed in a socket call is an IP address or an LSI? The fact that a name resolver library gave an application an LSI is no guarantee that the application will use that information in its socket call. It may also have cached some IP address from before or received an IP address as side information. This difficulty is now relieved as the LSIs are constrained to the well-known private subnet space.

Handing an LSI may confuse legacy applications that assume that what is being passed to them is an IP address. Good examples of this are diagnostic tools such as dig and ping. The conclusion is that HIP should most not be used with diagnostic applications.

What does kernel do with an LSI that it cannot map to an address based on information that it has locally cached?

It seems that some modification to the resolver library (to explicitly convey HIP information rather than spoofing IP addresses), as well as modifications to socket API to explicitly let the kernel know that the application is HIP aware, are the cleanest long-term solution, but what to do about legacy applications?? -- still partially an open issue. The HUT team has been considering these problems.

Appendix B. Probabilities of HIT collisions

The birthday paradox sets a bound for the expectation of collisions. It is based on the square root of the number of values. A 64-bit hash, then, would put the chances of a collision at 50-50 with 2^32 hosts (4 billion). A 1% chance of collision would occur in a population of 640M and a .001% collision chance in a 20M population. A 128 bit hash will have the same .001% collision chance in a 9x10^16 population.

Appendix C. Probabilities in the cookie calculation

A question: Is it guaranteed that the Initiator is able to solve the puzzle in this way when the K value is large?

Answer: No, it is not guaranteed. But it is not guaranteed even in the old mechanism, since the Initiator may start far away from J and arrive to J after far too many steps. If we wanted to make sure that the Initiator finds a value, we would need to give some hint of a suitable J, and I don't think we want to do that.

In general, if we model the hash function with a random function, the probability that one iteration gives are result with K zero bits is 2^-K. Thus, the probablity that one iteration does *not* give K zero bits is (1 - 2^-K). Consequently, the probablity that 2^K iterations does not give K zero bits is (1 - 2^-K)^(2^K).

Since my calculus starts to be rusty, I made a small experiment and found out that

```
lim     (1 - 2^-k)^(2^k)     = 0.36788
k->inf

lim     (1 - 2^-k)^(2^(k+1)) = 0.13534
k->inf

lim     (1 - 2^-k)^(2^(k+2)) = 0.01832
k->inf

lim     (1 - 2^-k)^(2^(k+3)) = 0.000335
k->inf
```

Thus, if hash functions were random functions, we would need about 2^(K+3) iterations to make sure that the probability of a failure is less than 1% (actually less than 0.04%). Now, since my perhaps flawed understanding of hash functions is that they are "flatter" than random functions, 2^(K+3) is probably an overkill. OTOH, the currently suggested 2^K is clearly too little. The draft has been changed to read 2^(K+2).

Appendix D. Using responder cookies

As mentioned in Section 4.1.1, the responder may delay state creation and still reject most spoofed I2s by using a number of pre-calculated R1s and a local selection function. This appendix defines one possible implementation in detail. The purpose of this appendix is to give the implementators an idea on how to implement the mechanism. The method described in this appendix SHOULD NOT be used in any real implementation. If the implementation is based on this appendix, it SHOULD contain some local modification that makes an attacker's task harder.

The basic idea is to create a cheap, varying local mapping function f:

```
   f( IP-I, IP-R, HIT-I, HIT-R ) -> cookie-index
```

That is, given the Initiators and Responders IP addresses and HITs, the function returns an index to a cookie. When processing an I1, the cookie is embedded in an pre-computed R1, and the Responder simply sends that particular R1 to the Initiator. When processing an I2, the cookie may still be embedded in the R1, or the R1 may be depracated (and replaced with a new one), but the cookie is still there. If the received cookie does not match with the R1 or saved cookie, the I2 is simply dropped. That prevents the Initiator from generating spoofed I2s with a probability that depends on the number of pre-computed R1s.

As a concrete example, let us assume that the Responder has an array of R1s. Each slot in the array contains a timestamp, an R1, and an old cookie that was sent in the previous R1 that occupied that particular slot. The Responder replaces one R1 in the array every few minutes, thereby replacing all the R1s gradually.

To create a varying mapping function, the Responder generates a random number every few minutes. The octets in the IP addresses and HITs are XORed together, and finally the result is XORed with the random number. Using pseudo-code, the function looks like the following.

```
Pre-computation:
    r1 := random number

Index computation:
    index := r1    XOR hit_r[0] XOR hit_r[1] XOR ... XOR hit_r[15]
    index := index XOR hit_i[0] XOR hit_i[1] XOR ... XOR hit_i[15]
    index := index XOR  ip_r[0] XOR  ip_r[1] XOR ... XOR  ip_r[15]
    index := index XOR  ip_i[0] XOR  ip_i[1] XOR ... XOR  ip_i[15]
```

The index gives the slot used in the array.

It is possible that an Initator receives an I1, and while it is computing I2, the Responder deprecates an R1 and/or chooses a new random number for the mapping function. Therefore the Responder must remember the cookies used in deprecated R1s and the previous random number.

To check an received I2, the Responder can use a simple algorithm, expressed in pseudo-code as follows.

```
  If I2.hit_r does not match my_hits, drop the packet.

  index := compute_index(current_random_number, I2)
  If current_cookie[index]  == I2.cookie, go to cookie check.
  If previous_cookie[index] == I2.cookie, go to cookie check.

  index := compute_index(previous_random_number, I2)
  If current_cookie[index]  == I2.cookie, go to cookie check.
  If previous_cookie[index] == I2.cookie, go to cookie check.

  Drop packet.

cookie_check:
  V := Ltrunc( SHA-1( I2.I, I2.hit_i, I2.hit_r, I2.J ), K )
  if V != 0, drop the packet.
```

Whenever the Responder receives an I2 that fails on the index check, it can simply drop the packet on the floor and forget about it. New I2s with the same or other spoofed parameters will get dropped with a reasonable probability and minimal effort.

If a Responder receives an I2 that passes the index check but fails on the puzzle check, it should create a state indicating this. After two or three failures the Responder should cease checking the puzzle but drop the packets directly. This saves the Responder from the SHA-1 calculations. Such block should not last long, however, or there would be a danger that a legitimite Initiator could be blocked from getting connections.

A key for the success of the defined scheme is that the mapping function must be considerably cheaper than computing SHA-1. It also

must detect any changes in the IP addresses, and preferably most changes in the HITs. Checking the HITs is not that essential, though, since HITs are included in the cookie computation, too.

The effectivity of the method can be varied by varying the size of the array containing pre-computed R1s. If the array is large, the

probability that an I2 with a spoofed IP address or HIT happens to map to the same slot is fairly slow. However, a large array means that each R1 has a fairly long life time, thereby allowing an attacker to utilize one solved puzzle for a longer time.

Intellectual Property Statement

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.